Machine learning and Genetic Algorithms for conformal geometries in design support systems

van Hassel, S.J.F.

Award date:
2022

MSc. Thesis

Machine learning and Genetic Algorithms for conformal geometries in design support systems

Eindhoven University of Technology, The Netherlands

S.J.F. (Sebastiaan) van Hassel

Student information

Name:	S.J.F. (Sebastiaan) van Hassel
Student number:	0955797
Graduation date:	13-04-2022

Graduation Committee

Chairman:	dr. H. (Herm) Hofmeyer
Second supervisor:	T. (Tessa) Ezendam MSc.
Third supervisor:	dr. P. (Pieter) Pauwels

April, 19, 2022

CONTENTS

Summary

To optimise both building designs and their underlying design processes, design support systems exist. For domain specific analyses, these systems benefit from a conformal (CF) representation for the Building Spatial Design (BSD). In a conformal representation, for all entities: the vertices of an entity are, if intersecting another entity, only allowed to coincide with this other entity's vertices. This thesis presents research on whether Machine Learning (ML) and Genetic Algorithms (GA) can be used to obtain a conformal geometry for BSDs. For ML, a neural network is trained to learn the complex relation between BSDs and their conformal representations. GAs are first used to find all quad-hexahedrons in the search space, then to find sets of quad-hexahedrons that form the conformal design. A trained ML model does provide outcomes, but not very useful, even with encoding the configuration type of the design. Differently, the GA finds conformal designs for many instances, even for some non-orthogonal designs.

1 INTRODUCTION

In the built environment, a building design process is understood as the creation of a plan to realise a building (Hofmeyer & Davila Delgado 2015). Building design processes are unique and iterative, where multiple disciplines are involved (Haymaker et al. 2004). This makes these processes complex, to be supported in the early stages to improve the final design (Kalay 2004), and to reduce costs of design changes (MacLeamy 2004). Ideally, decisions are made where all requirements of the different disciplines are considered. However, a design team is never able to analyse all possible alternatives to make the related well-founded decisions.

Design support systems exist that (a) suggest building designs, which can be used by the design teams; (b) simulate design processes (via simulations with partial models), to support the teams process-wise. In all such systems, a representation is used for the Building Spatial Design (BSD). And for domain specific analyses, it is very useful to have a so-called conformal (CF) representation for the BSD. In a conformal representation, for all entities it holds: the vertices of an entity are, if intersecting another entity, only allowed to coincide with this other entity's vertices. Figure 1 shows BSD non conformal surfaces (grey), which are transformed into a conformal geometry (blue), here via partitioning of surfaces.

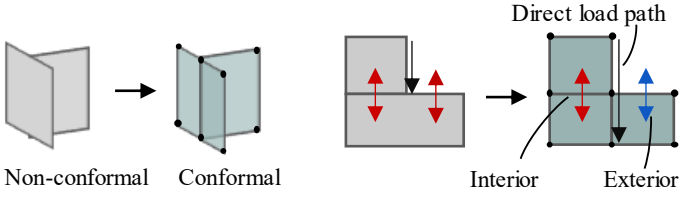


Figure 1. Non-conformal BSD surfaces are transformed into conformal geometries (l); domain related advantages (r).

A conformal geometry is convenient for discipline-specific models. For example, for structural design, proper finite element meshing and loading can be done more correctly with a conformal geometry. And if the conformal geometry is used as a blueprint for structural elements, also more efficient load paths may arise (black arrow). Thermal modelling of the building can be done more correctly too, as surfaces are either fully internal or external, and so fluxes (the red and blue arrows in Fig. 1) for each surface can be correctly described.

Current procedural methods to transform a BSD into a conformal design work for orthogonal, and sometimes for specific non-orthogonal designs (Ezendam 2021), whereas realistic building designs may be less specific non-orthogonal.

Therefore, research is presented on whether alternatively Machine Learning and Genetic Algorithms can be used to transform orthogonal and non-orthogonal BSDs into conformal designs.

1.1 *Related work*

For this research, an existing design support system is used (Boonstra et al. 2018), which is developed to design and optimise building spatial designs with respect to different disciplines. In this support system, two main transformations take place. The first transformation is from a BSD to its conformal representation. The second transformation is from a conformal model to a discipline-specific model. The research here is focused on the first transformation. The BSD is defined as an arrangement of spaces. For orthogonal designs, the spaces are limited to cuboid shapes. For non-orthogonal designs, the spaces are allowed to be different forms of quad-hexahedrons, as long as the 'walls' are vertical and the 'floors' are horizontal. Spaces of orthogonal designs can be defined either by an origin vector (**s**) and dimension vector (**d**) (Fig. 2a) or by eight corner-vertices (**p**) (Fig. 2b). See also Equations 1, 2 and 3. Non-orthogonal designs are (only) described by the eight corner-vertices.

$$\mathbf{s} = [x\ y\ z]^T; \quad \mathbf{d} = [w\ d\ h]^T; \quad \mathbf{p} = [x\ y\ z]^T \qquad (1, 2, 3)$$

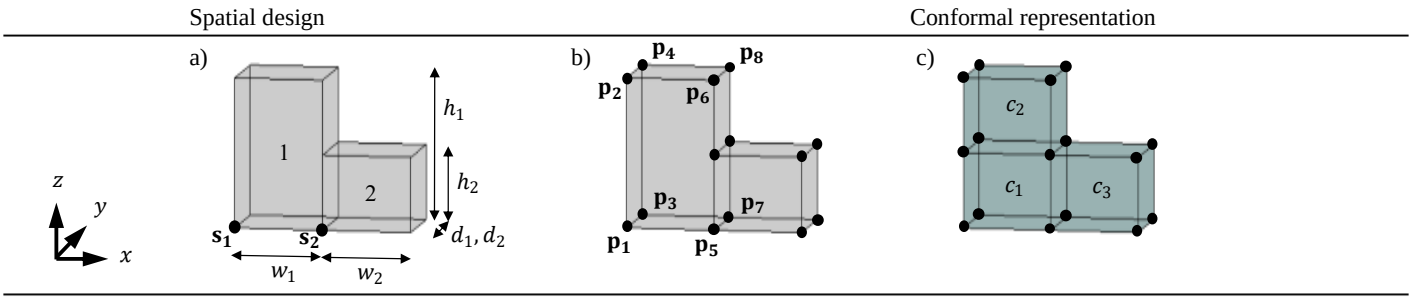


Figure 2. BSD described by (a) origin & dimensions vector, (b) corner-vertices. On the right, (c) its conformal representation.

For orthogonal designs, the conformal geometry is described by cuboids. See figure 2c for an example with three cuboids, where each cuboid is described by eight corner-vertices. For non-orthogonal designs, the conformal geometry uses quad-hexahedrons, which are also described by eight corner-vertices.

1.2 *Methods*

This research investigates Artificial Intelligence (AI) techniques to make spatial designs conformal. First, Machine Learning (ML) is used to make conformal geometry predictions. A dataset is generated via the design support system (Boonstra et al. 2018) with building spatial designs as input (features) and corresponding conformal geometries as output (targets). The dataset is used to train a neural network to make spatial designs conformal. Secondly, two Genetic Algorithms (GAs) are used to make spatial designs conformal. A first algorithm uses space coordinates to generate possible quad-hexahedrons. Then a second algorithm is used to find groups of quad-hexahedrons that form a conformal design. Initially, the research is focused on orthogonal BSDs with two spaces. If one of the methods functions acceptably, more spaces and non-orthogonal geometries are tried.

2 MACHINE LEARNING (ML)

A machine has a level of intelligence if it can learn from external information to improve its own knowledge (Mitchell 1997). The access to quantitative and qualitative data enables a machine to learn certain rules, which can be applied to solve tasks automatically. A machine can learn in three different ways (Goodfellow et al. 2017), namely Supervised, Unsupervised, and by Reinforcement learning. Here, Supervised learning will be used, where the dataset is fixed and the input and associated output are known. The machine learns to predict the output (target) from an input (feature) and the algorithm is trained on the relation between features and targets.

Machine learning (ML) will be used to make orthogonal spatial designs conformal. The machine needs to learn the relation between initial spatial designs and their conformal representations. The ML concept is visualised in Figure 3.

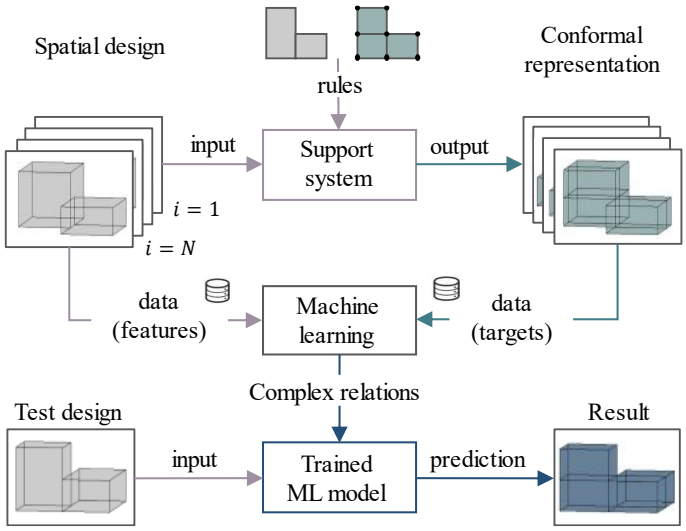


Figure 3. The existing support system generates a dataset of features and targets, which is used to train a machine that predicts the conformal model of new test designs.

Using the existing support system (Boonstra et al. 2018), a transformation is carried out N times, each time for a different spatial design. As such, data is generated at the input and output of the design process.

As shown in Figure 3, the input data (features) and the output data (targets) are extracted from the support system, placed in a dataset, and used for the machine learning process. As data is accessible and the input features and output targets are known, supervised ML is used.

Finally, a trained model is created that predicts the output by only the input variables. In this way, the algorithm that generates a conformal design is replaced by a trained ML model. The trained ML model is evaluated by several studies, but first the dataset and machine learning model need to be defined, as presented in the next sections.

2.1 *Dataset*

Within the first part of this research, using ML, only orthogonal BSDs are considered. Therefore, the conformal representation is defined as an arrangement of cuboids. To start with a basic design, the dataset is based on a building with two spaces, where space 1 is fixed and space 2 is positioned to the right of space 1. Later, space 2 can also be positioned around (front, behind, left) or on top of space 1. For the spatial de-

sign, the dimensions vectors (**d**) of the two spaces are randomly defined. The width h and depth d vary between 4000 and 8000 mm, and the height h varies between 2000 and 6000 mm. The origin vector (**s**) of space 1 is always defined as [10000 10000 0], and the origin vector of space 2 is based on its position related to space 1. However, the coordinates in the origin vector need to be defined, such that the two spaces are adjacent.

The resulting BSDs are classified in five different variants for space 2 relative to space 1: 'right', 'left', 'front', 'behind', and 'top'. Additionally, each variant has several building types, which are given by the size and orientation of space 2 in relation with space 1, which in turn result in different numbers and configurations of the cuboids. As a result, 27 different building types can be generated for each of the variants: 'right', 'left', 'front', 'behind', and 81 different building types can be generated for the 'top' variant. The number of building types is important in the analysis, which is explained later.

The study starts with 1000 data-points where space 2 is positioned to the right of space 1. For this situation, the input data consists of 12 features and the output data of 168 targets (maximal 7 cuboids). One data-point represents the features and targets of one simulated conformal transformation.

Features (**s** and **d** of BSD) are described by:

$$\mathbf{f} = [\, w_1, d_1, h_1, x_1, y_1, z_1, \dots, z_s] \quad (4)$$

where s = number of the space.

Targets (**p** of CF model) are described by:

$$\mathbf{t} = \left[\, x_{11}, y_{11}, z_{11}, \dots, x_{cp}, y_{cp}, z_{cp} \,\right] \quad (5)$$

where c = number of the cuboid; and p = index of the corner-vertex of cuboid c.

Note that for the quality of the model, relatively few input data is available for many output options. The data is scaled with mean normalisation to reduce the importance of individual features and to improve the training process (Ioffe & Szegedy 2015). Additionally, the data is split into a training set (80%) and testing set (20%).

2.2 *Machine learning model*

A Feed forward Neural Network (FNN) is used to solve the regression task. The architecture of the FNN is shown in Figure 4. The dimensions and origins of the spatial design are allocated in the input layer. The input is connected with one hidden layer to the output. The corner-vertices of the conformal cuboids are allocated in the output layer.

Each layer consists of a number of neurons (or nodes). The neurons of one layer are connected to neurons of another layer and have certain weight values. These weights are initialised by the He function (He et al. 2015) and are adjusted in the training process by using the Adam optimisation function (Kingma & Ba 2014). The goal is to minimise the loss between predicted output and the actual output. The loss is indicated by the Mean Squared Error (MSE), and calculated as follows (Kolodiazhnyi 2020):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{m}\sum_{j=1}^{m}(\mathbf{t}_i[j] - \hat{\mathbf{t}}_i[j])^2\right) \quad (6)$$

where n = number of data-points; m = number of output values per data-point; $\mathbf{t}$ = actual output; and $\hat{\mathbf{t}}$ = predicted output.

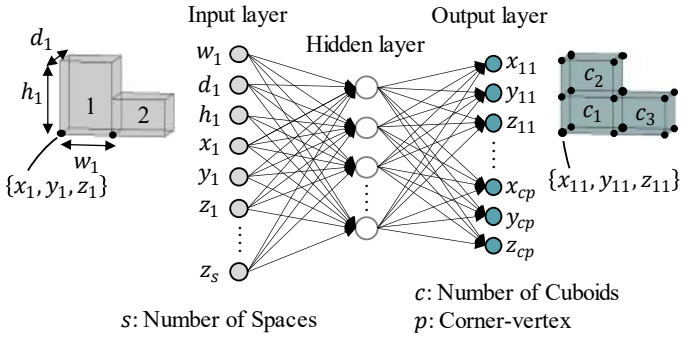


Figure 4. Neural network architecture.

Four hyper-parameters need to be defined in the analysis: number of hidden layers, number of neurons, learning rate and batch size. These parameters are dependent on the dataset and can be changed. In the analysis, one hidden layer will be used and the number of neurons is equal or slightly larger than the number of nodes in the adjacent input or output layers. Additionally, the initial learning rate is set to 0.01 and the batch size is 1/10 of the number of data-points.

2.3 *Analysis*

The performance of the trained ML model is measured by analysing the loss function (MSE) and the prediction for three test buildings. The loss is calculated for the training data during the training process of 12 iterations.

First, test building 'A' (Fig. 6) is used and needs to be made conformal by a trained ML model. Since the test building has two spaces and space 2 is to the right of space 1, the ML model is trained on a dataset with 1000 data-points (800 training, 200 testing), where space 2 is located to the right of space 1. This dataset is called 'Single configuration'. Figure 5 shows the loss during training. The training MSE (Not encoded) reduces only slightly during the 12 iterations. The actual conformal model and prediction of test building 'A' are visualised in Figures 6b and 6c. The predicted CF model (Not encoded dataset) consists of a set of hexahedrons, but bears hardly a resemblance with the actual output.

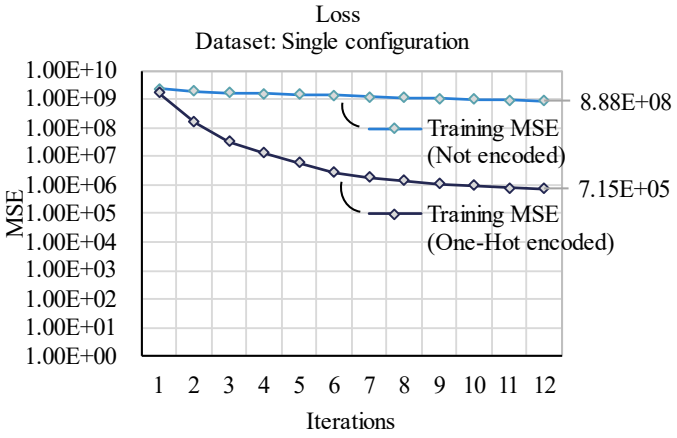


Figure 5. Training loss of ML model. Trained on Not & One-hot encoded dataset with single configuration, where space 2 is to the right of space 1.

Thus it can be concluded that One-hot encoding of the different output types improves the model significantly. And as such the improved model can predict a conformal model, based on a BSD with two spaces, where space 2 is at the right of space 1. However, it has to be tested to what extent the trained model is able to predict alternative designs.

Therefore, test building 'B', the 180 degrees rotated version of test building 'A' is used as an alternative design, see Figure 7a. Using still the 'Single configuration' dataset, the position of space 2, now to the left of space 1, is not included in the training dataset. Therefore, the trained machine learning model has never seen this configuration and is not able to make proper predictions, as can be seen in Figure 7c.

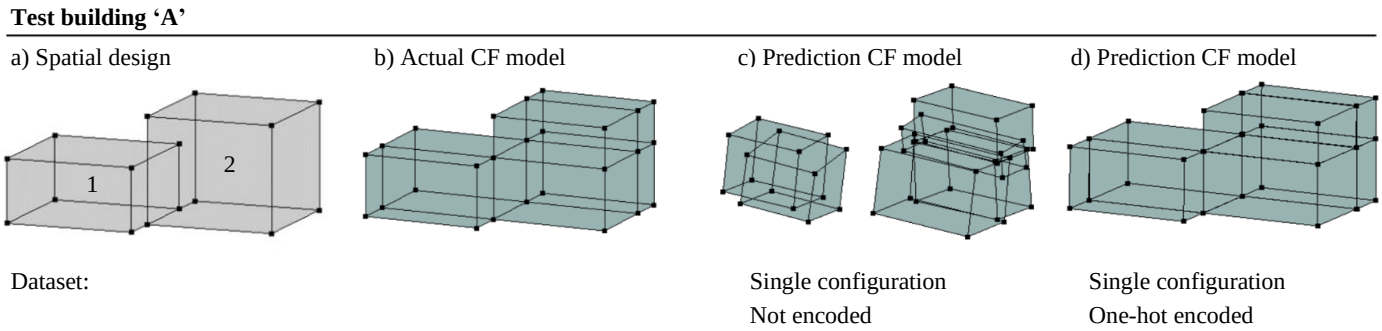


Figure 6. Test building 'A': (a) spatial design, (b) actual CF model & (c,d) predictions.

To improve the model, it is realised that typically the number of input values is larger than the number of output values. But here, the input consists of only 12 features and needs to predict 168 output values. Therefore, extra information is applied to the dataset by adding the type of the design as a feature. The 27 different building types of the variant 'right' (space 2 at the right of space 1) are included in the input features by using One-hot encoding. One-hot encoding is used to convert categorical data into numerical data and prevents ranking of the different building types. As shown in Figure 5 (One-hot encoded), now the loss is reduced significantly. In addition, the prediction using the One-hot encoded dataset has many similarities with the actual conformal model, see Figure 6d.

Clearly, a trained ML model can only make predictions of building configurations that are used in the training process. Hence the need for elaborate diversity in the training set. A new dataset is defined, 'All configurations', and is created with 1000 data-points for each variant (left, right, etc.). A new ML model is trained and the loss during 12 iterations is shown in Figure 8. Consequently, the conformal prediction of test building 'B' using the One-hot encoded dataset on all configurations has more similarities with the actual CF model, see Figure 7d.

Lastly, test building 'C' is considered, which is the two-space building with the most cuboids (10) in the conformal model, see Figure 9.

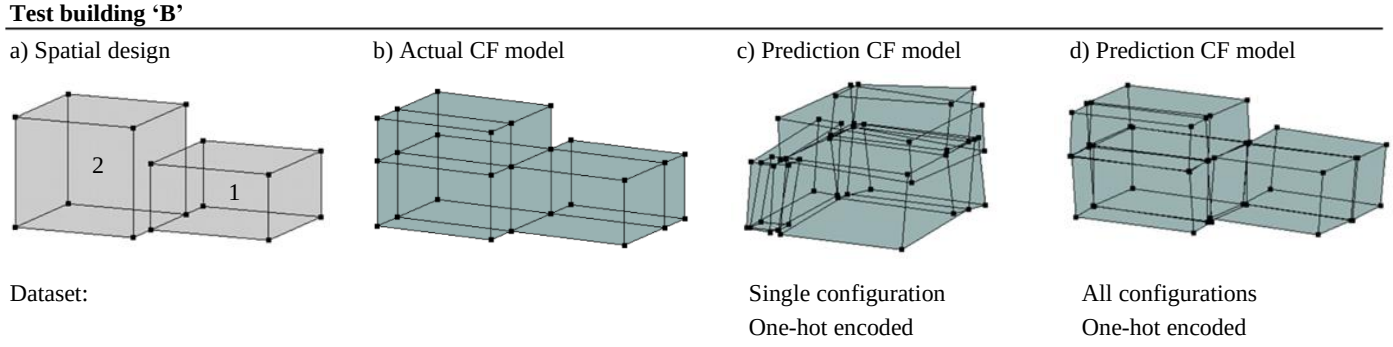


Figure 7. Test building 'B': (a) spatial design, (b) actual CF model & (c,d) predictions.

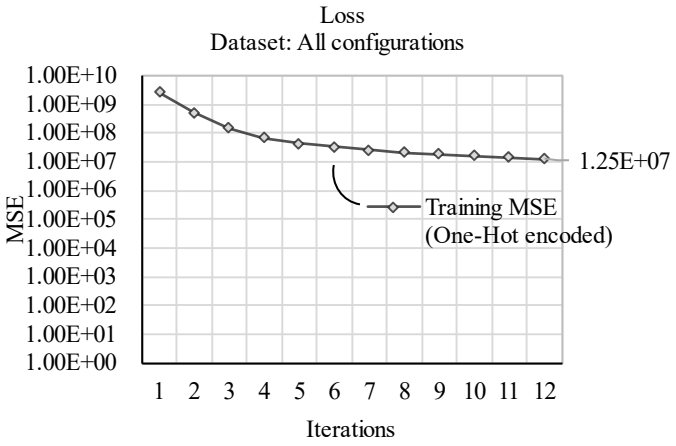


Figure 8. Training loss of ML model. Trained on One-hot encoded dataset with all configurations.

As shown in Figure 9c, the shape of the conformal model and the number cuboids are predicted by the trained ML model. However, there is still an error, and the predictions are numerically not perfect. The inaccurate predicted corner-vertices lead to non-cubic results.

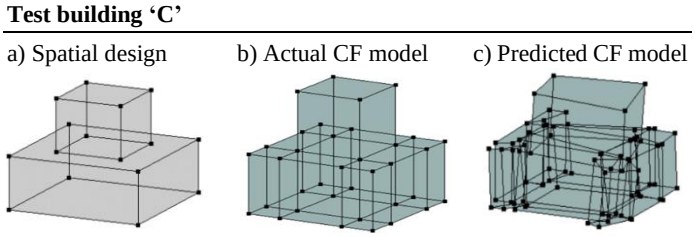


Figure 9. Test building 'C': (a) spatial design, (b) actual CF model & (c) prediction. Based on 'All configurations' One-hot encoded dataset.

3 GENETIC ALGORITHMS (GA)

A Genetic Algorithm (GA) is based on the evolution theory of Darwin and applies stochastic search to find one or more optimal solutions in a problem (Holland 1975). Unlike most ML algorithms, GAs are not based on gradient-descent optimisation functions, but based on heuristic methods, and use the process of natural selection, reproduction and mutation (Dudek 2013). According to Stanovov et al. (2017), GAs are beneficial in high-dimensional search spaces.

A GA workflow is illustrated in Figure 10. GAs start with a population of possible solutions (individuals), which evolve to better solutions, and finally one or more optimal solutions are found. After the population is initialised, the process consists of four main operators: fitness, selection, crossover, and mutation, which are repeatedly executed in every generation. The individual solutions are marked with a fitness value that represents the quality of the solution. Thereafter, the fittest solutions are selected and called parents. A crossover is applied on the parents, where information is exchanged from parents to new solutions, called offspring. To maintain diversity and mimic natural selection, mutations are applied on the individual solutions.

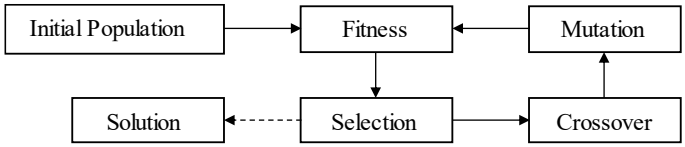


Figure 10. GA workflow.

The GA approach here consists of three steps, namely pre-processing of BSD data, a first GA1 to generate quad-hexahedrons, and a second GA2 to generate the conformal model.

3.1 *Pre-processing of BSD data*

In this section, the corner-vertices (**p**) are used to describe the BSD spaces. Thereafter, a point cloud with vertices (v) is generated by all combinations of existing x, y and z values of the BSD corner-vertices, see Figure 11. Also new non-existing vertices will arise, but vertices outside the spatial design are neglected to reduce computational costs.

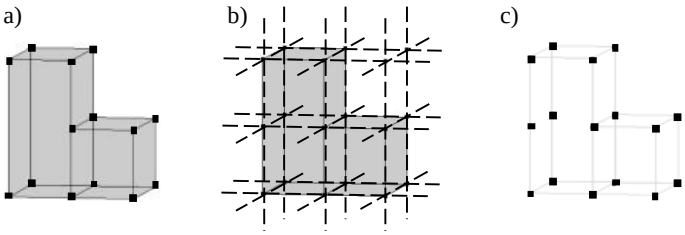
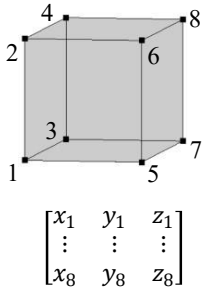


Figure 11. Pre-processing data: (a) BSD corner-vertices, (b) existing x, y, z coordinates, and (c) the final point cloud.

3.2 *GA1: Quad-Hexahedrons*

For the first GA, the initial population consists of N individuals, each created by a combination of 8 vertices. These vertices are chosen randomly from the point cloud and should form a quad-hexahedron. When an individual gains the maximum fitness score, it is a quad-hexahedron and saved for the second GA (GA2). The individuals are represented by an 8×3 matrix, see Figure 12, which is efficient (Wallet et al. 1996).



$$\begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_8 & y_8 & z_8 \end{bmatrix}$$

Orthogonal constraints

$$x_i = x_{i+1} = x_{i+2} = x_{i+3} \quad \text{for } i = 1,5$$
$$y_i = y_{i+1} = y_{i+4} = y_{i+5} \quad \text{for } i = 1,3$$
$$z_i = z_{i+2} = z_{i+4} = z_{i+6} \quad \text{for } i = 1,2$$

Non-orthogonal constraints:

$$x_i = x_{i+1} \quad \text{for } i = 1,3,5,7$$
$$y_i = y_{i+1} \quad \text{for } i = 1,3,5,7$$
$$z_i = z_{i+2} = z_{i+4} = z_{i+6} \quad \text{for } i = 1,2$$

Figure 12. Representation of quad-hexahedron and fitness constraints.

The quad-hexahedrons are labelled with a fitness score based on their shape. For orthogonal shapes, a perfect rectangular cuboid gets the maximum fitness

score. For non-orthogonal shapes, a perfect quad-hexahedron with vertical walls and horizontal floors obtains the maximum fitness score, see Figure 12. Additionally, non-convex and triangular solutions are eliminated from the population, as well as quad-hexahedrons that are located in more than one space of the BSD.

All individuals in the population are used as parents and undergo a crossover to create four new individuals, i.e. offspring. Preselection is used and preserves diversity in the population, according to Wong (2015) and Cavicchio (1970). The parents compete with their offspring, and the best 2 individuals are chosen for the next generation. Crossover is alternately applied in three ways: X, Y, and Z. Crossover X uses one-point crossover with crossover point between the 4th and 5th row of the chromosome matrix. Crossover Y uses a three-point crossover with a crossover point per two rows. Crossover Z uses a seven-point crossover with a crossover point per row.

Mutation is applied at chromosome (quad-hexahedron) and gene (vertex) level. Duplicate quad-hexahedrons in the population are replaced by new unique quad-hexahedrons. Furthermore, duplicate vertices in a chromosome are replaced by new vertices, randomly chosen from the point cloud.

It is difficult to determine whether all necessary quad-hexahedrons are found. Therefore, GA1 is running n generations, until no new unique quad-hexahedron is found. The parameter n is design-specific, and can be adjusted.

3.3 *GA2: Conformal model*

In GA2, first the (number *q*) quad-hexahedrons found in GA1 are used to create a population of candidate conformal models. The initial population consists of *M* individuals based on a cluster of quad-hexahedrons. In GA2, an individual is called quad-cluster, and represented by an array of zeros and ones. The length of this array is equal to the number *q* of found quad-hexahedrons. The ones in the array form the conformal model, see Figure 13 for an example.

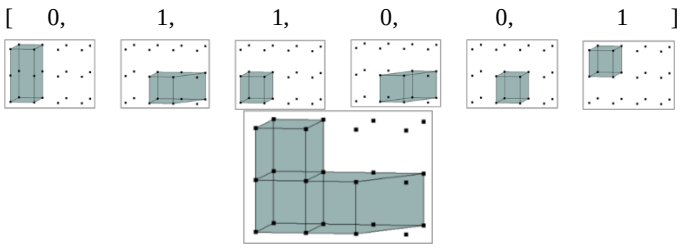


Figure 13. Quad-cluster as an array of bits.

The exact number of quad-hexahedrons needed for the final conformal model is unknown. Therefore, the number of quad-hexahedrons in the initial population ranges from 1% to 100% of the total quad-hexahedrons. The algorithm will find the best number by the evolving generations and the selection of the fittest.

The quad-clusters are assessed by a multi-objective fitness function. First, the volume of the conformal model (V_{CF}) needs to be equal to the volume of the BSD (V_{BSD}), indicated by a fitness volume (f_V), see Equation 7. Secondly, a conformal model should not have intersections between individual quad-hexahedrons, indicated by a fitness intersect (f_I), see Equation 8. Intersections are represented by line-line and line-polygon intersections. The fitness values range from zero (best) to one (worst) and are calculated by equations 7 and 8.

$$f_V = \frac{abs(V_{CF} - V_{BSD})}{\sum_{i=1}^{q}(V_{Qi}) - V_{BSD}}; \qquad f_I = \frac{n_{Inter}}{(n_{Inter} + n_{Links})} \qquad (7,8)$$

Where V_Q indicates the volume of each quad-hexahedron. Furthermore, n_{Inter} indicates the number of quad-hexahedron pairs that intersect, and n_{Links} indicates the number of quad-hexahedron pairs that perfectly link. Perfectly linked means that two adjacent quad-hexahedrons are connected by four corner-vertices without overlapping faces or edges.

Due to the multi-objective fitness function, the design solutions cannot be simply separated by a single fitness score. Therefore, the non-dominated sorting principle (Kalyanmoy 2002) is used to select the best individuals, at the same time maintaining diversity in the population. Finally, the best half of the individuals that are not or less dominated by others will be selected for a next generation. The selected parents undergo a uniform crossover and create new offspring. Uniform crossover outperforms the one-point and two-point crossover approaches, Syswerda (1989). When two parents have the same binary code, the offspring will also be the same. Therefore, random offspring generation (ROG) (Rocha 1999) is applied, where the parents are randomly shuffled to create new off-spring, but with the same number of quad-hexahedrons.

The intermediate population with parents and offspring is mutated, where one or more genes are flipped. The mutation probability of the population is set to 50%.

The evolution process stops when a perfect conformal model is found. This is achieved when the volume of the CF model is equal to the volume of the BSD ($f_V = 0$), and there are no intersections present ($f_I = 0$).

3.4 *Analysis*

3.4.1 *Orthogonal (O) designs*

The test buildings ‘A’, ‘B’ and ‘C’ from Section 2.3 are also used for the orthogonal GA analysis, and are now defined as ‘A-O’, ‘B-O’, and ‘C-O’ (‘O’ for Orthogonal). The results are shown in Table 1 and Figure 14.

Table 1. Results orthogonal GA analysis.

Test building	*v*	*N* GA1	*g* GA1	*q* GA1	*M* GA2	*g* GA2	*r* GA2
'A-O'	26	512	200	12	8	5	6
'B-O'	26	512	200	12	8	19	6
'C-O'	36	2048	20,000	37	8	67	10

With v = number of vertices in point cloud; N or M = population size; g = number of generations; q = number of quad-hexahedrons found in GA1; and r = number of quad-hexahedrons in final CF model.

Figure 14 shows the BSD with associated point cloud and the correctly predicted CF model for test buildings 'A-O', 'B-O' and 'C-O'. There are no complications by using the same GA for all these three test buildings. Therefore, different from a trained ML model, a GA is more generally applicable. Test building 'C-O' needs the most generations in GA1 and GA2. This has two reasons. Firstly, the position of space 2 leads to only one quad-hexahedron in space 2 and is difficult to be found in GA1. Secondly, the larger number of quad-hexahedrons found in GA1 (q) and needed in the final conformal model (r) result in more combinations, which leads to more computations and generations.

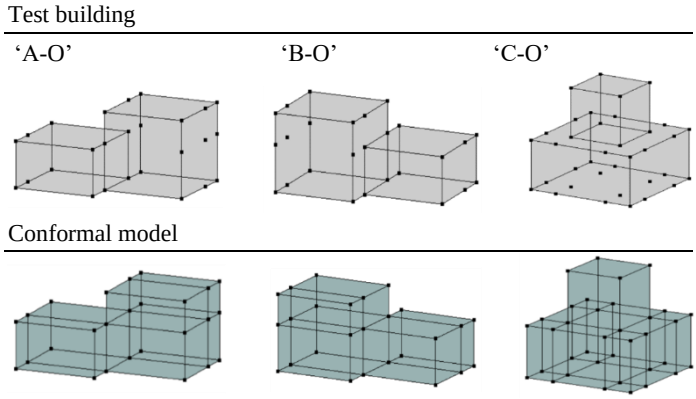


Figure 14. Test buildings 'A-O', 'B-O' & 'C-O'. Top row: BSD with point cloud. Bottom row: conformal models generated by GA.

It is shown that the GA is able to make two-space BSDs conformal. However, a GA is only beneficial if it uses less computational time than a process considering all possible solutions (a brute force technique). All possible combinations C are given by Equation 9.

$$C(q,r) = \frac{q!}{(r!(q-r)!)} \quad (9)$$

For test building 'C-O', deterministic search in GA2 would lead to: $C(37,10)/M = 3.48 \cdot 10^8/8 = 4.35 \cdot 10^7$ generations, if the number of quad-hexahedrons in the CF model is known. Now only 67 generations are needed, which indicates that the GA is helpful.

3.4.2 *Non-orthogonal (NO) designs*

As mentioned earlier, realistic buildings may see non-orthogonal spaces, for which it is challenging to find conformal solutions with existing approaches (Ezendam 2021). Therefore, here a GA will be tested for some test buildings with non-orthogonal shapes. First, test building 'A-NO' is evaluated and has the same number of vertices in the point cloud as test building 'A-O' from Section 3.4.1. Nevertheless, the number of quad-hexahedrons in GA1 is increased to 36 for test building 'A-NO'. Naturally, conformal non-orthogonal designs have more possible combinations, and therefore, show larger computational costs. Figure 15 shows test building 'A-NO' and its correctly GA predicted CF model. The results are listed in Table 2.

Table 2. Results non-orthogonal GA analysis.

Test building	*v*	*N* GA1	*g* GA1	*q* GA1	*M* GA2	*g* GA2	*r* GA2
'A-NO'	26	512	200	36	8	360	6
'B-NO'	29	512	200	60	-	-	-
'C-NO'	36	2048	2000	120	32	4500	6

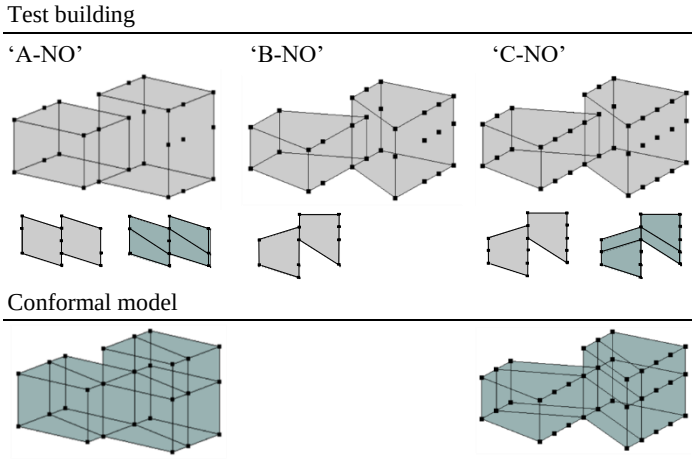


Figure 15. Test buildings 'A-NO', 'B-NO' & 'C-NO'. Top row: BSD with point cloud and floorplan of building. Bottom row: Conformal models, if generated by GA.

Test building 'A-NO' is solved by the presented technique of point cloud generation. However, it is expected that normally the point cloud needs to be expanded to create all necessary quad-hexahedrons. For example, when considering test building 'B-NO', with the standard cloud, the GA is not able to find a CF model. If an extra y coordinate is added, relabelling the design as 'C-NO', see Figure 15, the total number of quad-hexahedrons q increases to 120 and a CF model is found after 4500 generations. Note that deterministic search would lead to: $C(120,6)/M = 3.65 \cdot 10^9/32 = 1.14 \cdot 10^8$ generations.

Nevertheless, generating the conformal representation of 'C-NO' on current hardware still takes hours.

4 DISCUSSION

This research investigated the use of ML techniques and Genetic Algorithms. However, other techniques could also be used, even within the domain of the previously described methods.

Within ML, many other techniques exist. Although, as concluded in this research, ML techniques, where relations or patterns are learned from a dataset, are not suitable to solve the described problem. Optimisation methods, which search for the optimal solution in a complex and unknown domain, could be more applicable. Therefore, ML techniques that use, for example Bayesian optimisation, are recommended to investigate in future work.

For the GA method, the main operators could be changed to improve the evolutionary process. For example, the fitness function, which is an important part of the algorithm. The fitness function is computed every generation, and therefore, should be computed fast. Unfortunately, the current fitness function in GA2 involves large computation times, which leads to unfavourably large evolutionary processes. The fitness function could be enhanced by lowering the computational costs or using more or better formulated objectives. Additionally, after some preliminary research, a combination of two GAs have been selected to not diffuse the second GA with the relatively simple task of the first GA. Nevertheless, future research could reinvestigate the use of a single GA.

There consist three large uncertainties in the GA method. First, whether the generated point cloud is sufficient to create a perfect CF model. A strategy should be developed to generate a point cloud that for all possible situations ensures a solution. Secondly, the uncertainty whether all necessary quad-hexahedrons are found in GA1, to finally create the CF model in GA2. This is one of the disadvantages of using two GAs. GA1 does not have a discrete stopping criteria like GA2, where both the volume and intersection fitness should be zero. Thirdly, whether GA2 converges to an optimal solution. Premature convergence is a common problem in GAs. The individual solutions in the population converge to a suboptimal solution. However, in this research, the solution needs to be optimal to create a perfect conformal geometry. Therefore, some techniques are already implemented to preserve diversity and prevent local convergence, such as mutations, uniform crossovers, ROG, and non-dominated sorting. Nevertheless, GA2 still converges to local optima. Especially, for non-orthogonal designs when the number of possible solutions increases exponentially. The problem enlarges when the ratio between the length of the chromosome (in GA2) and the number of quad-hexahedrons needed in the final conformal model becomes too large. As a result, the applied mutations become ineffective.

5 CONCLUSION

This research investigated the use of ML techniques and Genetic Algorithms for the creation of conformal models in the design and engineering of buildings. These techniques are investigated as an alternative to procedural coding approaches.

A ML model shows to be able to generate conformal designs, but only when the specific topology (e.g. two spaces side by side) is used in the training process. As such, it is not able to develop a reasoning scheme to find conformal solutions for e.g. 3 spaces in a new configuration, unless it is trained for each specific configuration and in combination with One-hot encoding. For the basic example dataset, there are already 189 different configurations, which implies 189 One-hot encodings. When the number of variables increases for more realistic designs, the number of encodings increases exponentially too. As a result, ML cannot be used. Additionally, ML predictions are compromised numerically. There is a slight error in the results, which makes the predicted conformal model non-cubic, as so unusable.

A combination of two GAs is able to create conformal representations of BSDs, at least for the examples tested here. It shows an advantage above deterministic search, for less computations are needed to converge to the optimal solution. In addition, the algorithm is adaptable to solve multiple configurations, and the found conformal geometries are perfect quad-hexahedrons without numerical errors. For orthogonal designs, the number of possible cuboids remains relatively low and the GA is able to transform BSDs with two spaces into conformal representations. The GA is also able to transform non-orthogonal designs, but the number of possible solutions increases exponentially. This leads to unfavourably large computation times.

Eventually, the GA and ML methods provide results, but cannot guarantee optimality.

REFERENCES

Boonstra, S., Van der Blom, K., Hofmeyer, H., Emmerich, M.T.M., Van Schijndel, A.W.M. & De Wilde, P. (2018). Toolbox for super-structured and super-structure free multi-disciplinary building spatial design optimisation. *Advanced Engineering Informatics*, Volume 36: 86-100.

Boonstra, S., van der Blom, K., Hofmeyer,H. Emmerich, M.T.M. (2021). Hybridization of an evolutionary algorithm and simulations of co-evolutionary design processes for early- stage building spatial design optimization. *Automation in Construction*, Volume 124, 103522, ISSN 0926-5805.

Cavicchio, D. J. (1970). *Adaptive search using simulated evolution.* (PhD thesis). Ann Arbor, MI: University of Michigan.

Claessens, D. P., Boonstra, S., & Hofmeyer, H. (2020). Spatial zoning for better structural topology design and performance. *Advanced Engineering Informatics*. Volume 46, October 2020, 101162, ISSN 14740346.

Deb, K. (2002). A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*. Volume 6(2): 182-197.

Dudek, G. (2013). Genetic algorithm with binary representation of generating unit start-up and shut-down times for the unit commitment problem. *Expert Systems with Applications*, Volume 40: 6080–6086.

Ezendam, T. (2021). *Two geometry conformal methods for the use in a multi-disciplinary non-orthogonal building spatial design optimisation framework.* (MSc thesis). Eindhoven, The Netherlands: Eindhoven University of Technology.

Fasoulaki, E. (2007). Genetic Algorithms in Architecture: a Necessity or a Trend? *Proceedings of the 10TH Generative Art International Conference, Milan, Italy, December 2007.*

Feng, K., Lu, W., & Wang, Y. (2019). Assessing environmental performance in early building design stage: An integrated parametric design and machine learning method. *Sustainable Cities and Society*, Volume 50, October 2019, 101596.

Frey, P. J., & George, P. L. (2000). *Mesh Generation.* HERMES Science Publishing.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* Cambridge, MA: MIT Press.

Hamalainen, A., Teixeira, A., Almeida, N., Meinedo, H., Fegy´o, T., & Dias, M. S. (2015). Multilingual Speech Recognition for the Elderly: The AALFred Personal Life Assistant. *Procedia Computer Science*. Volume 67: 283–292.

Haymaker, J., Fischer, M., Kunz, J. & Suter, B. (2004). Engineering Test Cases to Motivate the Formalization of an AEC Project Model as a Directed Acyclic Graph of Views and Dependencies. *Electronic Journal of Information Technology in Construction*, Volume 9: 419-441.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, December 2015*.

Hofmeyer, H. & Davila Delgado, J.M. (2015). Coevolutionary and Genetic Algorithm Based Building Spatial and Structural Design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Volume 29: 351-370.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on Machine Learning, Lille, France, July 2015*. Vol. 37: 448-456.

Kalay, Y. (2004). *Architecture's New Media: Principles, Theories, and Methods of Computer-Aided Design*. MIT Press. ISBN 978026211284-0.

Khosravian, A., Amirkhani, A., Kashiani, H., & Masih-Tehrani, M. (2021). Generalizing state-of-the-art object detectors for autonomous vehicles in unseen environments. *Expert Systems with Applications*. Volume 183, 115417.

Kingma, D.P. & Ba, J. (2015). Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR), San Diego, CA, USA, May 2015.*

Kolodiazhnyi, K. (2020). *Hands-On Machine Learning with C++: Build, train, and deploy end-to-end machine learning and deep learning pipelines.* Birmingham: Packt Publishing.

Kumar, D., Sarangi, P. K., & Verma, R. (2021). A systematic review of stock market prediction using machine learning and statistical techniques. *Materials Today: Proceedings.* Volume 49: 3187-3191.

Maher, M., Tang, HH. (2003). Co-evolution as a computational and cognitive model of design. *Research in Engineering and Design*, Volume 14: 47–64.

MacLeamy, P. (2004). *Collaboration, integrated information and the project lifecycle in building design, construction and operation.* Construction Users Roundtable, WP-2012.

Mitchell, T. M. (1997). *Machine Learning*. New York: McGraw-Hill. ISBN 0070428077.

Rezaei, Z. (2021). A review on image-based approaches for breast cancer detection, segmentation, and classification. *Expert Systems with Applications.* Volume 182, 115204.

Rocha, M. & Neves, J. (1999). Preventing Premature Convergence to Local Optima in Genetic Algorithms via Random Offspring Generation. *Proc. of the 12th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Cairo, Egypt, May 1999.*

Sadek, K., Raslan, R., El-Bastawissi, I. & Sayary, S. (2019). Impact of BIM on Building Design Quality. *BAU Journal. Creative Sustainable Development. 1.* November 2019.

Simon, H. A. (1997). *Models of Bounded Rationality: Empirically Grounded Economic Reason*. Cambridge, MA: The MIT Press.

Stanovov, V., Brester, C., Kolehmainen, M., & Semenkina, O. (2017). Why don't you use Evolutionary Algorithms in Big Data? *IOP Conference Series: Materials Science and Engineering*, Volume 173: 12-20.

Syswerda, G. (1989). Uniform Crossover in Genetic Algorithms. *Proceedings of the 3rd International Conference on Genetic Algorithms, George Mason University, Virginia, June 1989.*

Wallet, B., Marchette, D. & Solka, J. (1996). A Matrix Representation for Genetic Algorithms. *Proceedings of Automatic Object Recognition VI of SPIE Aerosense, ort, May 1996.* Volume 2756: 206-214. SPIE.

Wong, K. (2015). Evolutionary Multimodal Optimization: A Short Survey. arXiv online curated research-sharing platform (Cornell University), ref: arXiv:1508.0045.

Wu, Z. Y. & Katta, P. (2009). Applying Genetic Algorithm to Geometry Design Optimization: Improving Design by Emulating Natural Evolution, *Technical Report*, Bentley Systems, Incorporated, Watertown, CT, USA.

Von Buelow, P., Falk, A. & Turrin, M. (2010). Optimization of structural form using a genetic algorithm to search associative parametric geometry. *Proceedings of the First International Conference on Structures and Architecture (ICSA), Guimarães, Portugal, July 2010.*

Zhang, J., Liu, N., & Wang, S. (2021). Generative design and performance optimization of residential buildings based on parametric algorithm. *Energy and Buildings*, Volume 244, August 2021, 111033, ISSN 0378-7788.

A LITERATURE REVIEW

A.1 Building design process and support systems

In the built environment, a building design process is understood as the creation of a plan to realise a building (Hofmeyer & Davila Delgado 2015). According to the reviewed test cases in the research of Haymaker et al. (2004), building design processes are unique and iterative, where multiple disciplines are involved. This makes these processes complex, time-consuming and error prone. Therefore, design processes should be supported in the early stages to improve the final design (Kalay 2004), and to reduce the cost of design changes (MacLeamy 2004). As a result, designers have to make important decisions in early stages, where all requirements of different disciplines need to be considered. However, the constraints and requirements affect each other (Maher & Tang, 2003), so design alternatives need to be made. Even though, a designer is not able to consider all possible alternatives to make well-founded decisions. Simon (1997) reasons that the bounded rationality of humans limits their decision-making capability. To overcome this limitation, engineers may apply design support systems.

Design processes can be supported by parametric and generative design methods to evaluate large numbers of design solutions (Feng et al. 2019, Zhang et al. 2021). Furthermore, engineers make use of simulation methods to predict the performance of a building, and optimisation methods to find optimal design solutions within certain criteria.

Virtual environments are used to improve the collaboration between all participants. BIM is used to maintain a digital representation of the building along the design, engineering and construction phases (Hofmeyer & Davila Delgado, 2015). Moreover, by using BIM in the building life-cycle, information can be stored, managed and transferred between all stakeholders, which enhances the building design quality, according to Sadek et al. (2019).

A.2 Conformal geometries in design support systems

Design support systems exist that (a) suggest building designs, which can be used by the design teams; (b) simulate design processes (via simulations with partial models), to support the teams process-wise. In all such systems, a representation is used for the Building Spatial Design (BSD). And for domain specific analyses, it is very useful to have a so-called conformal (CF) representation for the BSD. In a conformal representation, for all entities it holds: the vertices of an entity are, if intersecting another entity, only allowed to coincide with this other entity's vertices. Figure A1 shows BSD surfaces with T-joints and varying dimensions, which are transformed into a conformal geometry, here via partitioning of surfaces.

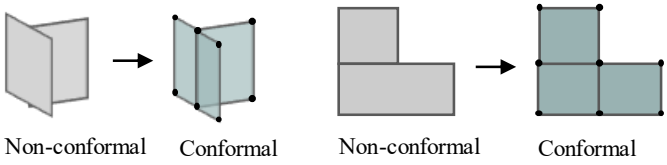


Figure A1. Non-conformal BSD surfaces are transformed into conformal geometries.

Initially, the transformation to a conformal geometry is used to generate a mesh for the finite element model. This is partially presented as the multi-block method by Frey and George (2000). In addition, a conformal geometry is convenient for discipline-specific models. For example, for structural design, proper finite element meshing and loading can be done more correctly with a conformal geometry, see Figure A2. Due to the partitioning of surfaces, the live loads can be assigned more correctly, with distinction between different types of structural elements. For example, in Figure A2, internal floors have different loads than external roofs. Furthermore, if the conformal geometry is used as a blueprint for structural elements, also more efficient load paths may arise (Claessens et al. 2020), see Figure A2 (red arrows).

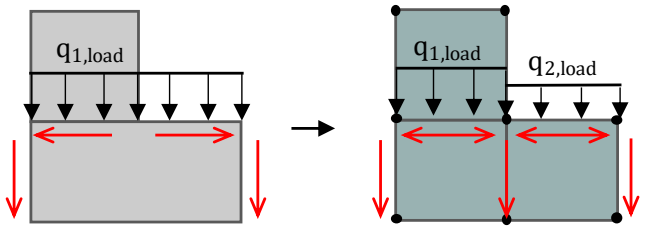


Figure A2. Allocation of live loads ($q_{i,load}$) to building components. Conformal representation (right) ensures for more systematic allocation. More efficient load paths (red arrows).

Thermal modelling of the building can be done more correctly too, as surfaces are either fully internal or external, and so fluxes for each surface can be correctly described, see Figure A3.

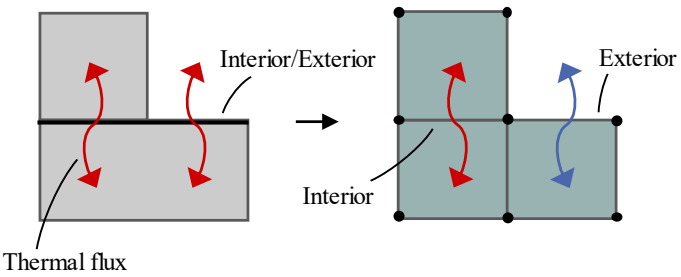


Figure A3. Allocation of thermal properties to building components. Conformal representation (right) ensures for more systematic allocation.

Current procedural methods to transform a BSD into a conformal design work for orthogonal, and sometimes for specific non-orthogonal designs (Ezendam 2021), whereas realistic building designs may be less specific non-orthogonal.

A.3 Machine learning and applications in building design process

The state-of-the-art research of Hong et al. (2020) explored many articles to present an overview of relevant applications of machine learning in different parts of the building life cycle. The review of the design phase is most interesting for this research. According to Hong et al.(2020), machine learning can be used in the generation and evaluation of design models for multiple disciplines. Design processes can be supported by parametric and generative design methods. However, the design models are manually preprogrammed sets of functions. To improve time, cost and effort in the generation of design models, time-consuming processes, like manually implementing design rules, could be automated (Hong et al. 2020). Therefore, machine learning can be used to emulate these hard-coded rules by replacing the functions with a trained ML model. Additionally, Tamke et al. (2018) shows that machine learning can be used to better understand the unknown relations between parameters in design processes. In the optimisation of topology, Wang et al. (2020) demonstrates that the used machine learning model cannot substitute the optimisation process. Although, the process can be improved with machine learning by reducing the computation time. The process of training a model is computationally demanding. However, if the model is trained, predictions can be made instantly.

A.4 Genetic Algorithms and applications in building design process

In architecture and the building design process, genetic algorithms can be used as optimisation and form-generation tools (Fasoulaki, 2007). GAs are beneficial in geometry design optimisation according to Wu et al. (2009). It enables design teams to explore a wide range of design alternatives. Moreover, GAs are proved to be flexible on multiple designs and user-defined parameters. The research of Buelow et al. (2010) shows that GAs are suitable to support the design phase. However, they should not substitute the existing methods. The designer should still play an active role in the design process and decision-making.

Building spatial designs can be optimised on structural and thermal performance by using hybridisation of an evolutionary algorithm and simulations of co-evolutionary design processes (Boonstra et al. 2021). In such optimisation processes, there is not one perfect solution, but the algorithm searches for a range of possible solutions. GAs can only provide good quality solutions, and it cannot guarantee optimality.

B MACHINE LEARNING

Extra information is provided about ML.

B.1 Learning type and task

A machine uses information to learn certain rules, which can be applied to solve tasks automatically. A machine can learn in three different ways (Goodfellow et al. 2017).

- Supervised learning: The dataset is fixed and the input and associated output are known. The machine learns to predict the output (target) from an input (feature). The algorithm is trained on the complex relation between features and targets.
- Unsupervised learning: The dataset is fixed, but the output is not known. The machine learns useful properties of the dataset, such as hidden patterns or clusters.
- Reinforcement learning. The dataset is not fixed, but interacts with the environment. The machine learns by trial and error in combination with rewards.

Within supervised learning, two main tasks exist, namely regression and classification. Since this research wants to find complex relations between real numerical values, the machine needs to solve a regression task. Regression problems can be solved by, for example, linear, multiple or logistic regression algorithms. However, these algorithms are limited in complexity and in managing large sets of data with many features (Goodfellow et al. 2017). Nevertheless, neural networks do not have these limitations and are capable of solving both regression and classification problems. Therefore, neural networks seem to be the most appropriate machine learning algorithms for this research.

B.2 Neural networks (FNN, RNN, CNN)

The feed forward neural network (FNN) is the fundamental network, where each layer is connected to the next layer, without feedback. In recurrent neural networks, neurons in one layer can be connected to other neurons in the same layer or to a previous layer. As a result, a feedback loop is integrated in the network. Recurrent neural networks are mainly used for speech recognition (Hamalainen et al., 2015) and other applications that include time-series, such as stock price predictions (Kumar et al. 2021) . Regular feed forward networks are powerful on structured datasets with relatively small number of features. However, datasets can become large and unstructured. Therefore, a convolutional operation could be applied. In convolutional neural networks, the hidden layers are regularised and a convolution is applied on each neuron. Convolutional neural networks are mainly used for image classification (Rezaei 2021) or object

recognition (Khosravian et al. 2021). The large number of pixels in an image are compressed to less pixels. Finally, the FNN is used in the analysis. There is no need to use a feedback loop or to apply a convolutional operation. Compressing the data structures (corner-vertices) would not be convenient for building designs.

B.3 Neuron as processing unit

Neurons are processing units that form the neural network, see Figure B1.

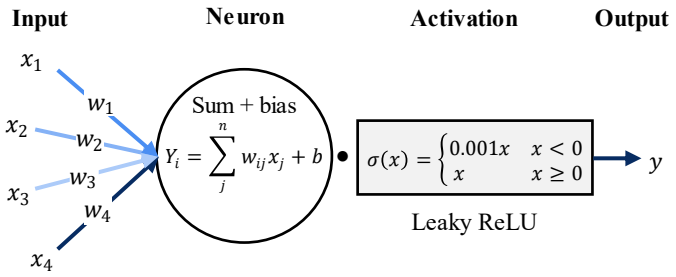


Figure B1. Neuron as processing unit.

A neuron consists of four parts:

1. **Input**
   The output values of neurons in the previous layer are used as input for the next layer (x_j). In a fully connected network, all neurons of one layer are connected to all neurons of the next layer. Furthermore, each connection has a certain weight (w_{ij}). In Figure B1 the opacity of the connection is proportional to the weight. Before the model is trained, the weights should have initial values. The weight initialisation is done randomly with a variance. According to He et. al (2015), the He initialisation function is most suitable when using a ReLU based activation function. The weights are finally adjusted in the training process.
2. **Neuron**
   Each input value (x_j) is multiplied with the weight of the connection (w_{ij}). These multiplications are summed up and set the value of the neuron. A bias can be added to shift the result of the neuron.
3. **Activation function**
   After each neuron, an activation function is applied to include non-linearity. The Leaky ReLU (Rectified Linear Unit) activation function is applied after each hidden layer. The ReLU function is commonly used, because it avoids the vanishing gradient problem (Kolodiazhnyi, 2020). Moreover, a generalisation of the rectified linear unit (Leaky ReLU) will be used to avoid the dying ReLU problem (Kolodiazhnyi, 2020, Goodfellow et al. 2017). Using Leaky ReLU, all values lower than zero become almost zero and all values above zero are kept the same, see function and graph in Figure B1.
4. **Output**
   The result of previously mentioned operations is the output of this neuron, and can be used as input for next layer.

B.3 Training and evaluation

If the architecture of the machine learning model is designed, the training phase can start. The model is trained on the training dataset (80 %). In the training process, the weights are adjusted so that the network approaches the complex relation between input and output layer. Back propagation is used to train the network. The goal is to minimise the loss between predicted output and the actual output. The loss is indicated by the mean squared error (MSE). A lower MSE means a more accurate prediction. The MSE is calculated as follows:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(\mathrm{Y}_i - \hat{\mathrm{Y}}_i\right)^2 \qquad \text{(B1)}$$

where, n = the number of data points; Y = actual output; and $\hat{\mathrm{Y}}$ = predicted output.

The training process of a neural network starts with forward propagation, where each neuron output value is calculated according to Figure B1 with the initial weights, biases and input values. This process starts at the input, goes through the hidden layer, and ends at the output. The calculated output is compared to the actual output using the loss function. If the predictions are not accurate, the loss needs to be minimised and is propagated back into the network. During backward propagation, an optimization function is used to reduce the loss by adjusting the weights in the right direction. The new weights are calculated by equation B2 (Kolodiazhnyi 2020).

$$\omega_{new} = \omega - \eta \cdot \frac{\partial L}{\partial \omega} \qquad \text{(B2)}$$

where ω = weight; η = learning rate; and L = loss (MSE).

The learning rate determines the degree of adjustment of the weights and varies between zero and one. The derivative of the loss indicates the slope of the gradient descent. Therefore, the direction of updating the weights to a global minimum can be defined.

The Adam optimisation algorithm is used in the network and can be specified as an advanced gradient descent function. The Adam optimiser uses a variable learning rate that is adapted depending on the learning process (Kolodiazhnyi 2020). The initial learning rate and batch size have to be defined for the optimiser.

A relatively large learning rate leads to divergent behaviour of the error. On the contrary, a small learning rate leads to convergent behaviour, but uses many

steps and results in a longer simulation time. The optimal learning rate is dependent on the dataset, and therefore, needs to be tuned by analysing the MSE for different rates.

Based on the batch size, the training process can be done in three modes, namely stochastic, batch or mini-batch (Kolodiazhnyi 2020). In stochastic mode, the batch size is equal to one. In batch mode, the batch size is equal to the number of training samples. In mini-batch mode, the batch varies between one and number of training samples. The combination of batch size and training iterations determines the number of epochs. If all training samples are passed through the algorithm, one epoch is performed. For example, a training dataset has 80 data points. If the batch size is 40, there are two iterations needed for one epoch. It is convenient to use a batch size that is a factor of the number of training data points. A larger batch size leads to a faster training process, but uses more memory space (Goodfellow et al., 2017). Therefore, a mini-batch between these extremes is most favourable. The optimal batch size varies for different datasets, and therefore, the optimal batch size needs to be tuned by analysing the MSE for different batch sizes.

The testing data (20%) is used to evaluate the trained model. By analysing the errors of the training set and testing set, two problems could be detected, namely over-fitting and under-fitting (Goodfellow et al., 2017). Over-fitting occurs when the trained model fits perfectly on training data, but less on testing data. This can be solved by using more data points or reducing the number of features of one data point. Under-fitting occurs when the model does not fit on training and testing data. This can be solved by using a more advanced machine learning model or using more features.

B.4 Hyper-parameters

There are four hyper-parameters that need to be defined: number of hidden layers, number of neurons, learning rate and batch size. These parameters are dependent on the dataset and can be changed during the analysis. Based on the One-hot encoded 'Single configuration' dataset with 100 data-points, a model is trained for different number of hidden layers and neurons (Figures B2 & B3), and for learning rate and batch size (Figures B4 & B5).

From Figure B2, one can see that more hidden layers (HL) do not improve the training process. Nevertheless, the calculation time increases by using more layers, so one hidden layer is sufficient for this problem.

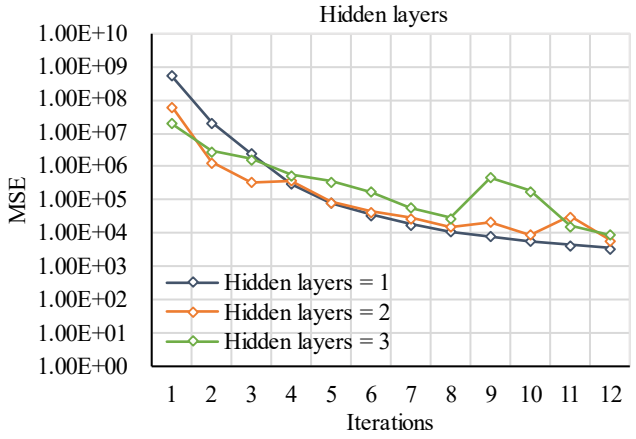


Figure B2. Analysis of hidden layers

For this dataset, the input layer has 39 nodes and the output layer has 168 nodes. The model is tested with one hidden layer and different number of neurons (10, 60, 120, 180, and 240). From Figure B3, it can be seen that the performance of the algorithm improves for more neurons in the hidden layer. However, more neurons lead to more weights, which leads to a more extensive computation. Moreover, the enhancement of using more neurons becomes less significant when using more neurons than the input or output layers have. As a result, the number of neurons in a hidden layer needs to be equal or slightly larger than the number of nodes in the adjacent input or output layers. A suitable number of neurons would be 180 in this situation.

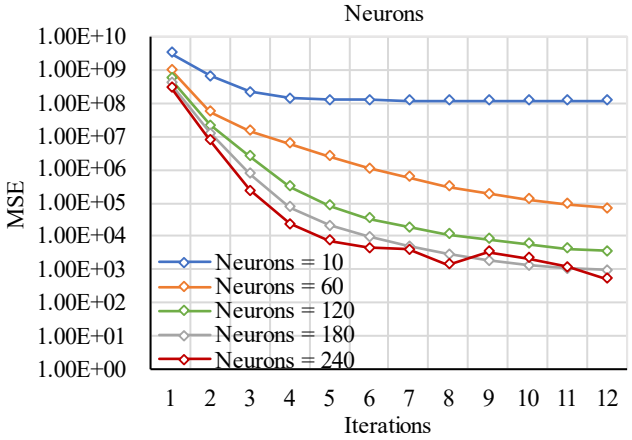


Figure B3. Analysis of neurons.

From Figure B4, it can be seen that a learning rate of 0.01 leads to the lowest and most stable results. The larger learning rate of 0.1 leads to divergent behaviour, where the error is not reducing steadily. The lower learning rate of 0.001 leads to a slower convergence.

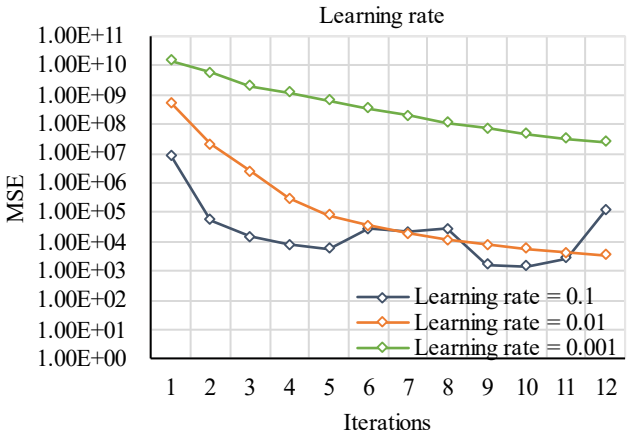


Figure B4. Analysis of learning rate.

The resulting loss does not vary for different batch sizes, according to Figure B5. A larger batch size leads to faster training process but uses more memory space. Therefore, a mini-batch between 1 and the number of training samples is most favourable.

Figure B5. Analysis of batch size.

B.5 Test buildings (ML analysis)

The test buildings in ML analysis are described by the origin vector (***s***) and dimensions vector (***d***). Here, the values are described in meters.

- Test building A
  Space 1:
  s = { 10, 10, 0 } **d** = { 6, 6, 3 }
  Space 2:
  s = { 16, 12, 0 } **d** = { 6, 6, 5 }
- Test building B
  Space 1:
  s = { 10, 10, 0 } **d** = { 6, 6, 3 }
  Space 2:
  s = { 4, 8, 0 } **d** = { 6, 6, 5 }
- Test building C
  Space 1:
  s = { 10, 10, 0 } **d** = { 8, 8, 4 }
  Space 2:
  s = { 12, 12, 4 } **d** = { 4, 4, 4 }

C GENETIC ALGORITHM

C.1 Test buildings (GA analysis)

The test buildings in GA analysis are described by the corner-vertices (***p***) of each space. Here, the values are described in meters.

Orthogonal (O) designs:

- Test building A-O
  Space 1:
  {{10,10,0}, {16,10,0}, {16,16,0}, {10,16,0},
  {10,10,3}, {16,10,3}, {16,16,3}, {10,16,3}}
  Space 2:
  {{16,12,0}, {22,12,0}, {22,18,0}, {16,18,0},
  {16,12,5}, {22,12,5}, {22,18,5}, {16,18,5}}
- Test building B-O
  Space 1:
  {{10,10,0}, {16,10,0}, {16,16,0}, {10,16,0},
  {10,10,3}, {16,10,3}, {16,16,3}, {10,16,3}}
  Space 2:
  {{4,8,0}, {10,8,0}, {10,14,0}, {4,14,0},
  {4,8,5}, {10,8,5}, {10,14,5}, {4,14,5}}
- Test building C-O
  Space 1:
  {{10,10,0}, {18,10,0}, {18,18,0}, {10,18,0},
  {10,10,4}, {18,10,4}, {18,18,4}, {10,18,4}}
  Space 2:
  {{12,12,4}, {16,12,4}, {16,16,4}, {12,16,4},
  {12,12,8}, {16,12,8}, {16,16,8}, {12,16,8}}

Non-orthogonal (NO) designs:

- Test building A-NO
  Space 1:
  {{10,12,0}, {16,10,0}, {16,16,0}, {10,18,0},
  {10,12,3}, {16,10,3}, {16,16,3}, {10,18,3}}
  Space 2:
  {{16,12,0}, {22,10,0}, {22,16,0}, {16,18,0},
  {16,12,5}, {22,10,5}, {22,16,5}, {16,18,5}}
- Test building B-NO
  Space 1:
  {{10,12,0}, {16,10,0}, {16,18,0}, {10,16,0},
  {10,12,3}, {16,10,3}, {16,18,3}, {10,16,3}}
  Space 2:
  {{16,16,0}, {22,12,0}, {22,20,0}, {16,20,0},
  {16,16,5}, {22,12,5}, {22,20,5}, {16,20,5}}
- Test building C-NO
  Space 1:
  {{10,12,0}, {16,10,0}, {16,18,0}, {10,16,0},
  {10,12,3}, {16,10,3}, {16,18,3}, {10,16,3}}
  Space 2:
  {{16,16,0}, {22,12,0}, {22,20,0}, {16,20,0},
  {16,16,5}, {22,12,5}, {22,20,5}, {16,20,5}}

D SOFTWARE

The software that is used for this graduation thesis is documented in this Appendix. Parts of the BSO-toolbox developed at the University, as well as new code files written by the author are used. First, an overview of all files and external software is given. Thereafter, Section D.1 elaborates the Machine Learning files and Section D.2 elaborates the Genetic Algorithm files. Finally, the modified C++ files from the BSO-toolbox are listed in Section D.3.

Seven C++ files from the BSO-toolbox v1.0.0 are slightly modified. The files are listed with modifications at the end of the appendix in Section D.3. It concerns the following files with directories:

- BSO-toolbox/bso/spatial_design/ms_building.cpp
- BSO-toolbox/bso/spatial_design/ms_building.hpp
- BSO-toolbox/bso/spatial_design/ms_space.cpp
- BSO-toolbox/bso/spatial_design/ms_space.hpp
- BSO-toolbox/bso/building_physics/properties/construction.cpp
- BSO-toolbox/bso/structural_design/component/structure.cpp
- BSO-toolbox/bso/utilities/geometry/quad_hexahedron.hpp

The code is related to the two research methods: Machine Learning (ML) and Genetic Algorithm (GA). These files are explained and listed in respectively Section D.1 and Section D.2. The code is written in C++14 and uses external libraries in some of the files. An overview of all created C++ files is listed in Table D1.

Table D1. Overview of created C++ files. File name, for which research method, on what operating system, and which external libraries are included.

File name	Method	Operating system	Includes BSO-toolbox	Includes Eigen	Includes OpenGL	Includes mlpack
generateBSDs_ML.cpp	ML	Linux				
generateDataset_ML.cpp	ML	Linux	x	x		
NeuralNetwork_TRAIN.cpp	ML	Windows				x
NeuralNetwork_PREDICT.cpp	ML	Windows				x
Visualisation_ML.cpp	ML	Linux			x	
GA1and2.cpp	GA	Linux	x			
Visualisation_GA.cpp	GA	Linux			x	

The files are separated in two folders, based on their operating system: Linux or Windows.

Operating system
Most of the files are compiled for a Linux operating system using GCC C++ compiler version 9.4.0. The two Neural Network files, which include the mlpack C++ library are compiled for a Windows operating system. These two files are compiled and ran via Visual Studio Community 2019 version 16.8.4 using MSVC compiler version 14.28.

Makefile (Linux)
A makefile has been added to the Linux folder, which can be used to compile the code for Linux operating system. The files can be compiled and ran by typing the following commands in the terminal.

Table D2. Makefile commands.

File name	Compile	Run
generateBSDs_ML.cpp	make clean design	./bsd
generateDataset_ML.cpp	make clean data	./dataset
Visualisation_ML.cpp	make clean viml	./visualML
GA1and2.cpp	make clean ga	./algo
Visualisation_GA.cpp	make clean viga	./visualGA

External software
The code is mainly based on two external software packages: BSO-toolbox and mlpack C++ library. Other external libraries are included within these two software installations. An overview of all external software is listed below. The reference to the documentation and installation guide, as well as, the used versions of all libraries are indicated.

- BSO-toolbox developed at Eindhoven University of Technology
  - See https://github.com/TUe-excellent-buildings/BSO-toolbox for documentation and installation guide.
  - Version = v1.0.0
- Eigen C++ library for linear algebra. (Installed with the BSO-toolbox installation)
  - Version = v3.4.0
- Boost C++ library for various utilities. (Installed with the BSO-toolbox installation)
  - Version = v1.071.00
- OpenGL C++ library for visualisation. (Installed with the BSO-toolbox installation)
  - GSL (last tested for v2.4+dfsg-6 amd64)
  - freeglut3 (last tested for v2.8.1-3 amd64)
- Mlpack C++ library for Machine learning.
  - See https://www.mlpack.org/ for documentation and installation guide.
  - mlpack is built and used on Windows 10 and Visual Studio 2019.
  - Version = v3.4.2
  - mlpack is built on:
    - Armadillo C++ linear algebra library (v10.8.2)
    - Ensmallen C++ numerical optimisation library (v2.14.2)
    - Boost C++ library (v1.75.0)

Input text-files
Furthermore, the software contains two initial text-files: "input_BSD_ML.txt" and "cornervertices BSD_GA.txt". These files are used as input for the two methods to predict the conformal representation of Test building 'A' and Test building 'A-O', as described in the analyses (Sections 2.3 & 3.4).

Table D3. Test buildings described by input text-files and used for example.

Test building	Input file	Method
‘A’	"input_BSD_ML.txt"	ML
‘A-O’	"cornerverticesBSD_GA.txt"	GA

Test Building 'A' is described in "test_inputBSD_ML.txt":
6000,6000,3000,10000,10000,0,6000,6000,5000,16000,12000,0

Test Building 'A-O' is described in "cornerverticesBSD_GA.txt":
N, 1, 10,10,0, 16,10,0, 16,16,0, 10,16,0, 10,10,3, 16,10,3, 16,16,3, 10,16,3
N, 2, 16,12,0, 22,12,0, 22,18,0, 16,18,0, 16,12,5, 22,12,5, 22,18,5, 16,18,5

These two buildings are used for an example that can be executed. This is elaborated in Sections D.1 and D2.

Makefile (Linux)

```
# specify location of libraries
BOOST = /usr/include/boost
EIGEN = /usr/include/eigen
BSO = ..
ALL_LIB = -I$(BOOST) -I$(EIGEN) -I$(BSO)

# compiler settings
CPP = g++ -std=c++14
FLAGS = -O3 -march=native -lglut -lGL -lGLU -lpthread

# specify file(s) to be compiled
BSDFILE  = generateBSDs_ML.cpp
DATAFILE = generateDataset_ML.cpp
VIMLFILE = visualisation_ML.cpp

GENETICF = GA1and2.cpp
VIGAFILE = visualisation_GA.cpp

# specify name of executables
EXE = bsd
EXE = dataset
EXE = visualML
EXE = algo
EXE = visualGA

.PHONY: all clean

# definition of arguments for make command
# MACHINE LEARNING
design:
  $(CPP) -o bsd $(ALL_LIB) $(BSDFILE) $(FLAGS)
data:
  $(CPP) -o dataset $(ALL_LIB) $(DATAFILE) $(FLAGS)
viml:
  $(CPP) -o visualML $(ALL_LIB) $(VIMLFILE) $(FLAGS)

# GENETIC ALGORITHM
ga:
  $(CPP) -o algo $(ALL_LIB) $(GENETICF) $(FLAGS)
viga:
  $(CPP) -o visualGA $(ALL_LIB) $(VIGAFILE) $(FLAGS)



# remove previously compiled executable
clean:
  @rm -f $(EXE)

```

D.1 Machine Learning code files

The machine learning code consists of five C++ files, which are compiled separately. The filenames are listed below, and they are generally used in this order. The raw code files are listed at the end of this Section D.1.

- "generateBSDs_ML.cpp"
- "generateDataset_ML.cpp"
- "NeuralNetwork_TRAIN.cpp"
- "NeuralNetwork_PREDICT.cpp"
- "visualisation_ML.cpp"

See Figure D1 for an overview of all C++ files, input and output text-files, used libraries, and operating system.

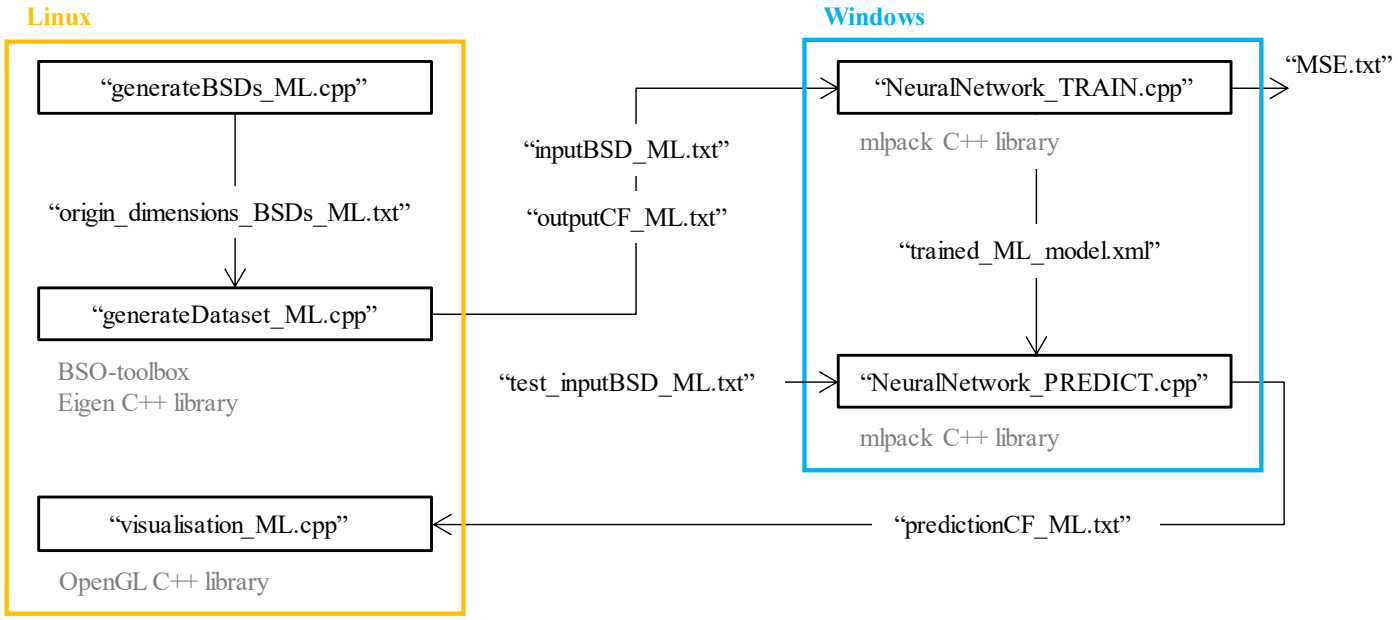


Figure D1. Overview of all files related to ML method with included libraries.

Besides the explanations, an example is added, where Test Building 'A' from Section 2.3 will be made conformal by a trained Machine Learning model. Therefore, first a dataset needs to be created with the files: "generateBSDs_ML.cpp" and "generateDataset_ML.cpp". Secondly, a ML model is trained using the created dataset and the file: "NeuralNetwork_TRAIN.cpp" and saved afterwards to make predictions with "NeuralNetwork_PREDICT.cpp". Finally, a prediction of the conformal model can be visualised using "visualisation_ML.cpp".

D.1.1 "generateBSDs_ML.cpp"

The ML method starts with "generateBSDs_ML.cpp" to define a wide range of building configurations. The number of BSDs, included variant types and the range of dimensions can all be changed by the user. The output is a text-file: "origin_dimensions_BSDs_ML.txt", which includes the input for next C++ file ("generateDataset_ML.cpp").

The "origin_dimensions_BSDs_ML.txt" file consists of multiple buildings. However, as example, one Building Spatial Design (BSD) can be described as follows:

R, 1, 6000, 5000, 5000, 10000, 10000, 0, A
R, 2, 6000, 7000, 3000, 9000, 5000, 5000, A

Which indicates: R, space ID, dimensions vector (width, depth, height), origin vector (x, y, z), A

Example

For the example, the file "generateBSDs_ML.cpp" generates 100 BSDs with 2 spaces for each variant: "top", "front", "behind", "left", "right". The width and depth range between 4000 - 8000 mm, and the height ranges between 2000 - 6000 mm.

The file can be compiled by typing the following command in the terminal:

```
make clean design
```

The file can be executed by typing the following command in the terminal:

```
./bsd
```

D.1.2 "generateDataset.cpp"
A dataset with input features (BSDs) and output targets (conformal models) can be created by "generate-Dataset_ML.cpp". For each BSD in "origin_dimensions_BSDs_ML.txt", the transformation to its conformal geometry is executed by the BSO-toolbox. As a result, the input features and output targets are saved in the following text-files.

The input can be described in 4 different ways. Input text-files (features):

- "inputBSD_ML.txt"
  - describes the BSD by origin and dimensions vector.
- "inputBSD_NumberEncoded_ML.txt"
  - describes the BSD by origin and dimensions vector plus number encoded building types.
- "inputBSD_OneHotEncoded_ML.txt"
  - describes the BSD by origin and dimensions vector plus One-hot encoded building types.
- "cornerverticesBSD_ML.txt"
  - describes the BSD by corner-vertices of each space.

Output text-files (targets):

- "outputCF_ML.txt"
  - describes the CF model by corner-vertices of each cuboid.

Example
For the example, the "inputBSD_ML.txt" file is used for the features, and the "outputCF_ML.txt" file is used for the targets. When the features need to be, for example One-hot encoded, the "inputBSD_OneHotEncoded_ML.txt" file is used as input.

The file can be compiled by typing the following command in the terminal:

```
make clean data
```

The file can be executed by typing the following command in the terminal:

```
./dataset
```

D.1.3 "NeuralNetwork_TRAIN.cpp"
The resulting text-files from previous C++ file are copied and pasted to the Visual Studio 19 project on Windows 10 OS, where all the dependencies of mlpack library are installed. The features ("inputBSD_ML.txt") and targets ("outputCF_ML.txt") are used as dataset to train the Neural network. Finally, a trained ML model is saved, which then can be used to make predictions of conformal models (see "NeuralNetwork_PREDICT.cpp"). Additionally, the loss (MSE) is calculated during the training process and saved in "MSE.txt".

Example
For the example, a neural network with one hidden layer and 180 neurons is trained on "inputBSD_ML.txt" as features and "outputCF_ML.txt" as targets. The C++ files and text-files should be located in the same folder in the Visual Studio project. The trained ML model is also saved in the same folder as XML file: "trained_ML_model.xml".

D.1.4 "NeuralNetwork_PREDICT.cpp"
With the trained ML model ("trained_ML_model.xml"), a test building can be used as input to predict its conformal (CF) representation ("predictionCF_ML.txt").

Example
For the example, test building 'A' is used to predict its conformal representation. Test building 'A' is described by "test_inputBSD_ML.txt", and should be placed in the same folder.

D.1.5 "visualisation_ML.cpp"
The predicted CF models can be visualised by "visualisation_ML.cpp" to compare the actual conformal model and its prediction.

Example
The predicted CF model of test building 'A' is visualised below. The input for the visualisation is the "predictionCF_ML.txt"-file.

The file can be compiled by typing the following command in the terminal:

```
make clean viml
```

The file can be executed by typing the following command in the terminal:

```
./visualML
```

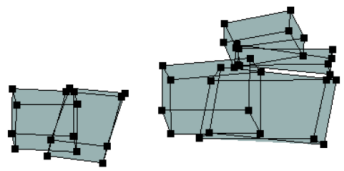


Figure D2. Visualisation of predicted conformal model Test building 'A'

"generateBSDs_ML.cpp"

```
1.  // C++ References
2.  #include <iostream>
3.  #include <algorithm>
4.  #include <functional>
5.  #include <random>
6.  #include <vector>
7.  #include <fstream>
8.  #include <string>
9.
10. // Namespaces
11. using namespace std;
12.
13. // Data generation of building spatial designs (BSDs) with 2 spaces
14. // Wide range of possible configurations, randomly defined
15. // 5 Different variants: 'top' || 'front' || 'behind' || 'left' || 'right'
16.
17. // Input Parameters
18. int nBSDs = 20;            // nr. of Building Spatial Designs (BSDs) per variant.
19. int nSpaces = 2;            // nr. of spaces in BSD. Currently, fixed to 2
20. int step = 1000;            // The step in the dimensions range.
21.                            // --> For example, if the width ranges from 4000-8000, the values 4000,
    5000, 6000, 7000, and 8000 are considered
22.
23. // The variants which are included in the dataset, can be one or multiple
24. vector<string> typesIncluded = {"top", "front", "behind", "left", "right"};
25.
26.
27. // Space dimensions (w, d, h) are randomly defined
28. class spaceDimensions {
29.     private:
30.         int minW, maxW, minD, maxD, minH, maxH;
31.     public:
```

```
32.         std::vector<int> width, depth, height;
33.
34.         // CONSTRUCTOR: space dimensions range
35.         spaceDimensions(int miw, int maw, int mid, int mad, int mih, int mah) {
36.             minW = miw;     maxW = maw;                                         //mm        Width
37.             minD = mid;     maxD = mad;                                         //mm        Depth
38.             minH = mih;     maxH = mah;                                         //mm        Height
39.         }
40.
41.         // METHOD: random number generator for dimensions
42.         void randomNumberGen() {
43.             for (int jw = 0; jw <= (nBSDs*typesIncluded.size())/((maxW-minW)/step+1); jw++)
44.             {
45.                 for (int iw=minW; iw<=maxW; iw+=step)
46.                 {
47.                     width.push_back(iw);
48.                 }
49.                 int iw = minW;
50.             }
51.             std::random_shuffle(width.begin(), width.end());
52.             for (int jd = 0; jd <= (nBSDs*typesIncluded.size())/((maxD-minD)/step+1); jd++)
53.             {
54.                 for (int id=minD; id<=maxD; id+=step)
55.                 {
56.                     depth.push_back(id);
57.                 }
58.                 int id = minD;
59.             }
60.             std::random_shuffle(depth.begin(), depth.end());
61.             for (int jh = 0; jh <= (nBSDs*typesIncluded.size())/((maxH-minH)/step+1); jh++)
62.             {
63.                 for (int ih=minH; ih<=maxH; ih+=step)
64.                 {
65.                     height.push_back(ih);
66.                 }
67.                 int ih = minH;
68.             }
69.             std::random_shuffle(height.begin(), height.end());
70.         }
71. };
72.
73.
74. // --- MAIN --- //
75. int main(int argc, char *argv[])
76. {
77.     std::cout << "Start BSD Generation" << std::endl;
78.
79.     // Space 1  (min Width, max Width, min Depth, max Depth, min Height, max Height)
80.     // Define range of dimensions
81.     spaceDimensions space1(4000, 8000, 4000, 8000, 2000, 6000);
82.     space1.randomNumberGen();
83.
84.     // Space 2  (min Width, max Width, min Depth, max Depth, min Height, max Height)
85.     // Define range of dimensions
86.     spaceDimensions space2(4000, 8000, 4000, 8000, 2000, 6000);
87.     space2.randomNumberGen();
88.
89.     // Initial origin space 1 = {10000,10000,0}
90.     vector<int> coor10, coor0;
91.     for (int ic=0; ic<nBSDs*typesIncluded.size(); ic++) coor10.push_back(10000);
92.     for (int ic=0; ic<nBSDs*typesIncluded.size(); ic++) coor0.push_back(0);
93.     std::vector<int> coorX1 = coor10, coorY1 = coor10, coorZ1 = coor0;
94.
95.
96.     // Generate wide range of possible configurations, for each included variant
97.     vector<int> coorX2;
98.     vector<int> coorY2;
99.     vector<int> coorZ2;
100.      int nrOfTypes = 0;
101.
102.     for (int i = 0; i < typesIncluded.size(); i++ ) {
103.         if ( typesIncluded[i] == "top" )
104.         {
105.             nrOfTypes++;
106.             for (int i=(nrOfTypes-1)*nBSDs; i<nrOfTypes*nBSDs; i++)
107.             {
```

```
108.                 coorX2.push_back( (rand() % (((coorX1[i] + space1.width[i]) - (coorX1[i] -
    space2.width[i] + step)) / step )) * step + (coorX1[i] - space2.width[i] + step));
109.                 coorY2.push_back( (rand() % (((coorY1[i] + space1.depth[i]) - (coorY1[i] -
    space2.depth[i] + step)) / step )) * step + (coorY1[i] - space2.depth[i] + step));
110.                 coorZ2.push_back(space1.height[i]);
111.             }
112.         }
113.         if ( typesIncluded[i] == "front" )
114.         {
115.             nrOfTypes++;
116.             for (int i=(nrOfTypes-1)*nBSDs; i<nrOfTypes*nBSDs; i++)
117.             {
118.                 coorX2.push_back( (rand() % (((coorX1[i] + space1.width[i]) - (coorX1[i] -
    space2.width[i] + step)) / step )) * step + (coorX1[i] - space2.width[i] + step));
119.                 coorY2.push_back(coorY1[i] - space2.depth[i]);
120.                 coorZ2.push_back(coorZ1[i]);
121.             }
122.         }
123.         if ( typesIncluded[i] == "behind" )
124.         {
125.             nrOfTypes++;
126.             for (int i=(nrOfTypes-1)*nBSDs; i<nrOfTypes*nBSDs; i++)
127.             {
128.                 coorX2.push_back( (rand() % (((coorX1[i] + space1.width[i]) - (coorX1[i] -
    space2.width[i] + step)) / step )) * step + (coorX1[i] - space2.width[i] + step));
129.                 coorY2.push_back(coorY1[i] + space1.depth[i]);
130.                 coorZ2.push_back(coorZ1[i]);
131.             }
132.         }
133.         if ( typesIncluded[i] == "left" )
134.         {
135.             nrOfTypes++;
136.             for (int i=(nrOfTypes-1)*nBSDs; i<nrOfTypes*nBSDs; i++)
137.             {
138.                 coorX2.push_back(coorX1[i] - space2.width[i]);
139.                 coorY2.push_back( (rand() % (((coorY1[i] + space1.depth[i]) - (coorY1[i] -
    space2.depth[i] + step)) / step )) * step + (coorY1[i] - space2.depth[i] + step));
140.                 coorZ2.push_back(coorZ1[i]);
141.             }
142.         }
143.         if ( typesIncluded[i] == "right" )
144.         {
145.             nrOfTypes++;
146.             for (int i=(nrOfTypes-1)*nBSDs; i<nrOfTypes*nBSDs; i++)
147.             {
148.                 coorX2.push_back(coorX1[i] + space1.width[i]);
149.                 coorY2.push_back( (rand() % (((coorY1[i] + space1.depth[i]) - (coorY1[i] -
    space2.depth[i] + step)) / step )) * step + (coorY1[i] - space2.depth[i] + step));
150.                 coorZ2.push_back(coorZ1[i]);
151.             }
152.         }
153.     }
154.
155.     // Write origins and dimensions of BSDs in text file: "origin_dimensions_BSDs.txt"
156.     // This text file is according to the Movable/Sizable Model in the BSO-toolbox ("ms_in-
    put_file.txt")
157.     // The file is used as input for the simulation ("simulation.cpp")
158.     std::ofstream dataG("origin_dimensions_BSDs_ML.txt", std::ofstream::trunc);
159.     for (int i=0; i < nBSDs*nrOfTypes; i++)
160.     {
161.         // Space 1
162.         dataG << 'R' << ',' << '1' << ',';
163.         dataG << space1.width[i] << ',';
164.         dataG << space1.depth[i] << ',';
165.         dataG << space1.height[i] << ',';
166.         dataG << coorX1[i] << ',' << coorY1[i] << ',' << coorZ1[i] << ',' << 'A' << std::endl;
167.         // Space 2
168.         dataG << 'R' << ',' << '2' << ',';
169.         dataG << space2.width[i] << ',';
170.         dataG << space2.depth[i] << ',';
171.         dataG << space2.height[i] << ',';
172.         dataG << coorX2[i] << ',' << coorY2[i] << ',' << coorZ2[i] << ',' << 'A' << std::endl;
173.
174.         if (i < nBSDs*nrOfTypes - 1) { dataG << std::endl; }
175.         else if (i == nBSDs*nrOfTypes - 1) {}
176.     }
177.
```

```
178.       return 0;
179.   }
180.
```

“generateDataset.cpp”

```
1.  //--- BSO-toolbox header files ---//
2.  #include <bso/spatial_design/ms_building.hpp>
3.  #include <bso/spatial_design/cf_building.hpp>
4.
5.  // C++ References
6.  #include <iostream>
7.  #include <fstream>
8.  #include <sstream>
9.  #include <string>
10. #include <cctype>
11. #include <cstring>
12. #include <cmath>
13. #include <vector>
14.
15. // Eigen C++ library
16. #include <Eigen/Dense>
17.
18. // Namespaces
19. using namespace std;
20. using namespace Eigen;
21.
22. // The simulation function uses the origin and dimensions vectors of the BSDs ("origin_dimen-
    sions_BSDs.txt") as input
23. // Via the BSO-toolbox, the BSDs are transformed into Conformal designs
24. // INPUT:
25.     // The input is saved in a new text file: "inputBSD.txt", and can be used as the input features in
    the Machine Learning process
26.     // The inputBSD file can be enhanced by adding Number or One-hot Encoding: "inputBSD_NumberEn-
    coded.txt", "inputBSD_OneHotEncoded.txt"
27.     // In "cornerverticesBSD.txt", the BSDs are described by the corner-vertices of the quad-hexahedron
    spaces.
28. // OUTPUT:
29.     // The output is the Conformal (CF) model of each BSD, described by the corner-vertices of conformal
    cuboids
30.     // Saved in "outputCF.txt"
31.
32. void simulation(string inputsFile, string transformation, string geometry, vector<string> typesIncluded )
33. {
34.     // Read BSD origin and dimensions file
35.     string InputsLine;
36.     ifstream Inputs(inputsFile);                // = "origin_dimensions_BSDs.txt"
37.     ofstream msinput("ms_input_file.txt");
38.     ofstream inputBSDf("inputBSD.txt", ios::trunc);
39.     ofstream inputBSDf_OneHotEncoded("inputBSD_OneHotEncoded.txt", ios::trunc);
40.     ofstream inputBSDf_NumberEncoded("inputBSD_NumberEncoded.txt", ios::trunc);
41.     ofstream cornerverticesBSDf("cornerverticesBSD_ML.txt", ios::trunc);
42.     ofstream outputCFf("outputCF.txt", ios::trunc);
43.     int r = 0;
44.     int nlines = 0;
45.
46.     int topTypes, frontTypes, behindTypes, leftTypes, rightTypes;
47.     if (find(typesIncluded.begin(), typesIncluded.end(), "top"   ) != typesIncluded.end()) { topTypes
    = 81; } else { topTypes    = 0; }
48.     if (find(typesIncluded.begin(), typesIncluded.end(), "front" ) != typesIncluded.end()) { frontTypes
    = 27; } else { frontTypes  = 0; }
49.     if (find(typesIncluded.begin(), typesIncluded.end(), "behind") != typesIncluded.end()) { behindTypes
    = 27; } else { behindTypes = 0; }
50.     if (find(typesIncluded.begin(), typesIncluded.end(), "left"  ) != typesIncluded.end()) { leftTypes
    = 27; } else { leftTypes   = 0; }
51.     if (find(typesIncluded.begin(), typesIncluded.end(), "right" ) != typesIncluded.end()) { rightTypes
    = 27; } else { rightTypes  = 0; }
52.     int totalTypes = topTypes + frontTypes + behindTypes + leftTypes + rightTypes;
53.
54.     Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic> I_matrix;
55.     I_matrix = Eigen::MatrixXd::Identity(totalTypes, totalTypes);
56.     IOFormat CleanFmt(5, 0, ", ", "\n");
57.
```

```
58.
59.     if (Inputs.is_open())
60.     {
61.         while ( !Inputs.eof() )
62.         {
63.             ofstream msinput("ms_input_file.txt", ofstream::trunc);
64.             while ( getline(Inputs,InputsLine) && InputsLine != ""  )
65.             {
66.                 msinput << InputsLine << endl;
67.             }
68.             nlines++;
69.             bso::spatial_design::ms_building MS("ms_input_file.txt");
70.
71.             // write BSD input to "inputBSD.txt" file, so it can be used as features
72.             string line;
73.             string dna;
74.             ifstream input("ms_input_file.txt");
75.             if (input.is_open())
76.             {
77.                 while ( getline (input, line) )
78.                 {
79.                     line.erase(0,4);
80.                     line.erase(line.end()-1, line.end());
81.                     dna += line;
82.                 }
83.                 dna.erase(dna.end()-1, dna.end());
84.                 ofstream inputBSDf("inputBSD.txt", ios::app);
85.                 inputBSDf << dna << endl;
86.                 inputBSDf.close();
87.
88.                 // dna is a string with several integers
89.                 // convert string to c++ integers
90.                 vector<int> inputBSD_values;
91.                 stringstream dna_stream(dna);
92.                 string item;
93.                 while ( getline(dna_stream, item, ',') )
94.                 {
95.                     inputBSD_values.push_back(stoi(item));
96.                 }
97.                 int width1  = inputBSD_values[0];    int width2  = inputBSD_values[6];
98.                 int depth1  = inputBSD_values[1];    int depth2  = inputBSD_values[7];
99.                 int height1 = inputBSD_values[2];    int height2 = inputBSD_values[8];
100.                  int x1      = inputBSD_values[3];    int x2      = inputBSD_values[9];
101.                  int y1      = inputBSD_values[4];    int y2      = inputBSD_values[10];
102.                  int z1      = inputBSD_values[5];    int z2      = inputBSD_values[11];
103.
104.                  // Number encoding and One-hot encoding are applied on the dataset
105.                  ofstream inputBSDf_OneHotEncoded("inputBSD_OneHotEncoded.txt", ios::app);
106.                  ofstream inputBSDf_NumberEncoded("inputBSD_NumberEncoded.txt", ios::app);
107.                  int w, d, h, x, y, z;
108.                  if (x1 +  width1 ==  x2 +  width2) { w = 0; }     if (x1 +  width1 < x2 +  width2) { w
     = 1; }     if (x1 +  width1 > x2 +  width2) { w = 2; }
109.                  if (y1 +  depth1 ==  y2 +  depth2) { d = 0; }     if (y1 +  depth1 < y2 +  depth2) { d
     = 1; }     if (y1 +  depth1 > y2 +  depth2) { d = 2; }
110.                  if (z1 + height1 ==  z2 + height2) { h = 0; }     if (z1 + height1 < z2 + height2) { h
     = 1; }     if (z1 + height1 > z2 + height2) { h = 2; }
111.                  if (          x1 ==            x2) { x = 0; }     if (          x1 <            x2) { x
     = 1; }     if (          x1 >            x2) { x = 2; }
112.                  if (          y1 ==            y2) { y = 0; }     if (          y1 <            y2) { y
     = 1; }     if (          y1 >            y2) { y = 2; }
113.                  if (          z1 ==            z2) { z = 0; }     if (          z1 <            z2) { z
     = 1; }     if (          z1 >            z2) { z = 2; }
114.                  int itype;
115.                  if (z2 == height1) {                                        // "top"
116.                      itype = 0;
117.                      for (int iw = 0; iw < 3; iw++)
118.                      {
119.                          if (iw != w) { itype+=27; continue; } else {
120.                              for (int id = 0; id < 3; id++)
121.                              {
122.                                  if (id != d) { itype+=9; continue; } else {
123.                                      for (int ix = 0; ix < 3; ix++)
124.                                      {
125.                                          if (ix != x) { itype+=3; continue; } else {
126.                                              for (int iy = 0; iy < 3; iy++)
127.                                              {
128.                                                  if (iy != y) { itype++; continue; }
```

```
                                                    else {  inputBSDf_OneHotEncoded << I_matrix.block(itype,0,1,totalTypes).format(CleanFmt) << "," << dna << endl;
                                                            inputBSDf_NumberEncoded << itype+1 << "," << dna << endl; }
                                                }}
                                            }}
                                        }}
                                    }
                                } else if (z2 == z1 && y2 == y1 - depth2) {               // "front"
                                    itype = 0 + topTypes;
                                    for (int iw = 0; iw < 3; iw++)
                                            {
                                                if (iw != w) { itype+=9; continue; } else {
                                                    for (int ih = 0; ih < 3; ih++)
                                                    {
                                                        if (ih != h) { itype+=3; continue; } else {
                                                            for (int ix = 0; ix < 3; ix++)
                                                            {
                                                                if (ix != x) { itype++; continue; }
                                                                else {  inputBSDf_OneHotEncoded << I_matrix.block(itype,0,1,totalTypes).format(CleanFmt) << "," << dna << endl;
                                                                        inputBSDf_NumberEncoded << itype+1 << "," << dna << endl; }
                                                            }}
                                                        }}
                                            }
                                } else if (z2 == z1 && y2 == y1 + depth1) {               // "behind"
                                    itype = 0 + topTypes + frontTypes;
                                    for (int iw = 0; iw < 3; iw++)
                                            {
                                                if (iw != w) { itype+=9; continue; } else {
                                                    for (int ih = 0; ih < 3; ih++)
                                                    {
                                                        if (ih != h) { itype+=3; continue; } else {
                                                            for (int ix = 0; ix < 3; ix++)
                                                            {
                                                                if (ix != x) { itype++; continue; }
                                                                else {  inputBSDf_OneHotEncoded << I_matrix.block(itype,0,1,totalTypes).format(CleanFmt) << "," << dna << endl;
                                                                        inputBSDf_NumberEncoded << itype+1 << "," << dna << endl; }
                                                            }}
                                                        }}
                                            }
                                } else if (z2 == z1 && x2 == x1 - width2) {               // "left"
                                    itype = 0 + topTypes + frontTypes + behindTypes;
                                    for (int id = 0; id < 3; id++)
                                            {
                                                if (id != d) { itype+=9; continue; } else {
                                                    for (int ih = 0; ih < 3; ih++)
                                                    {
                                                        if (ih != h) { itype+=3; continue; } else {
                                                            for (int iy = 0; iy < 3; iy++)
                                                            {
                                                                if (iy != y) { itype++; continue; }
                                                                else {  inputBSDf_OneHotEncoded << I_matrix.block(itype,0,1,totalTypes).format(CleanFmt) << "," << dna << endl;
                                                                        inputBSDf_NumberEncoded << itype+1 << "," << dna << endl; }
                                                            }}
                                                        }}
                                            }
                                } else if (z2 == z1 && x2 == x1 + width1) {               // "right"
                                    itype = 0 + topTypes + frontTypes + behindTypes + leftTypes;
                                    for (int id = 0; id < 3; id++)
                                            {
                                                if (id != d) { itype+=9; continue; } else {
                                                    for (int ih = 0; ih < 3; ih++)
                                                    {
                                                        if (ih != h) { itype+=3; continue; } else {
                                                            for (int iy = 0; iy < 3; iy++)
                                                            {
                                                                if (iy != y) { itype++; continue; }
                                                                else {  inputBSDf_OneHotEncoded << I_matrix.block(itype,0,1,totalTypes).format(CleanFmt) << "," << dna << endl;
                                                                        inputBSDf_NumberEncoded << itype+1 << "," << dna << endl; }
```

```
                                        }}
                                    }}
                                }
                            }
                            inputBSDf_OneHotEncoded.close();
                            inputBSDf_NumberEncoded.close();
                            input.close();
                        }
                        bso::spatial_design::cf_building CF(MS);

                        // Write corner-vertices of BSD to "cornerverticesBSD.txt" file
                        ofstream cornerverticesBSDf("cornerverticesBSD_ML.txt", ios::app);
                        int ip = 0;
                        for (const auto& i_cornerverticesBSD : CF.cfPoints())
                        {
                            for (const auto& j_cornerverticesBSD : *i_cornerverticesBSD)
                            {
                                if (ip == 0) {
                                    cornerverticesBSDf << j_cornerverticesBSD ;
                                } else {
                                    cornerverticesBSDf << "," << j_cornerverticesBSD  ;
                                }
                                ip++;
                            }
                        }
                        cornerverticesBSDf << endl;
                        cornerverticesBSDf.close();


                //-------------- CONFORMAL TRANSFORMATION --------------//
                //  Write targets to outputCF file (Conformal model described by corner-vertices)
                        ofstream outputCFf("outputCF.txt", ios::app);
                        if (transformation == "conformal")
                        {
                            int ii = 0;
                            if (geometry == "vertex")
                            {
                                for (const auto& i : CF.cfVertices())
                                {
                                    if (ii > 0) { outputCFf << "," ; }
                                    ii++;
                                    int jj = 0;
                                    for (const auto& j : *i)
                                    {
                                        if (jj == 0) {
                                            outputCFf << j ;
                                        } else if (jj >= 0) {
                                            outputCFf << "," << j ; }
                                        jj++;
                                    }
                                }
                            } else if (geometry == "line")
                            {
                                for (const auto& i : CF.cfLines())
                                {
                                    if (ii > 0) { outputCFf << "," ; }
                                    ii++;
                                    int jj = 0;
                                    for (const auto& j : *i)
                                    {
                                        if (jj == 0) {
                                            outputCFf << j[0] << "," << j[1] << "," << j[2];
                                        } else if (jj >= 0) {
                                            outputCFf << "," << j[0] << "," << j[1] << "," << j[2]; }
                                        jj++;
                                    }
                                }
                            }
                            else if (geometry == "rectangle")
                            {
                                for (const auto& i : CF.cfRectangles())
                                {
                                    if (ii > 0) { outputCFf << "," ; }
                                    ii++;
                                    int jj = 0;
                                    for (const auto& j : *i)
                                    {
```

```
                            if (jj == 0) {
                                outputCFf << j[0] << "," << j[1] << "," << j[2];
                            } else if (jj >= 0) {
                                outputCFf << "," << j[0] << "," << j[1] << "," << j[2]; }
                            jj++;
                        }
                    }
                }
                else if (geometry == "cuboid")
                {
                    for (const auto& i : CF.cfCuboids())
                    {
                        if (ii > 0) { outputCFf << "," ; }
                        ii++;
                        int jj = 0;
                        for (const auto& j : *i)
                        {
                            if (jj == 0) {
                                outputCFf << j[0] << "," << j[1] << "," << j[2];
                            } else if (jj >= 0) {
                                outputCFf << "," << j[0] << "," << j[1] << "," << j[2]; }
                            jj++;
                        }
                    }
                }
            }
            outputCFf << endl;
        }
    }
    Inputs.close();

    cout << "\n" << "End of simulations" << endl;
}


// --- MAIN --- //
int main(int argc, char* argv[])
{
  //---Simulation---//
  // DEFAULT:
  //    Input file          = "origin_dimensions_BSDs.txt"
  //    Transformation      = "conformal"
  //    Geometry entity     = "cuboid"
  //    {types included}    = {"top", "front", "behind", "left", "right"}
    simulation("origin_dimensions_BSDs_ML.txt", "conformal", "cuboid", {"top", "front", "behind", "left",
"right"} );
}

```

“NeuralNetwork_TRAIN.cpp”

```
// Include MLPACK library
#include <mlpack/core.hpp>
// Pre-processing Data
#include <mlpack/core/data/split_data.hpp>
#include <mlpack/core/data/scaler_methods/mean_normalization.hpp>
// Neural Network
#include <mlpack/methods/ann/layer/layer.hpp>
#include <mlpack/methods/ann/ffn.hpp>
// Loss function
#include <mlpack/methods/ann/loss_functions/mean_squared_error.hpp>
// Initialisation of the weights
#include <mlpack/methods/ann/init_rules/he_init.hpp>
// Activation function
#include <mlpack/methods/ann/layer/leaky_relu.hpp>
#include <mlpack/methods/ann/layer/linear.hpp>
// Optimisation function
#include <ensmallen.hpp>

// C++ References
#include <iostream>
#include <fstream>
#include <algorithm>
```

```
23.
24. // Namespaces
25. using namespace std;
26. using namespace mlpack;
27. using namespace mlpack::ann;
28. using namespace arma;
29. using namespace ens;
30.
31.
32. int main()
33. {
34.   // LOAD DATA
35.   mat features;
36.   data::Load("inputBSD.txt", features, false, true);
37.   mat targets;
38.   data::Load("outputCF.txt", targets, false, true);
39.
40.   // shuffle data
41.   mat data = move(join_cols(features, targets));
42.   mat shuffledData = shuffle(data,1);
43.   mat shuffledfeatures = shuffledData.rows(0,features.n_rows -1);
44.   mat shuffledtargets = shuffledData.rows(features.n_rows, features.n_rows + targets.n_rows -1);
45.
46.   shuffledfeatures.brief_print();
47.   shuffledtargets.brief_print();
48.
49.   // FEATURE SCALING
50.   mat shuffledscaledfeatures;
51.   mlpack::data::MeanNormalization scaler;
52.   scaler.Fit(shuffledfeatures);
53.   scaler.Transform(shuffledfeatures, shuffledscaledfeatures);
54.   shuffledscaledfeatures.brief_print();
55.
56.   // SPLIT DATA
57.   // Training & Testing Data
58.   mat train_features, test_features, train_targets, test_targets;
59.   mlpack::data::Split(shuffledscaledfeatures, train_features, test_features, 0.2, false);
      // 80% train, 20% test
60.   mlpack::data::Split(shuffledtargets , train_targets,  test_targets,  0.2, false);
61.
62.
63.   // BUILD ML MODEL
64.   FFN<MeanSquaredError<>, HeInitialization>  model;
65.   model.Add<Linear<> >(shuffledfeatures.n_rows, 180);                                    //
    Input layer --> Hidden layer
66.   model.Add<LeakyReLU<> >(0.0001);
              // Activation function on hidden layer
67.   model.Add<Linear<> >(180, shuffledtargets.n_rows);                                     //
    Hidden layer --> Output layer
68.   model.Add<LeakyReLU<> >(0.0001);
              // Activation function on output layer
69.
70.
71.   // TRAIN & TEST
72.   // Define optimisation function: Adam(Learning rate, Batch size)
73.   ens::Adam optimizer(0.01, 100);
74.   // Mean Squared Error (MSE) is saved in text file
75.   ofstream MSE("MSE.txt", ios::trunc);
76.   // t: training iterations
77.   int t = 12;
78.   for (int i = 0; i < t; i++)
79.   {
80.             // Train
81.             model.Train(train_features, train_targets, optimizer);
82.             // Evaluate train & test
83.             cout << i+1 << ") Evaluate Train: " << model.Evaluate(train_features, train_targets) << endl;
84.             MSE << model.Evaluate(train_features, train_targets) << "," << model.Evaluate(test_features,
    test_targets) << endl;
85.             cout << i+1 << ") Evaluate Test : " << model.Evaluate(test_features, test_targets) << endl;
86.   }
87.
88.
89.   // SAVE model
90.   data::Save("trained_ML_model.xml", "model", model, false);
91.
92.
93.
```

```
94.   return 0;
95. }
96.
```

“NeuralNetwork_PREDICT.cpp”

```
1.  // Include MLPACK library
2.  #include <mlpack/core.hpp>
3.  // Pre-processing Data
4.  #include <mlpack/core/data/split_data.hpp>
5.  #include <mlpack/core/data/scaler_methods/mean_normalization.hpp>
6.  // Neural Network
7.  #include <mlpack/methods/ann/layer/layer.hpp>
8.  #include <mlpack/methods/ann/ffn.hpp>
9.  // Loss function
10. #include <mlpack/methods/ann/loss_functions/mean_squared_error.hpp>
11. // Initialisation of the weights
12. #include <mlpack/methods/ann/init_rules/he_init.hpp>
13. // Activation function
14. #include <mlpack/methods/ann/layer/leaky_relu.hpp>
15. #include <mlpack/methods/ann/layer/linear.hpp>
16. // Optimisation function
17. #include <ensmallen.hpp>
18.
19. // C++ References
20. #include <iostream>
21. #include <fstream>
22. #include <algorithm>
23.
24.
25. using namespace std;
26. using namespace mlpack;
27. using namespace mlpack::ann;
28. using namespace arma;
29. using namespace ens;
30.
31.
32. int main()
33. {
34.   // LOAD trained ML model
35.   FFN<MeanSquaredError<>, HeInitialization>  model;
36.   data::Load("trained_ML_model.xml", "model", model);
37.
38.
39.   // LOAD features of test building that needs to be predicted
40.   mat featuresTest; data::Load("test_inputBSD.txt", featuresTest);
41.
42.
43.   // SCALE features (same as features where model is trained on: "inputBSD.txt")
44.   mat features; data::Load("inputBSD.txt", features);
45.   mat scaledfeaturesTest;
46.   mlpack::data::MeanNormalization scaler;
47.   scaler.Fit(features);
48.   scaler.Transform(featuresTest, scaledfeaturesTest);
49.   scaledfeaturesTest.print();
50.
51.
52.   // PREDICT conformal model
53.   mat prediction;
54.   model.Predict(scaledfeaturesTest, prediction);
55.
56.   cout << fixed << setprecision(0);
57.   trans(prediction).raw_print(cout, "prediction Target: ");     cout << endl;
58.
59.
60.   // Write prediction to file
61.   ofstream predictionf("predictionCF.txt", ios::trunc);
62.   for (int i = 0; i < prediction.size(); i++)
63.   {
64.             if (i < prediction.size() - 1) {
65.                       predictionf << fixed << setprecision(0) << prediction[i] << endl;
66.             }
67.             else {
68.                       predictionf << fixed << setprecision(0) << prediction[i];
```

```
            }
    }


    return 0;
}

```

“visualisation_ML.cpp”

```
// Visualization BSD and Conformal model

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <cmath>

using namespace std;


// --- predictionCF_ML Function ---
//     : create multidimensional vector with predicted coordinates of conformal model
vector<vector<vector<float>>> predictionCF_ML()
{
    // Load predicted data
    vector<string> predictedCoordinatesString;
    string coordinate;
    ifstream predictC("predictionCF.txt");
    while ( !predictC.eof() )
    {
        while ( getline(predictC,coordinate) )
        {
            predictedCoordinatesString.push_back(coordinate) ;
        }
    }
    // convert string to Int vector
    vector<float> predictedCoordinatesFl;
    for (int i=0; i<predictedCoordinatesString.size(); i++)
    {
        float num = atoi(predictedCoordinatesString.at(i).c_str());
        predictedCoordinatesFl.push_back(num);
    }

    // Create multidimensional vector, each individual quad-hexahedron is listed in a vector
    vector<vector<vector<float>>> ncoordinates;
    int p = 0;
    for (int c = 0; c < predictedCoordinatesFl.size()/24; c++)              // number of quad-hexahedrons
    {
        vector<vector<float>> cuboidVec;
        for (int j = 0; j < 24; j+=3)                                       // number of numerical values per cuboid
        {
            vector<float> pointVec = {predictedCoordinatesFl[p+j+0], predictedCoordinatesFl[p+j+1], predictedCoordinatesFl[p+j+2]};
            cuboidVec.push_back(pointVec);
        }
        ncoordinates.push_back(cuboidVec);
        p+=24;
    }

    return ncoordinates;            // return coordinates to use in VISUALIZATION
}
vector<float> allvertices()
{
    vector<string> predictedCoordinatesString;
    string coordinate;
    ifstream predictC("predictionCF.txt");
    while ( !predictC.eof() )
    {
        while ( getline(predictC,coordinate) )
        {
```

```
61.             predictedCoordinatesString.push_back(coordinate) ;
62.         }
63.     }
64.     // convert string to Int vector
65.     vector<float> predictedCoordinatesFl;
66.     for (int i=0; i<predictedCoordinatesString.size(); i++)
67.     {
68.         float num = atoi(predictedCoordinatesString.at(i).c_str());
69.         predictedCoordinatesFl.push_back(num);
70.     }
71.     // Create multidimensional vector, each individual cuboid is listed in a vector
72.     vector<vector<vector<float>>> ncoordinates;
73.     int p = 0;
74.     for (int c = 0; c < predictedCoordinatesFl.size()/24; c++)              // number of cuboids
75.     {
76.         vector<vector<float>> cuboidVec;
77.         for (int j = 0; j < 24; j+=3)                                        // number of numerical val-
    ues per cuboid
78.         {
79.             vector<float> pointVec = {predictedCoordinatesFl[p+j+0], predictedCoordinatesFl[p+j+1], pre-
    dictedCoordinatesFl[p+j+2]};
80.             cuboidVec.push_back(pointVec);
81.         }
82.         ncoordinates.push_back(cuboidVec);
83.         p+=24;
84.     }
85.     return predictedCoordinatesFl;            // return coordinates to use in VISUALIZATION
86. }
87.
88.
89.
90. //--------------------------------------------------------------------------------------------------------
91. // VISUALIZE Rectangles with OpenGL/glut                    ***         https://gist.github.com/hkule-
    kci/2300262
92. //--------------------------------------------------------------------------------------------------------
93. #include <stdio.h>
94. #include <GL/glut.h>
95. #include <GL/freeglut.h>
96. #define KEY_ESC 27 /* GLUT doesn't supply this */
97.
98.
99. // --- Initial parameters ---
100.  // mouse operation: rotate around object
101.      int mouseDown = 0;
102.      float xrot = 10.0f;
103.      float yrot = -10.0f;
104.      float xdiff = 10.0f;
105.      float ydiff = 10.0f;
106.  // translate shape
107.      float tra_x = 0.0f;
108.      float tra_y = 0.0f;
109.      float tra_z = 0.0f;
110.  // View
111.      float grow_shrink = 72.0f;                  // fovy & zoom in/out
112.      float resize_f = 1.0f;
113.  // Output screen
114.      #define SCREEN_WIDTH 640
115.      #define SCREEN_HEIGHT 480
116.
117.
118.  // --- DRAW ---
119.
120.  void drawVertices()
121.  {
122.      vector<float> coorN = allvertices();
123.      int norm = 10000;
124.      for (int c = 0; c < coorN.size(); c++)
125.      {
126.          coorN[c] = coorN[c] / norm;
127.      }
128.
129.      //glTranslatef(tra_x-0.3f, tra_y-0.3f, tra_z-0.3f);
130.      //glTranslatef(tra_x, tra_y, tra_z);
131.
132.      glPointSize(8.0f);  //10?
133.      glBegin(GL_POINTS);
134.      glColor3f(0.0f, 0.0f, 0.0f);
```

```
135.
136.      for (int c = 0; c < coorN.size(); c+=3)
137.      {
138.          glVertex3f(coorN[c+0],     coorN[c+2],     -coorN[c+1]);
139.      }
140.
141.      glEnd();
142.  }
143.
144.  void drawBuildingFILL()
145.  {
146.          vector<vector<vector<float>>> coorN = predictionCF_ML();                    // call coordi-
    nates vector from predictionCF_ML() Function
147.          int norm = 10000;                                          // divide by 10000, all numbers between
    0 and 1 (normalized)
148.          for (int c = 0; c < coorN.size(); c++)
149.          {
150.              for (int p = 0; p < 8; p++)
151.              {
152.                  for (int j = 0; j < 3; j++)
153.                  {
154.                      coorN[c][p][j] = coorN[c][p][j] / norm;
155.                  }
156.              }
157.          }
158.          //glTranslatef(tra_x-0.3f, tra_y-0.3f, tra_z-0.3f);                       // translate shape to
    middle of the screen
159.          glTranslatef(tra_x, tra_y, tra_z);
160.          glBegin(GL_QUADS);
161.
162.          // Color(RED, GREEN, BLUE)    0,0,0 = BLACK
163.          //glColor3f(0.8f, 0.8f, 0.8f);       // INPUT: light grey
164.          glColor3f(0.6f, 0.7f, 0.7f);       // OUTPUT: middle light blue-grey
165.          //glColor3f(0.4f, 0.5f, 0.6f);       // PREDICTION: blue-grey
166.
167.          int nShape = 13;
168.          int x = 0; int y = 2; int z = 1;
169.          for (int c = 0; c < coorN.size(); c++)
170.          {
171.              for (int p = 0; p < 8; p++)
172.              {
173.                  glVertex3f(coorN[c][p][x],coorN[c][p][y],-coorN[c][p][z]);
174.              }
175.              for (int p = 0; p < 4; p++)
176.              {
177.                  if (p < 3){
178.                  glVertex3f(coorN[c][p][x],coorN[c][p][y],-coorN[c][p][z]);
179.                  glVertex3f(coorN[c][p+1][x],coorN[c][p+1][y],-coorN[c][p+1][z]);
180.                  glVertex3f(coorN[c][p+5][x],coorN[c][p+5][y],-coorN[c][p+5][z]);
181.                  glVertex3f(coorN[c][p+4][x],coorN[c][p+4][y],-coorN[c][p+4][z]);
182.                  } else {
183.                  glVertex3f(coorN[c][p][x],coorN[c][p][y],-coorN[c][p][z]);
184.                  glVertex3f(coorN[c][p+1-4][x],coorN[c][p+1-4][y],-coorN[c][p+1-4][z]);
185.                  glVertex3f(coorN[c][p+5-4][x],coorN[c][p+5-4][y],-coorN[c][p+5-4][z]);
186.                  glVertex3f(coorN[c][p+4][x],coorN[c][p+4][y],-coorN[c][p+4][z]);
187.                  }
188.              }
189.          }
190.          glEnd();
191.  }
192.  void drawBuildingLINE()
193.  {
194.          vector<vector<vector<float>>> coorN = predictionCF_ML();                    // call coordi-
    nates vector from predictionCF_ML() Function
195.          float norm = 10000;                                          // divide by 10000, all numbers be-
    tween 0 and 1 (normalized)
196.          for (int c = 0; c < coorN.size(); c++)
197.          {
198.              for (int p = 0; p < 8; p++)
199.              {
200.                  for (int j = 0; j < 3; j++)
201.                  {
202.                      coorN[c][p][j] = coorN[c][p][j] / norm;
203.                  }
204.              }
205.          }
206.          glBegin(GL_QUADS);
```

```
207.          glColor3f(0.0f, 0.0f, 0.0f);     // black
208.
209.          int nShape = 13;
210.          int x = 0; int y = 2; int z = 1;
211.          for (int c = 0; c < coorN.size(); c++)
212.          {
213.              for (int p = 0; p < 8; p++)
214.              {
215.                  glVertex3f(coorN[c][p][x],coorN[c][p][y],-coorN[c][p][z]);
216.              }
217.              for (int p = 0; p < 4; p++)
218.              {
219.                  if (p < 3){
220.                  glVertex3f(coorN[c][p][x],coorN[c][p][y],-coorN[c][p][z]);
221.                  glVertex3f(coorN[c][p+1][x],coorN[c][p+1][y],-coorN[c][p+1][z]);
222.                  glVertex3f(coorN[c][p+5][x],coorN[c][p+5][y],-coorN[c][p+5][z]);
223.                  glVertex3f(coorN[c][p+4][x],coorN[c][p+4][y],-coorN[c][p+4][z]);
224.                  } else {
225.                  glVertex3f(coorN[c][p][x],coorN[c][p][y],-coorN[c][p][z]);
226.                  glVertex3f(coorN[c][p+1-4][x],coorN[c][p+1-4][y],-coorN[c][p+1-4][z]);
227.                  glVertex3f(coorN[c][p+5-4][x],coorN[c][p+5-4][y],-coorN[c][p+5-4][z]);
228.                  glVertex3f(coorN[c][p+4][x],coorN[c][p+4][y],-coorN[c][p+4][z]);
229.                  }
230.              }
231.          }
232.          glEnd();
233.  }
234.
235.
236.  // --- INITIALIZATION ---
237.  int init(void)
238.  {
239.            glClearColor(1.0f, 1.0f, 1.0f, 1.0f);                    // background color(1f, 1f, 1f) &
    opaque(transparency)
240.            //glEnable(GL_DEPTH_TEST);                                    // enable depth-test, which
    shape is in front of another?
241.            //glDepthFunc(GL_LEQUAL);                                     // type of depth-test: LESS or
    EQUAL: Passes if the fragment's depth value is less than or equal to the stored depth value.
242.            //glClearDepth(1.0f);
243.
244.      //glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);                  // wireframe view (GL_LINE), or
    GL_POINT or GL_FILL
245.      glShadeModel(GL_SMOOTH);                                      // shading: _SMOOTH or _FLAT
246.
247.      // glEnable(GL_CULL_FACE);
248.      // glCullFace(GL_BACK);
249.
250.      // glEnable(GL_BLEND);
251.      // glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
252.
253.            return 1;
254.  }
255.
256.
257.  // --- DISPLAY ---
258.  void display(void)
259.  {
260.            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
261.            glLoadIdentity();
262.
263.      // camera position:
264.            gluLookAt(  0.0f, 0.0f, 1.0f,           // EYE
265.                        0.0f, 0.0f, -0.5f,           // AT
266.                        0.0f, 1.0f, 0.0f    );      // UP, corresponds to y-axis
267.
268.            glRotatef(xrot, 1.0f, 0.0f, 0.0f);
269.            glRotatef(yrot, 0.0f, 1.0f, 0.0f);
270.
271.
272.      glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
273.      drawBuildingFILL();
274.      glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
275.      drawBuildingLINE();
276.      glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
277.      drawVertices();
278.
279.            glFlush();
```

```
280.            glutSwapBuffers();
281.  }
282.
283.
284.  // --- RESHAPE ---
285.  void resize(int w, int h)
286.  {
287.      // MATRIXMODE( "specifies which stack is the target for subsequent matrix operations.")
288.      //              mode: MODELVIEW, PROJECTION, TEXTURE, COLOR
289.            glMatrixMode(GL_PROJECTION);               // To operate on the Projection Matrix
290.            glLoadIdentity();
291.
292.      // VIEWPORT(     x, y:           lower left corner of the viewport rectangle, in pixels. Initial =
    (0,0),
293.      //              width, height:  of the viewport. When a GL context is first attached to a window,
    width and height are set to the dimensions of that window
294.            glViewport(0, 0, w, h);
295.
296.      // Set up a perspective projection matrix
297.      // PERSPECTIVE(  fovy:   field of view angle,
298.      //               aspect: ratio of x (width) to y (height),
299.      //               zNear:  distance from viewer to near clipping plane (always positive),
300.      //               zFar:   distance from viewer to far  clipping plane (always positive)   )
301.            //gluPerspective(grow_shrink, resize_f*SCREEN_WIDTH/SCREEN_HEIGHT, 0.0f, 2.0f);
302.      // ORTHOGRPHIC( left, right, bottom, top, nearVal, farVal )
303.      glOrtho(-1.5, 1.5, -1.5, 1.5, 0, 2);
304.
305.            glMatrixMode(GL_MODELVIEW);
306.            glLoadIdentity();
307.  }
308.
309.
310.
311.  // --- VIEW OPTIONS ---
312.  // ZOOM VIEW:  IN (I) / OUT (O)
313.  void keyboard(unsigned char key, int x, int y)
314.  {
315.            switch(key)
316.            {
317.                      case 27 :
318.                                exit(1);
319.                                break;
320.
321.          // FORWARD
322.          case 'w':
323.                                tra_z += 0.1f;
324.                                break;
325.          // BACKWARD
326.                      case 's':
327.                                tra_z -= 0.1f;
328.                                break;
329.          // LEFT
330.                      case 'a':
331.                                tra_x -= 0.1f;
332.                                break;
333.          // RIGHT
334.                      case 'd':
335.                                tra_x += 0.1f;
336.                                break;
337.
338.          // Zoom OUT
339.                      case 'o':
340.                                grow_shrink++;
341.                                resize(SCREEN_WIDTH, SCREEN_HEIGHT);
342.                                break;
343.          // Zoom IN
344.                      case 'i':
345.                                grow_shrink--;
346.                                resize(SCREEN_WIDTH, SCREEN_HEIGHT);
347.                                break;
348.            }
349.            glutPostRedisplay();
350.  }
351.
352.  // ORBIT VIEW:   Rotate with left mouse button around object shape
353.  void mouse(int button, int state, int x, int y)
354.  {
```

```
355.            if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
356.            {
357.                    mouseDown = 1;
358.
359.                    xdiff = x - yrot;
360.                    ydiff = -y + xrot;
361.            }
362.            else
363.                    mouseDown = 0;
364.    }
365.    void mouseMotion(int x, int y)
366.    {
367.            if (mouseDown)
368.            {
369.                    yrot = x - xdiff;
370.                    xrot = y + ydiff;
371.
372.                    glutPostRedisplay();
373.            }
374.    }
375.
376.
377.
378.
379.    // --- MAIN FUNCTION ---
380.    int main(int argc, char *argv[])
381.    {
382.            glutInit(&argc, argv);
383.
384.            glutInitWindowPosition(0, 0);
385.            glutInitWindowSize(SCREEN_WIDTH, SCREEN_HEIGHT);
386.
387.            glutInitDisplayMode(GLUT_RGB);
388.
389.            glutCreateWindow("--- Building ---");
390.
391.            glutDisplayFunc(display);
392.        glutKeyboardFunc(keyboard);
393.            glutMouseFunc(mouse);
394.            glutMotionFunc(mouseMotion);
395.            glutReshapeFunc(resize);
396.
397.            if (!init())
398.                    return 1;
399.
400.            glutMainLoop();
401.
402.            return 0;
403.    }
404.
```

D.2 Genetic Algorithm code files

The GA code consists of two individual C++ files. One for the both GAs, and one for the visualisation of the BSD and CF model. First, "GA1and2.cpp" is executed to run the evolutionary process and to find a conformal representation of the input BSD. Secondly, "visualisation_GA.cpp" can be executed to visualise the generated conformal model by the GA.

- "GA1and2.cpp"
- "visualisation_GA.cpp"

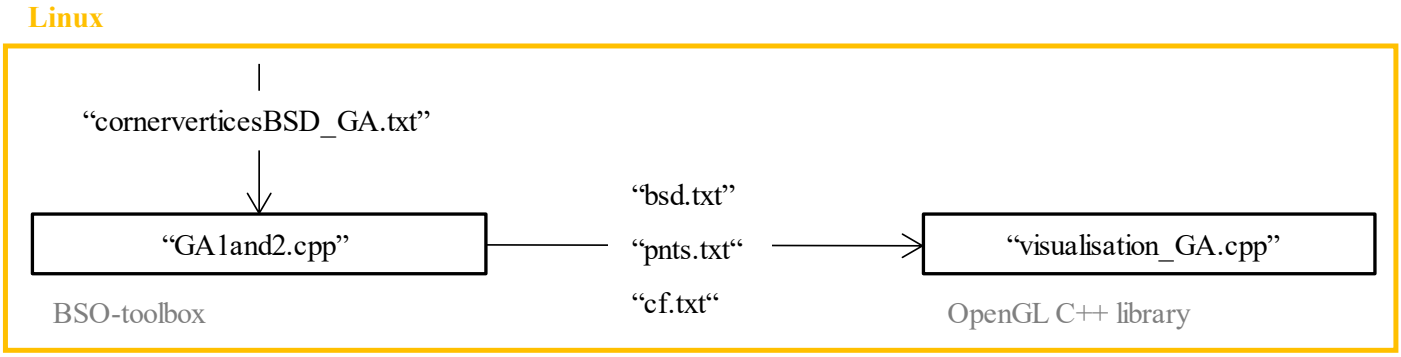


Figure D3. Overview of all files related to GA method with included libraries.

Also for the GA method, an example is added. Here, test building 'A-O' is used as input to make conformal by the GA.

D.2.1 "GA1and2.cpp"

Both GAs are structured in the same file. The simulations of GA1 and GA2 are ran in succession.

Input and output files

The corner-vertices (**p**) of the spaces of a BSD ("cornerverticesBSD_GA.txt") are used as input for the GA. If a perfect conformal geometry is found by the GA, it is saved in "cf.txt"-file, and can be used for visualisation. Additionally, the BSD corner-vertices are saved in "bsd.txt", and the point cloud is saved in "pnts.txt". All can be used as input for "visualisation_GA.cpp".

Input text-files:

- "cornerverticesBSD_GA.txt"

Output text-files:

- "bsd.txt"
- "pnts.txt"
- "cf.txt"

User-defined parameters

Parameters exist in both GAs that can be changed by the user. The default values for following example are listed below.

For GA1:

- initial Population (N) = 512
- stopping criteria (n) = 100 (after n generations where no new unique Quad-Hexahedron is found, GA1 stops)

For GA2:

- initial Population (M) = 8
- probability_1_min = 10 (minimal probability of ones in chromosome)
- probability_1_max = 100 (maximal probability of ones in chromosome)
- mutation rate population = 0.5 (50% of population undergoes mutation)
- mutation rate individual = 1 (nr. of bit flips in chromosome)

Example
In the example, Test building 'A-O' from Section 3.4 is used to make conformal by the GA.
Test building 'A-O' is described in the text-file "cornerverticesBSD_GA.txt" as follows:

- N, 1, 10,10,0, 16,10,0, 16,16,0, 10,16,0, 10,10,3, 16,10,3, 16,16,3, 10,16,3
- N, 2, 16,12,0, 22,12,0, 22,18,0, 16,18,0, 16,12,5, 22,12,5, 22,18,5, 16,18,5

In this example, the dimensions and coordinates of the test building are indicated in meters.

The file can be compiled by typing the following command in the terminal:
`make clean ga`
The file can be executed by typing the following command in the terminal:
`./algo`

Finally, Test building 'A-O' is made conformal by the Genetic Algorithm file "GA1and2.cpp". The results are listed below.

Table D4. Results of example GA.

Test building	generations GA1	generations GA2	nr. of quad-hexahedrons in CF model
'A-O'	200	5	6

D.2.2 "visualisation_GA.cpp"
The BSD, point cloud, and conformal model can be visualised. The input for the visualisation is the "bsd.txt", "pnts.txt", or "cf.txt" text-file.

Example
The conformal model of Test Building 'A-O' is shown in Figure D4. One can see the generated conformal model by the GA ("cf.txt") and point cloud ("pnts.txt").

The file can be compiled by typing the following command in the terminal:
`make clean viga`
The file can be executed by typing the following command in the terminal:
`./visualGA`

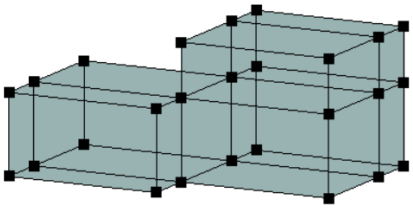


Figure D4. Visualisation of predicted conformal model Test building 'A-O'

“GA1and2.cpp”

```
1.  //--- C References ---//
2.  #include <iostream>
3.  #include <fstream>
4.  #include <sstream>
5.  #include <string>
6.  #include <cmath>
7.  #include <vector>
8.  #include <algorithm>
9.  #include <map>
10. #include <iterator>
11. #include <set>
12. #include <unordered_map>
13.
14. #include <limits>
15. #include <numeric>
16. #include <stdexcept>
17. #include <tuple>
18. #include <utility>
19.
20. #include <bso/spatial_design/ms_space.hpp>
21. #include <bso/spatial_design/ms_building.hpp>
22. #include <bso/utilities/geometry.hpp>
23.
24.
25. // namespaces
26. using namespace std;
27.
28. // ------------------------------ //
29. // --- GA1 : Quad-Hexahedrons --- //
30. // ------------------------------ //
31.
32. // collect coordinates (existing x, y, z values) of the building spatial design (BSD)
33. // INPUT    = "testDesign" : BSD described by corner-vertices
34. // OUTPUT   + nSpaces : nr. of spaces in BSD
35. //          + total_volume_BSD
36. //          + hexaSpaces : spaces of BSD described by quad-hexahedrons according to the BSO-toolbox
    ("ms_space")
37. //          + x,y,z_coordinates : vectors with integers of all existing x, y, z values. Duplicates are
    removed by using the map container.
38. //          + "ms.txt" : text file with corner-vertices of BSD. This is used for "visualisation.cpp"
39. void collectCoordinates( string inputsFileBSD,
40.                           int &nSpaces, double &total_volume_MS,
41.                           vector<bso::utilities::geometry::quad_hexahedron> &hexaSpaces,
42.                           vector<int> &x_coordinates, vector<int> &y_coordinates, vector<int> &z_coor-
    dinates)
43. {
44.     total_volume_MS = 0.0;  nSpaces = 0;
45.     map<int, int> x_coordinates_map;    map<int, int> y_coordinates_map;    map<int, int> z_coordi-
    nates_map;
46.     string line;
47.     ifstream input(inputsFileBSD);
48.     ofstream inputBSD("bsd.txt");
49.     if (input.is_open())
50.     {
51.         while ( getline (input, line) )
52.         {
53.             nSpaces++;
54.             bso::spatial_design::ms_space MS(line);
55.             bso::utilities::geometry::quad_hexahedron cornerpoints = MS.getGeometry();
56.             hexaSpaces.push_back(cornerpoints);
57.             total_volume_MS += cornerpoints.getVolume();
58.             for (int p = 0; p < 8; p++) {
59.                 x_coordinates_map[cornerpoints.getVertices()[p][0]] = cornerpoints.getVertices()[p][0];
60.                 y_coordinates_map[cornerpoints.getVertices()[p][1]] = cornerpoints.getVertices()[p][1];
61.                 z_coordinates_map[cornerpoints.getVertices()[p][2]] = cornerpoints.getVertices()[p][2];
62.                 inputBSD << cornerpoints.getVertices()[p][0] << "\n" << cornerpoints.getVertices()[p][1]
    << "\n" << cornerpoints.getVertices()[p][2] << endl;
63.             }
64.         }
65.     }
66.     typedef map<int, int> MapType;
67.     for( MapType::iterator it = x_coordinates_map.begin(); it != x_coordinates_map.end(); ++it )
68.     { x_coordinates.push_back(it->second); }
69.     for( MapType::iterator it = y_coordinates_map.begin(); it != y_coordinates_map.end(); ++it )
```

```
70.     { y_coordinates.push_back(it->second); }
71.     for( MapType::iterator it = z_coordinates_map.begin(); it != z_coordinates_map.end(); ++it )
72.     { z_coordinates.push_back(it->second); }
73. }
74. 
75. // get all vertices to create point cloud grid.
76. // INPUT    = nSpaces, hexaSpaces, x,y,z_coordinates
77. // OUTPUT   + allVertices : Make all possible combinations of the previously defined x,y,z values. Dupli-
    cates are removed.
78. //                        AND vertices outside the BSD spaces are removed.
79. //          + "pnts.txt" : text file with vertices of Point cloud. This is used for "visualisation.cpp"
80. void generatePointCloud(    int nSpaces, vector<bso::utilities::geometry::quad_hexahedron> hexaSpaces,
81.                            vector<int> x_coordinates, vector<int> y_coordinates, vector<int> z_coordi-
    nates,
82.                            vector<vector<double>> &PointCloud)
83. {
84.     map<vector<double>, vector<double>> PointCloud_map;
85.     for (int ix = 0; ix < x_coordinates.size(); ix++) {
86.         for (int iy = 0; iy < y_coordinates.size(); iy++) {
87.             for (int iz = 0; iz < z_coordinates.size(); iz++)
88.             {
89.                 PointCloud_map[{double(x_coordinates[ix]), double(y_coordinates[iy]), double(z_coordi-
    nates[iz])}] = {double(x_coordinates[ix]), double(y_coordinates[iy]), double(z_coordinates[iz])};
90.             }
91.         }
92.     }
93.     ofstream PointCloudf("pnts.txt");
94.     typedef map<vector<double>, vector<double>> MapbType;
95.     vector<vector<double>> PointCloudv;
96.     double mTol = 1e-3;
97.     int a = 0;
98.     for( MapbType::iterator it = PointCloud_map.begin(); it != PointCloud_map.end(); ++it )
99.     {
100.         // if point is Inside or On BSD space[i], add to PointCloud:
101.         bso::utilities::geometry::vertex point = {it->second[0], it->second[1], it->second[2]};
102.         for (int s = 0; s < nSpaces; s++)
103.         {
104.             if ( hexaSpaces[s].isInsideOrOn(point, mTol) )
105.             {
106.                 PointCloudv.push_back( it->second );
107.                 PointCloudf << it->second[0] << endl << it->second[1] << endl << it->second[2] << endl;
108.                 break;
109.             }
110.         }
111.     }
112.     cout << "total unique vertices in Point cloud (inside BSD spaces):    " << PointCloudv.size() <<
    endl;
113.     PointCloud = PointCloudv;
114. }
115. 
116. 
117. // Individuals of GA1: Quad-hexahedrons described by a Class.
118. //          + points8 : combination of 8 vertices that form the quad-hexahedron
119. //          + fitness function GA1
120. //              - Orthogonal designs:       (O)
121. //              - Non-Orthogonal designs:   (NO)
122. //          + Convexity : check if quad-hexahedron is convex or not.
123. class individual_GA1 {
124.     public:
125.         vector<vector<double>> points8;
126. 
127.         double fitness_GA1() {
128.             double fitness;
129. 
130.         // ------------ begin fitness Orthogonal --------------- //
131.         //
132.         //     int score_nx = 0; int score_ny = 0; int score_nz = 0;
133.         //  // X:
134.         //     if ((points8[0][0] == points8[1][0]) && (points8[0][0] == points8[2][0]) &&
    (points8[0][0] == points8[3][0]) ) {score_nx++  ;}
135.         //     if ((points8[4][0] == points8[5][0]) && (points8[4][0] == points8[6][0]) &&
    (points8[4][0] == points8[7][0]) ) {score_nx++  ;}
136.         //     if (points8[0][0] == points8[4][0]) {score_nx-=1 ;}     // avoid line instead of rectan-
    gle
137.         //  // Y:
138.         //     if ((points8[0][1] == points8[1][1]) && (points8[0][1] == points8[4][1]) &&
    (points8[0][1] == points8[5][1]) ) {score_ny++  ;}
```

```
139.          //     if ((points8[2][1] == points8[3][1]) && (points8[2][1] == points8[6][1]) &&
    (points8[2][1] == points8[7][1]) ) {score_ny++  ;}
140.          //     if (points8[0][1] == points8[2][1]) {score_ny-=1 ;}
141.          //  // Z:
142.          //     if ((points8[0][2] == points8[2][2]) && (points8[0][2] == points8[4][2]) &&
    (points8[0][2] == points8[6][2]) ) {score_nz++  ;}
143.          //     if ((points8[1][2] == points8[3][2]) && (points8[1][2] == points8[5][2]) &&
    (points8[1][2] == points8[7][2]) ) {score_nz++  ;}
144.          //     if (points8[0][2] == points8[1][2]) {score_nz-=1 ;}
145.
146.          //     fitness =  4 + score_nx + score_ny + score_nz ;       ;
147.          // ------------ end   fitness Orthogonal --------------- //
148.
149.
150.          // ------------ begin fitness NON-Orthogonal --------------- //
151.              int score_nx = 0; int score_ny = 0; int score_nz = 0;
152.          // Walls are VERTICAL
153.           // X:
154.              if (points8[0][0] == points8[1][0]) {score_nx++  ;}
155.              if (points8[2][0] == points8[3][0]) {score_nx++  ;}
156.              if (points8[4][0] == points8[5][0]) {score_nx++  ;}
157.              if (points8[6][0] == points8[7][0]) {score_nx++  ;}
158.              // prevent duplicates, 4 unique 'columns'
159.              if ( (points8[0][0] == points8[2][0]) && (points8[0][0] == points8[4][0]) && (points8[0][0]
    == points8[6][0]) ) {score_nx-=1 ;}
160.              // avoid line (3 points on one) instead of rectangle
161.              if ((points8[0][0] == points8[2][0]) && (points8[0][0] == points8[4][0])) {score_nx-=1 ;}
162.              if ((points8[0][0] == points8[2][0]) && (points8[0][0] == points8[6][0])) {score_nx-=1 ;}
163.              if ((points8[0][0] == points8[4][0]) && (points8[0][0] == points8[6][0])) {score_nx-=1 ;}
164.              if ((points8[2][0] == points8[4][0]) && (points8[2][0] == points8[6][0])) {score_nx-=1 ;}
165.           // Y:
166.              if (points8[0][1] == points8[1][1]) {score_ny++  ;}
167.              if (points8[2][1] == points8[3][1]) {score_ny++  ;}
168.              if (points8[4][1] == points8[5][1]) {score_ny++  ;}
169.              if (points8[6][1] == points8[7][1]) {score_ny++  ;}
170.              if ( (points8[0][1] == points8[2][1]) && (points8[0][1] == points8[4][1]) && (points8[0][1]
    == points8[6][1]) ) {score_ny-=1 ;}
171.              if ((points8[0][1] == points8[2][1]) && (points8[0][1] == points8[4][1])) {score_ny-=1 ;}
172.              if ((points8[0][1] == points8[2][1]) && (points8[0][1] == points8[6][1])) {score_ny-=1 ;}
173.              if ((points8[0][1] == points8[4][1]) && (points8[0][1] == points8[6][1])) {score_ny-=1 ;}
174.              if ((points8[2][1] == points8[4][1]) && (points8[2][1] == points8[6][1])) {score_ny-=1 ;}
175.
176.          // Floors are HORIZONTAL
177.           // Z:
178.              if ((points8[0][2] == points8[2][2]) && (points8[0][2] == points8[4][2]) && (points8[0][2]
    == points8[6][2]) ) {score_nz++  ;}
179.              if ((points8[1][2] == points8[3][2]) && (points8[1][2] == points8[5][2]) && (points8[1][2]
    == points8[7][2]) ) {score_nz++  ;}
180.              if (points8[0][2] == points8[1][2]) {score_nz-=1 ;}
181.
182.          // CHECK 3 vertices on one LINE
183.              bso::utilities::geometry::vertex vx1 = {points8[0][0], points8[0][1], points8[0][2]};
184.              bso::utilities::geometry::vertex vx2 = {points8[2][0], points8[2][1], points8[2][2]};
185.              bso::utilities::geometry::vertex vx3 = {points8[4][0], points8[4][1], points8[4][2]};
186.              bso::utilities::geometry::vertex vx4 = {points8[6][0], points8[6][1], points8[6][2]};
187.              int score_nL = 0;
188.              bso::utilities::geometry::line_segment lineA = {vx1, vx2};
189.              bso::utilities::geometry::line_segment lineB = {vx2, vx4};
190.              bso::utilities::geometry::line_segment lineC = {vx4, vx3};
191.              bso::utilities::geometry::line_segment lineD = {vx3, vx1};
192.              bso::utilities::geometry::vector vectorA = lineA.getVector();
193.              bso::utilities::geometry::vector vectorB = lineB.getVector();
194.              bso::utilities::geometry::vector vectorC = lineC.getVector();
195.              bso::utilities::geometry::vector vectorD = lineD.getVector();
196.              if (vectorA.isParallel(vectorB))
197.              { score_nL-=1; }
198.              if (vectorB.isParallel(vectorC))
199.              { score_nL-=1; }
200.              if (vectorC.isParallel(vectorD))
201.              { score_nL-=1; }
202.              if (vectorD.isParallel(vectorA))
203.              { score_nL-=1; }
204.
205.              fitness = score_nx + score_ny + score_nz + score_nL;
206.          // ------------ end   fitness NON-Orthogonal --------------- //
207.
208.              return fitness;
```

```
209.          }
210.
211.          int convexity() {
212.              // CHECK CONVEXITY
213.              bso::utilities::geometry::vertex vx1 = {points8[0][0], points8[0][1], points8[0][2]};
214.              bso::utilities::geometry::vertex vx2 = {points8[2][0], points8[2][1], points8[2][2]};
215.              bso::utilities::geometry::vertex vx3 = {points8[4][0], points8[4][1], points8[4][2]};
216.              bso::utilities::geometry::vertex vx4 = {points8[6][0], points8[6][1], points8[6][2]};
217.              bso::utilities::geometry::line_segment diagonal1 = {vx1, vx4};
218.              bso::utilities::geometry::line_segment diagonal2 = {vx2, vx3};
219.              bso::utilities::geometry::line_segment diagonal3 = {vx1, vx3};
220.              bso::utilities::geometry::line_segment diagonal4 = {vx2, vx4};
221.              bso::utilities::geometry::vertex pIntersectionE;
222.              double mETol = 1e-3;
223.              int convex;
224.              if ( (diagonal1.intersectsWith(diagonal2, pIntersectionE, mETol)) || (diagonal3.inter-
    sectsWith(diagonal4, pIntersectionE, mETol)) )
225.              { convex = 1;     // CONVEX
226.              } else {
227.                convex = 0;     // NON-CONVEX
228.              }
229.              return convex;
230.          }
231.   };
232.
233.  // Initialise population of GA1: Quad-hexahedrons. Make N combinations of 8 points
234.  // INPUT    = N : Population Size
235.  //          = PointCloud : all vertices in Point cloud
236.  // OUTPUT   + population_GA1 : all Individuals GA1
237.  void initializePopulation_GA1(  int N, vector<vector<double>> PointCloud,
238.                                  vector<individual_GA1> &population_GA1)
239.  {
240.      cout << "\nInitialise Population GA1 (N = " << N << ")" << endl;
241.      population_GA1.reserve(N);
242.      for(int n = 0; n < N; n++)
243.      {
244.          population_GA1.push_back(individual_GA1());
245.          for(int i = 0; i < 8; i++)
246.          {
247.              vector<double> randomPoint = PointCloud[rand() % PointCloud.size()];
248.              population_GA1[n].points8.push_back(randomPoint);
249.          }
250.      }
251.  }
252.
253.  // Mutation (chromosome)
254.  // Only unique individuals_GA1. Duplicate individuals are removed and replaced by a new unique individ-
    ual.
255.  // INPUT    = population_GA1, N, PointCloud
256.  // OUTPUT   + unique individuals
257.  void mutation1_GA1( vector<individual_GA1> population_GA1, int N, vector<vector<double>> PointCloud,
258.                          vector<individual_GA1> &mutated1_population_GA1)
259.  {
260.      map< vector<vector<double>>, vector<vector<double>> > population_GA1_map;
261.      for (int n = 0; n < N; n++)
262.      {
263.          population_GA1_map[population_GA1[n].points8] = population_GA1[n].points8;
264.      }
265.      if (population_GA1_map.size() < N)
266.      {
267.          while (population_GA1_map.size() < N)
268.          {
269.              for (int f = 0; f < (N - population_GA1_map.size()); f++)
270.              {
271.                  individual_GA1 new_individual;
272.                  for(int p = 0; p < 8; p++)
273.                  {
274.                      vector<double> randomPoint = PointCloud[rand() % PointCloud.size()];
275.                      new_individual.points8.push_back(randomPoint);
276.                  }
277.                  population_GA1_map[new_individual.points8] = new_individual.points8;
278.                  new_individual.points8.clear();
279.              }
280.          }
281.      }
282.      typedef map< vector<vector<double>>, vector<vector<double>> > mapuType;
283.      int u = 0;
```

```
284.      for( mapuType::iterator it = population_GA1_map.begin(); it != population_GA1_map.end(); ++it )
285.      {
286.          mutated1_population_GA1.push_back(individual_GA1());
287.          mutated1_population_GA1[u].points8 = it->second ;
288.          u++;
289.      }
290.  }
291.
292.  // Crossovers
293.  // 2 parents create 4 new offsprings by crossover operator
294.  // Crossover X (1-point crossover)
295.  void crossoverX( vector<individual_GA1> population_GA1, int N,
296.                   vector<individual_GA1> &offspring)
297.  {
298.      offspring.reserve(N*2);
299.      for (int n = 0; n < (population_GA1.size()*2); n+=4)
300.      {
301.          offspring.push_back(individual_GA1());
302.          offspring.push_back(individual_GA1());
303.          offspring.push_back(individual_GA1());
304.          offspring.push_back(individual_GA1());
305.          int i = 0;
306.          for (int p = 0; p < 8; p+=4)
307.          {
308.              int k = i%2;
309.              int m = (i+1)%2;
310.              offspring[n+0].points8.push_back(population_GA1[(n/2)+k].points8[0+0]);        // AC
311.              offspring[n+0].points8.push_back(population_GA1[(n/2)+k].points8[0+1]);
312.              offspring[n+0].points8.push_back(population_GA1[(n/2)+k].points8[0+2]);
313.              offspring[n+0].points8.push_back(population_GA1[(n/2)+k].points8[0+3]);
314.
315.              offspring[n+1].points8.push_back(population_GA1[(n/2)+k].points8[4+0]);        // BD
316.              offspring[n+1].points8.push_back(population_GA1[(n/2)+k].points8[4+1]);
317.              offspring[n+1].points8.push_back(population_GA1[(n/2)+k].points8[4+2]);
318.              offspring[n+1].points8.push_back(population_GA1[(n/2)+k].points8[4+3]);
319.
320.              offspring[n+2].points8.push_back(population_GA1[(n/2)+k].points8[(k*4)+0]);    // AD
321.              offspring[n+2].points8.push_back(population_GA1[(n/2)+k].points8[(k*4)+1]);
322.              offspring[n+2].points8.push_back(population_GA1[(n/2)+k].points8[(k*4)+2]);
323.              offspring[n+2].points8.push_back(population_GA1[(n/2)+k].points8[(k*4)+3]);
324.
325.              offspring[n+3].points8.push_back(population_GA1[(n/2)+k].points8[(m*4)+0]);    // BC
326.              offspring[n+3].points8.push_back(population_GA1[(n/2)+k].points8[(m*4)+1]);
327.              offspring[n+3].points8.push_back(population_GA1[(n/2)+k].points8[(m*4)+2]);
328.              offspring[n+3].points8.push_back(population_GA1[(n/2)+k].points8[(m*4)+3]);
329.
330.              i++;
331.          }
332.      }
333.  }
334.  // Crossover Y (3-point crossover)
335.  void crossoverY( vector<individual_GA1> population_GA1, int N,
336.                   vector<individual_GA1> &offspring)
337.  {
338.      offspring.reserve(N*2);
339.      for (int n = 0; n < population_GA1.size()*2; n+=4)
340.      {
341.          offspring.push_back(individual_GA1());
342.          offspring.push_back(individual_GA1());
343.          offspring.push_back(individual_GA1());
344.          offspring.push_back(individual_GA1());
345.          for (int p = 0; p < 8; p+=4)
346.          {
347.              offspring[n+0].points8.push_back(population_GA1[(n/2)+0].points8[p+0]);        // AD
348.              offspring[n+0].points8.push_back(population_GA1[(n/2)+0].points8[p+1]);
349.              offspring[n+0].points8.push_back(population_GA1[(n/2)+1].points8[p+2]);
350.              offspring[n+0].points8.push_back(population_GA1[(n/2)+1].points8[p+3]);
351.
352.              offspring[n+1].points8.push_back(population_GA1[(n/2)+0].points8[p+0]);        // AC
353.              offspring[n+1].points8.push_back(population_GA1[(n/2)+0].points8[p+1]);
354.              offspring[n+1].points8.push_back(population_GA1[(n/2)+1].points8[p+0]);
355.              offspring[n+1].points8.push_back(population_GA1[(n/2)+1].points8[p+1]);
356.
357.              offspring[n+2].points8.push_back(population_GA1[(n/2)+1].points8[p+0]);        // BC
358.              offspring[n+2].points8.push_back(population_GA1[(n/2)+1].points8[p+1]);
359.              offspring[n+2].points8.push_back(population_GA1[(n/2)+0].points8[p+2]);
360.              offspring[n+2].points8.push_back(population_GA1[(n/2)+0].points8[p+3]);
```

```
361.
362.             offspring[n+3].points8.push_back(population_GA1[(n/2)+0].points8[p+2]);        // BD
363.             offspring[n+3].points8.push_back(population_GA1[(n/2)+0].points8[p+3]);
364.             offspring[n+3].points8.push_back(population_GA1[(n/2)+1].points8[p+2]);
365.             offspring[n+3].points8.push_back(population_GA1[(n/2)+1].points8[p+3]);
366.         }
367.     }
368. }
369. // Crossover Z (7-point crossover)
370. void crossoverZ( vector<individual_GA1> population_GA1, int N,
371.                  vector<individual_GA1> &offspring)
372. {
373.     offspring.reserve(N*2);
374.     for (int n = 0; n < population_GA1.size()*2; n+=4)
375.     {
376.         offspring.push_back(individual_GA1());
377.         offspring.push_back(individual_GA1());
378.         offspring.push_back(individual_GA1());
379.         offspring.push_back(individual_GA1());
380.         for (int p = 0; p < 8; p+=2)
381.         {
382.             offspring[n+0].points8.push_back(population_GA1[(n/2)+0].points8[p+0]);        // AD
383.             offspring[n+0].points8.push_back(population_GA1[(n/2)+1].points8[p+1]);
384.
385.             offspring[n+1].points8.push_back(population_GA1[(n/2)+0].points8[p+0]);        // AC
386.             offspring[n+1].points8.push_back(population_GA1[(n/2)+1].points8[p+0]);
387.
388.             offspring[n+2].points8.push_back(population_GA1[(n/2)+0].points8[p+1]);        // BC
389.             offspring[n+2].points8.push_back(population_GA1[(n/2)+1].points8[p+0]);
390.
391.             offspring[n+3].points8.push_back(population_GA1[(n/2)+0].points8[p+1]);        // BD
392.             offspring[n+3].points8.push_back(population_GA1[(n/2)+1].points8[p+1]);
393.         }
394.     }
395. }
396.
397. // 2 Parents and 4 offsprings are merged to intermediate population (with increased size)
398. // OUTPUT   + merged_population_GA1 : intermediate population
399. void merge( vector<individual_GA1> population_GA1, vector<individual_GA1> offspring, int N,
400.             vector<individual_GA1> &merged_population_GA1)
401. {
402.     merged_population_GA1.reserve(N*3);
403.     int o1 = 0; int o2 = 1;
404.     for (int i = 0; i < N*3; i+=3)
405.     {
406.         merged_population_GA1.push_back(individual_GA1());
407.         merged_population_GA1.push_back(individual_GA1());
408.         merged_population_GA1.push_back(individual_GA1());
409.         for (int p = 0; p < 8; p++)
410.         {
411.             merged_population_GA1[i].points8.push_back(population_GA1[i/3].points8[p]);
412.             merged_population_GA1[i+1].points8.push_back(offspring[o1].points8[p]);
413.             merged_population_GA1[i+2].points8.push_back(offspring[o2].points8[p]);
414.         }
415.         o1+=2; o2+=2;
416.     }
417. }
418.
419. // Mutation (gene)
420. // Duplicate vertices in an individual are replaced by new unique vertices randomly chosen from Point
    cloud.
421. void mutation2_GA1( vector<individual_GA1> merged_population_GA1, vector<vector<double>> PointCloud,
422.                     vector<individual_GA1> &mutated2_population_GA1)
423. {
424.     map<vector<double>, vector<double>> points8_map;
425.     for (int n = 0; n < merged_population_GA1.size(); n++)
426.     {
427.         for (int p = 0; p < 8; p++)
428.         {
429.             points8_map[merged_population_GA1[n].points8[p]] = merged_population_GA1[n].points8[p];
430.         }
431.         while (points8_map.size() < 8)
432.         {
433.             for (int f = 0; f < (8 - points8_map.size()); f++)
434.             {
435.                 int randomN = rand() % PointCloud.size();
436.                 points8_map[PointCloud[randomN]] = PointCloud[randomN];
```

```
437.             }
438.         }
439.         merged_population_GA1[n].points8.clear();
440.         mutated2_population_GA1.push_back(individual_GA1());
441.         typedef map< vector<double>, vector<double> > mapbType;
442.         for( mapbType::iterator it = points8_map.begin(); it != points8_map.end(); ++it )
443.         {
444.             mutated2_population_GA1[n].points8.push_back( it->second );
445.         }
446.         points8_map.clear();
447.     }
448. }
449.
450. // Selection of the fittest
451. //  2 best individuals are chosen from 2 parents and their 4 offsprings
452. void selection_GA1( vector<individual_GA1> mutated2_population_GA1,
453.                     vector<individual_GA1> &new_population_GA1)
454. {
455.     multimap<double, individual_GA1> mapE;
456.     for (int i = 0; i < mutated2_population_GA1.size(); i+=6)
457.     {
458.         for (int j = 0; j < 6; j++)
459.         {
460.             mapE.insert(mapE.end(), pair<double, individual_GA1>(mutated2_population_GA1[i+j].fit-
    ness_GA1(), mutated2_population_GA1[i+j]) );
461.         }
462.         new_population_GA1.push_back(individual_GA1());
463.         new_population_GA1.push_back(individual_GA1());
464.         for (int p = 0; p < 8; p++)
465.         {
466.             new_population_GA1[int(i/3)].points8.push_back((*prev(mapE.rbegin())).second.points8[p]);
467.             new_population_GA1[int((i/3)+1)].points8.push_back((*prev(mapE.end())).second.points8[p]);
468.         }
469.         mapE.clear();
470.     }
471.     double total_fitness = 0.0;
472.     for (int k = 0; k < new_population_GA1.size(); k++)
473.     {
474.         total_fitness = total_fitness + new_population_GA1[k].fitness_GA1();
475.     }
476.     double average_fitness = total_fitness/new_population_GA1.size();
477.     cout << "Average fitness:                                    " << average_fitness << endl;
478. }
479.
480. // If an individual gains the maximum fitness score, it as a perfect quad-hexahedron
481. // The perfect quad-hexahedrons are used for next GA2.
482. void get_perfect_Quad_Hexahedron(   vector<individual_GA1> population_GA1,
483.                                     vector<individual_GA1> &perfect_Quad_Hexahedron)
484. {
485.     map< vector<vector<double>>, vector<vector<double>> > population_GA1_map;
486.     for (int n = 0; n < population_GA1.size(); n++)
487.     {
488.         // ONLY population_GA1 with maximum fitness!
489.         if (population_GA1[n].fitness_GA1() >= 10)
490.         { population_GA1_map[population_GA1[n].points8] = population_GA1[n].points8; }
491.     }
492.     typedef map< vector<vector<double>>, vector<vector<double>> > mapuType;
493.     int u = 0;
494.     for( mapuType::iterator it = population_GA1_map.begin(); it != population_GA1_map.end(); ++it ) {
495.         perfect_Quad_Hexahedron.push_back(individual_GA1());
496.         perfect_Quad_Hexahedron[u].points8 = it->second ;
497.         u++; }
498.     cout << "size of perfect unique population_GA1:        " << perfect_Quad_Hexahedron.size() << endl;
499. }
500. void save_perfect_Quad_Hexahedron( vector<individual_GA1> perfect_Quad_Hexahedron,
501.                                    map< vector<vector<double>>, vector<vector<double>> >
    &saved_Quad_Hexahedrons_map,
502.                                    vector<individual_GA1> &saved_Quad_Hexahedrons)
503. {
504.     for (int n = 0; n < perfect_Quad_Hexahedron.size(); n++)
505.     {
506.         saved_Quad_Hexahedrons_map[perfect_Quad_Hexahedron[n].points8] = perfect_Quad_Hexahe-
    dron[n].points8;
507.     }
508.     cout << "size of saved perfect unique objects:  " << saved_Quad_Hexahedrons_map.size() << endl;
509.     typedef map< vector<vector<double>>, vector<vector<double>> > mapuType;
510.     int u = 0;
```

```
511.      for( mapuType::iterator it = saved_Quad_Hexahedrons_map.begin(); it != saved_Quad_Hexahe-
    drons_map.end(); ++it )
512.      {
513.          saved_Quad_Hexahedrons.push_back(individual_GA1());
514.          saved_Quad_Hexahedrons[u].points8 = it->second ;
515.          u++;
516.      }
517.  }
518.
519.  // The quad-hexahedrons are checked if they are inside the BSD spaces.
520.  // Thereafter, the quad-hexahedrons are desrcibed by the BSO-toolbox class "bso:utilities:geome-
    try:quad_hexahdron"
521.  void isInsideBSD(    vector<individual_GA1> saved_Quad_Hexahedrons,
522.                       vector<bso::utilities::geometry::quad_hexahedron> hexaSpaces,
523.                       vector<vector<individual_GA1>> &Quad_Hexahedrons_InsidePerSpace,
524.                       vector<vector<bso::utilities::geometry::quad_hexahedron>> &Quad_Hexahedrons_per-
    Space,
525.                       vector<bso::utilities::geometry::quad_hexahedron> &Quad_Hexahedrons)
526.  {
527.      for (int ss = 0; ss < hexaSpaces.size(); ss++)
528.      {
529.          Quad_Hexahedrons_InsidePerSpace.push_back(vector<individual_GA1>());
530.          Quad_Hexahedrons_perSpace.push_back(vector<bso::utilities::geometry::quad_hexahedron>());
531.      }
532.      int nroftotalelements = 0;
533.      int nrofelements = 0;
534.      for (int i = 0; i < saved_Quad_Hexahedrons.size(); i++)
535.      {
536.          if (saved_Quad_Hexahedrons[i].convexity() != 0)
537.          {
538.              nroftotalelements++;
539.              for (int s = 0; s < hexaSpaces.size(); s++)
540.              {
541.                  int is = 0;
542.                  for (int p = 0; p < 8; p++)
543.                  {
544.                      bso::utilities::geometry::vertex point = { saved_Quad_Hexahedrons[i].points8[p][0],
    saved_Quad_Hexahedrons[i].points8[p][1], saved_Quad_Hexahedrons[i].points8[p][2] };
545.                      if (hexaSpaces[s].isInsideOrOn(point)) { is++; }
546.                  }
547.                  if (is == 8)
548.                  {
549.                      nrofelements++;
550.                      Quad_Hexahedrons_InsidePerSpace[s].push_back(individual_GA1());
551.                      int sizepu = Quad_Hexahedrons_InsidePerSpace[s].size();
552.                      Quad_Hexahedrons_InsidePerSpace[s][sizepu-1].points8 = saved_Quad_Hexahe-
    drons[i].points8;
553.                  }
554.              }
555.          }
556.      }
557.      // Quad-hexahedrons are now described by the BSO-toolbox class. These are used for next GA2.
558.      for (int s = 0; s < Quad_Hexahedrons_InsidePerSpace.size(); s++)
559.      {
560.          for (int c = 0; c < Quad_Hexahedrons_InsidePerSpace[s].size(); c++)
561.          {
562.              Quad_Hexahedrons_perSpace[s].push_back(
563.                  bso::utilities::geometry::quad_hexahedron({ {double(Quad_Hexahedrons_InsidePer-
    Space[s][c].points8[0][0]), double(Quad_Hexahedrons_InsidePerSpace[s][c].points8[0][1]), double(Quad_Hex-
    ahedrons_InsidePerSpace[s][c].points8[0][2])},
564.                                                  {double(Quad_Hexahedrons_InsidePer-
    Space[s][c].points8[1][0]), double(Quad_Hexahedrons_InsidePerSpace[s][c].points8[1][1]), double(Quad_Hex-
    ahedrons_InsidePerSpace[s][c].points8[1][2])},
565.                                                  {double(Quad_Hexahedrons_InsidePer-
    Space[s][c].points8[2][0]), double(Quad_Hexahedrons_InsidePerSpace[s][c].points8[2][1]), double(Quad_Hex-
    ahedrons_InsidePerSpace[s][c].points8[2][2])},
566.                                                  {double(Quad_Hexahedrons_InsidePer-
    Space[s][c].points8[3][0]), double(Quad_Hexahedrons_InsidePerSpace[s][c].points8[3][1]), double(Quad_Hex-
    ahedrons_InsidePerSpace[s][c].points8[3][2])},
567.                                                  {double(Quad_Hexahedrons_InsidePer-
    Space[s][c].points8[4][0]), double(Quad_Hexahedrons_InsidePerSpace[s][c].points8[4][1]), double(Quad_Hex-
    ahedrons_InsidePerSpace[s][c].points8[4][2])},
568.                                                  {double(Quad_Hexahedrons_InsidePer-
    Space[s][c].points8[5][0]), double(Quad_Hexahedrons_InsidePerSpace[s][c].points8[5][1]), double(Quad_Hex-
    ahedrons_InsidePerSpace[s][c].points8[5][2])},
```

```
569.                                                     {double(Quad_Hexahedrons_InsidePer-
    Space[s][c].points8[6][0]), double(Quad_Hexahedrons_InsidePerSpace[s][c].points8[6][1]), double(Quad_Hex-
    ahedrons_InsidePerSpace[s][c].points8[6][2])},
570.                                                     {double(Quad_Hexahedrons_InsidePer-
    Space[s][c].points8[7][0]), double(Quad_Hexahedrons_InsidePerSpace[s][c].points8[7][1]), double(Quad_Hex-
    ahedrons_InsidePerSpace[s][c].points8[7][2])}}) );
571.             }
572.         }
573.         for(const auto &v: Quad_Hexahedrons_perSpace)
574.         {
575.             Quad_Hexahedrons.insert(Quad_Hexahedrons.end(), v.begin(), v.end());
576.         }
577.     }
578.     // Finally, a list of Quad-Hexahedrons is created and used for next GA2
579.
580.     // ------------------------------- //
581.     // --- GA2 : Conformal geometry --- //
582.     // ------------------------------- //
583.
584.     class individual_GA2 {
585.         public:
586.
587.             // Binary notation of individual GA2
588.             vector<int> quad_cluster;
589.
590.             // Each '1' in quad_cluster represents a Quad-Hexahedron
591.             // CFquads = Conformal quad-hexahedrons
592.             vector<bso::utilities::geometry::quad_hexahedron> CFquads(vector<bso::utilities::geome-
    try::quad_hexahedron> Quad_Hexahedrons) {
593.                 vector<bso::utilities::geometry::quad_hexahedron> CFquads;
594.                 for (int i = 0; i < quad_cluster.size(); i++)
595.                 {
596.                     if (quad_cluster[i] == 1)
597.                     {
598.                         CFquads.push_back(Quad_Hexahedrons[i]);
599.                     }
600.                 }
601.                 return CFquads;
602.             }
603.
604.             // fitness calculation of GA2
605.             vector<double> fitness_GA2(vector<bso::utilities::geometry::quad_hexahedron> Quad_Hexahedrons)
606.             {
607.                 vector<bso::utilities::geometry::quad_hexahedron> CF_quads;
608.                 double volume_CF = 0;
609.                 for (int i = 0; i < quad_cluster.size(); i++)
610.                 {
611.                     if (quad_cluster[i] == 1)
612.                     {
613.                         CF_quads.push_back(Quad_Hexahedrons[i]);
614.                         volume_CF = volume_CF + Quad_Hexahedrons[i].getVolume();
615.                     }
616.                 }
617.                 // nCr: combinations claculator
618.                 double nCr;
619.                 double n = CF_quads.size();
620.                 double r = 2;
621.                 double nr = n - r;
622.                 long double n_fact = 1.0;
623.                 for(int i = 1; i <= n; ++i) { n_fact *= i; }
624.                 long double r_fact = 1.0;
625.                 for(int i = 1; i <= r; ++i) { r_fact *= i; }
626.                 long double nr_fact = 1.0;
627.                 for(int i = 1; i <= nr; ++i) { nr_fact *= i; }
628.                 nCr = n_fact / ( r_fact * nr_fact ) ;
629.
630.                 vector<double> fitness_CF;
631.                 double score_nIntersect;
632.                 double nIntersections = 0.0;
633.                 double isIntersecting = 0.0;
634.                 double score_nVolume = 0.0;
635.                 double score_nLink;
636.                 double score_nHyperlink;
637.                 double isHyperlinked = 0.0;
638.
639.                 // initialize quad combinations
640.                 set<vector<int>> quadCombinations;
```

```
641.             int k = 2;
642.             int nCombi = 0;
643.             for (int j = 1; j < CF_quads.size(); j++) { nCombi += j; }
644.             for (int k = 0; k < CF_quads.size(); k++) {
645.                 for (int m = k+1; m < CF_quads.size(); m++) {
646.                     quadCombinations.insert({k , m});
647.                 }
648.             }
649.
650.          // VOLUME fitness
651.             int nSpaces; double volume_MS2; vector<bso::utilities::geometry::quad_hexahedron> hexaSpaces;
652.             vector<int> x_coordinates; vector<int> y_coordinates; vector<int> z_coordinates;
653.             collectCoordinates("cornerverticesBSD_GA.txt", nSpaces, volume_MS2, hexaSpaces, x_coordinates, y_coordinates, z_coordinates);
654.             double total_volume_Q = 0;
655.             for (int i = 0; i < Quad_Hexahedrons.size(); i++) { total_volume_Q = total_volume_Q + Quad_Hexahedrons[i].getVolume(); }
656.
657.             // Compute fitness score of Intersections
658.             score_nVolume = abs(volume_CF - volume_MS2) / ( total_volume_Q - volume_MS2 );
659.
660.             fitness_CF.push_back(score_nVolume);
661.
662.
663.          // INTERSECTION fitness
664.             bso::utilities::geometry::vertex pIntersectioncf;
665.             double mTol = 1e-3;
666.             int intersectionsLL = 0;
667.             int intersectionsLR = 0;
668.             int intersectionsRR = 0;
669.             double link = 0.0;              // perfect link between 2 quads, no intersections/overlap
670.
671.             for (auto const &c : quadCombinations)  // check intersections for each combination
672.             {
673.                 // INTERSECTION (Line - Line)
674.                 intersectionsLL = 0;
675.                 for (unsigned int i = 0; i < CF_quads[c[0]].getLines().size(); ++i)
676.                 {
677.                     for (unsigned int j = 0; j < CF_quads[c[1]].getLines().size(); ++j)
678.                     {
679.                         if (CF_quads[c[0]].getLines()[i].intersectsWith(CF_quads[c[1]].getLines()[j], pIntersectioncf, mTol))
680.                         {
681.                             intersectionsLL++;
682.                             isIntersecting++;
683.                             goto newquadcombi;
684.                         }
685.                     }
686.                 }
687.                 // INTERSECTION (Line - Polygon)
688.                 intersectionsLR = 0;
689.                 for (unsigned int i = 0; i < CF_quads[c[0]].getPolygons().size(); ++i)
690.                 {
691.                     for (unsigned int j = 0; j < CF_quads[c[1]].getLines().size(); ++j)
692.                     {
693.                         if (CF_quads[c[0]].getPolygons()[i]->intersectsWith(CF_quads[c[1]].getLines()[j], pIntersectioncf, mTol))
694.                         {
695.                             intersectionsLR++;
696.                             isIntersecting++;
697.                             goto newquadcombi;
698.                         }
699.                     }
700.                 }
701.                 // INTERSECTION (Polygon - Line)
702.                 for (unsigned int i = 0; i < CF_quads[c[1]].getPolygons().size(); ++i)
703.                 {
704.                     for (unsigned int j = 0; j < CF_quads[c[0]].getLines().size(); ++j)
705.                     {
706.                         if (CF_quads[c[1]].getPolygons()[i]->intersectsWith(CF_quads[c[0]].getLines()[j], pIntersectioncf, mTol))
707.                         {
708.                             intersectionsLR++;
709.                             isIntersecting++;
710.                             goto newquadcombi;
711.                         }
```

```
712.                     }
713.                 }
714.
715.                 newquadcombi:
716.                 // INTERSECTION (Overlap)
717.                 if ((intersectionsLL == 0) && (intersectionsLR == 0))
718.                 {
719.                     intersectionsRR = 0;
720.                     for (unsigned int j = 0; j < CF_quads[c[1]].getVertices().size(); ++j)
721.                     {
722.                         if (CF_quads[c[0]].isInsideOrOn(CF_quads[c[1]].getVertices()[j], mTol))
723.                         {
724.                             intersectionsRR++;
725.                         }
726.                     }
727.                     if (intersectionsRR > 4)
728.                     {
729.                         isIntersecting++;
730.                         continue;
731.                     }
732.                     intersectionsRR = 0;
733.                     for (unsigned int j = 0; j < CF_quads[c[0]].getVertices().size(); ++j)
734.                     {
735.                         if (CF_quads[c[1]].isInsideOrOn(CF_quads[c[0]].getVertices()[j], mTol))
736.                         {
737.                             intersectionsRR++;
738.                         }
739.                     }
740.                     if (intersectionsRR > 4)
741.                     {
742.                         isIntersecting++;
743.                         continue;
744.                     }
745.
746.                     if (intersectionsRR == 4)
747.                     {
748.                         link++;
749.                     }
750.                 }
751.             }
752.
753.             // Compute fitness score of Intersections
754.             double double_link = double(link);
755.             if ( (isIntersecting > 0) || (double_link > 0) )
756.             {
757.                 score_nIntersect = isIntersecting / (isIntersecting + double_link);
758.             } else {
759.                 score_nIntersect = 0;
760.             }
761.
762.             fitness_CF.push_back(score_nIntersect);
763.
764.
765.             return fitness_CF;
766.         }
767.  };
768.
769.  // Initialise population of GA2: Conformal model. Make M combinations of quad-hexahedrons
770.  // INPUT    = M : Population Size
771.  //          = Quad-Hexahedrons : found in GA1
772.  //          = probability(min,max) of ones in individual (Chromosome) binary notation. Default = 1%,
    100%
773.  // OUTPUT   + population_GA2 : all Individuals GA2
774.  void initializePopulation_GA2(  int M, int probability_1_min, int probability_1_max,  vector<bso::util-
    ities::geometry::quad_hexahedron> Quad_Hexahedrons,
775.                                  vector<individual_GA2> &population_GA2)
776.  {
777.      population_GA2.reserve(M);
778.      int nGenes = Quad_Hexahedrons.size();
779.      int randbinary;
780.      int randprob;
781.      double probability_i;
782.      for (int i = 0; i < M; i++)
783.      {
784.          probability_i = double(probability_1_min) + i*(double((probability_1_max-probabil-
    ity_1_min)/M));
785.          population_GA2.push_back(individual_GA2());
```

```
786.          for (int j = 0; j < nGenes; j++)
787.          {
788.              randprob = rand() % 100;
789.              if (randprob < probability_i) {
790.                  randbinary = 1;
791.              } else {
792.                  randbinary = 0;
793.              }
794.              population_GA2[i].quad_cluster.push_back( randbinary );
795.          }
796.      }
797.  }
798.
799.  // Selection of the fittest individuals
800.  // INPUT    = M, population, Quad-Hexahedrons
801.  // OUTPUT   + parent_GA2 : fittest half (50%) is chosen as parents
802.  //          + fitness_vector : fitness scores (fV & fI) are saved in a 2D vector
803.  void selection_GA2(int M, vector<individual_GA2> population_GA2, vector<bso::utilities::geome-
    try::quad_hexahedron> Quad_Hexahedrons,
804.                     vector<individual_GA2> &parent_GA2, vector<vector<double>> &fitness_vector)
805.  {
806.      vector<int> dominate_count(M);
807.      vector<int> dominatedBy_count(M);
808.      for (int i = 0; i < M; i++)
809.      {
810.          fitness_vector.push_back(population_GA2[i].fitness_GA2(Quad_Hexahedrons));
811.      }
812.      for (int i = 0; i < M; i++)
813.      {
814.          for (int j = 0; j < M; j++)
815.          {
816.              if (i != j)
817.              {
818.                  // Dominated by other individuals based on 2 objectives
819.                  if ((fitness_vector[i][0] <= fitness_vector[j][0]) && (fitness_vector[i][1] <= fit-
    ness_vector[j][1]) )
820.                  {
821.                      if ((fitness_vector[i][0] < fitness_vector[j][0])  || (fitness_vector[i][1] < fit-
    ness_vector[j][1]) )
822.                      {
823.                          ++dominatedBy_count[j];
824.                      }
825.                  }
826.              }
827.          }
828.      }
829.      multimap<int, individual_GA2> mapCF;
830.      multimap< int, vector<double> > mapFV;
831.      vector<individual_GA2> sortCF;
832.      sortCF.reserve(M);
833.      parent_GA2.reserve(M/2);
834.      for (int i = 0; i < M; i++)
835.      {
836.          mapCF.insert(mapCF.end(), pair<int, individual_GA2>(dominatedBy_count[i], population_GA2[i]) );
837.          mapFV.insert(mapFV.end(), pair<int, vector<double> >(dominatedBy_count[i], fitness_vector[i])
    );
838.      }
839.      typedef multimap<int, individual_GA2> mapcfType;
840.      int i = 0;
841.      for( mapcfType::iterator it = mapCF.begin(); it != mapCF.end(); ++it )
842.      {
843.          sortCF.push_back(individual_GA2());
844.          sortCF[i] = it->second;
845.          i++;
846.      }
847.      typedef multimap<int, vector<double>> mapfvType;
848.      int k = 0;
849.      for( mapfvType::iterator it = mapFV.begin(); it != mapFV.end(); ++it )
850.      {
851.          cout << "(" << it->first << ") ,  n quads =  " << sortCF[k].CFquads(Quad_Hexahedrons).size() <<
    "\t [ " << it->second[0] << ", " << it->second[1] << " ]   " << endl;  // << ", " << it->second[2] << " ]
    " << endl;
852.          k++;
853.      }
854.      int j = 0;
855.      for (int i = 0; i < M/2; i++)
856.      {
```

```
857.          parent_GA2.push_back(individual_GA2());
858.          parent_GA2[j] = sortCF[i];
859.          j++;
860.      }
861.  }
862.
863.  // Crossover GA2 (Uniform crossover)
864.  // INPUT    = parents GA2
865.  // OUTPUT   + offspring GA2 : 2 parents create 2 new offspring by uniform crossover
866.  void crossover_GA2_uniform( int M, vector<individual_GA2> parent_GA2, vector<bso::utilities::geome-
    try::quad_hexahedron> Quad_Hexahedrons,
867.                              vector<individual_GA2> &offspring_GA2)
868.  {
869.      int sameBit;
870.      offspring_GA2.reserve(M/2);
871.      for (int i = 0; i < M/2; i+=2)
872.      {
873.          sameBit = 0;
874.          offspring_GA2.push_back(individual_GA2());
875.          offspring_GA2.push_back(individual_GA2());
876.          for (int j = 0; j < Quad_Hexahedrons.size(); j++)
877.          {
878.              int rand_j = rand() % 2;
879.              int rand_k = (rand_j+1)%2;
880.              offspring_GA2[i+0].quad_cluster.push_back(parent_GA2[i+rand_j].quad_cluster[j]);
881.              offspring_GA2[i+1].quad_cluster.push_back(parent_GA2[i+rand_k].quad_cluster[j]);
882.              if (parent_GA2[i+rand_j].quad_cluster[j] == parent_GA2[i+rand_k].quad_cluster[j])
883.              { sameBit++; }
884.          }
885.          if (sameBit == Quad_Hexahedrons.size()) // if parents are exactly the same, the parents are
    shuffled
886.          {
887.              random_shuffle(offspring_GA2[i+0].quad_cluster.begin(), offspring_GA2[i+0].quad_clus-
    ter.end()) ;
888.              random_shuffle(offspring_GA2[i+1].quad_cluster.begin(), offspring_GA2[i+1].quad_clus-
    ter.end()) ;
889.          }
890.      }
891.  }
892.
893.  // Merge parents and offspring
894.  void merge_GA2(int M, vector<individual_GA2> parent_GA2, vector<individual_GA2> offspring_GA2,
895.              vector<individual_GA2> &merged_population_GA2)
896.  {
897.      merged_population_GA2.reserve(M);
898.      for (int m = 0; m < M; m+=2)
899.      {
900.          merged_population_GA2.push_back(individual_GA2());
901.          merged_population_GA2.push_back(individual_GA2());
902.          merged_population_GA2[m] = parent_GA2[m/2];
903.          merged_population_GA2[m+1] = offspring_GA2[m/2];
904.      }
905.  }
906.
907.  // Mutation GA2
908.  // INPUT    = mutation_rate_population : if 0.50, then 50% of population size (M) is mutated
909.  //          = mutation_rate_individual : if 1.00, one binary in the quad-cluster is flipped. 0 becomes
    1 or vice versa
910.  // OUTPUT   + Mutated population GA2
911.  void mutation_GA2(  int M, double mutation_rate_population, double mutation_rate_individual, vector<in-
    dividual_GA2> merged_population_GA2,
912.                      vector<individual_GA2> &mutated_population_GA2)
913.  {
914.      cout << mutation_rate_population*merged_population_GA2.size() << endl;
915.      for (int i = 0; i < (mutation_rate_population*merged_population_GA2.size()); i++)
916.      {
917.          for (int j = 0; j < mutation_rate_individual; j++ )
918.          {
919.              int randi = rand() % merged_population_GA2[i].quad_cluster.size();
920.              if (merged_population_GA2[i].quad_cluster[randi] == 0) {
921.                  merged_population_GA2[i].quad_cluster[randi] = 1;
922.              } else if (merged_population_GA2[i].quad_cluster[randi] == 1) {
923.                  merged_population_GA2[i].quad_cluster[randi] = 0;
924.              }
925.          }
926.      }
927.      mutated_population_GA2 = merged_population_GA2;
```

```
928.  }
929.
930.
931.  int main(int argc, char *argv[])
932.  {
933.   // Pre-Processing BSD Data (collect Coordinates & generate Point Cloud)
934.      int nSpaces; double total_volume_MS;
935.      vector<bso::utilities::geometry::quad_hexahedron> hexaSpaces;
936.      vector<int> x_coordinates; vector<int> y_coordinates; vector<int> z_coordinates;
937.      collectCoordinates("cornerverticesBSD_GA.txt", nSpaces, total_volume_MS, hexaSpaces, x_coordinates,
    y_coordinates, z_coordinates);
938.
939.      vector<vector<double>> PointCloud;
940.      generatePointCloud(nSpaces, hexaSpaces, x_coordinates, y_coordinates, z_coordinates, PointCloud);
941.
942.
943.   // GA1 : Quad-Hexahedrons
944.      // Input Parameters
945.      int N = 128;                                  // Population size
946.      int g = 0;                                    // generation counter
947.      vector<int> cho = {0,0,0};                    // crossover indicator
948.      int n = 100;                                  // after n generations where no new unique Quad-Hexahe-
    dron is found, GA1 stops
949.      vector<int> vec_n;
950.      int x = 1;
951.      int uniqueQHs = 2;
952.
953.      // Initialize Population GA1
954.      vector<individual_GA1> population_GA1;
955.      initializePopulation_GA1( N, PointCloud, population_GA1);
956.
957.      vector<individual_GA1> perfect_Quad_Hexahedron;
958.      vector<individual_GA1> saved_Quad_Hexahedrons;
959.      map< vector<vector<double>>, vector<vector<double>> > saved_Quad_Hexahedrons_map;
960.
961.
962.      // Start evolution GA1
963.      while ( uniqueQHs > 1 )
964.      {
965.          cout << "\ngeneration (" << g << ") " << endl;
966.          perfect_Quad_Hexahedron.clear();
967.          saved_Quad_Hexahedrons.clear();
968.
969.          // Mutation 1
970.          vector<individual_GA1> mutated1_population_GA1;
971.          mutation1_GA1(population_GA1, N, PointCloud, mutated1_population_GA1);
972.          population_GA1 = mutated1_population_GA1;
973.          random_shuffle(population_GA1.begin(), population_GA1.end());
974.
975.          // Crossovers X, Y, Z: alternately applied
976.          vector<individual_GA1> offspring;
977.          int choI = min_element(cho.begin(), cho.end()) - cho.begin();
978.          if      (choI == 0) { crossoverX(population_GA1, N, offspring); cho[0]++; cout << "crossoverX"
    << endl; }
979.          else if (choI == 1) { crossoverY(population_GA1, N, offspring); cho[1]++; cout << "crossoverY"
    << endl; }
980.          else if (choI == 2) { crossoverZ(population_GA1, N, offspring); cho[2]++; cout << "crossoverZ"
    << endl; }
981.
982.          // Merge
983.          vector<individual_GA1> merged_population_GA1;
984.          merge(population_GA1, offspring, N, merged_population_GA1);
985.
986.          // Mutation 2
987.          vector<individual_GA1> mutated2_population_GA1;
988.          mutation2_GA1(merged_population_GA1, PointCloud, mutated2_population_GA1);
989.
990.          // Selection
991.          vector<individual_GA1> new_population_GA1;
992.          selection_GA1(mutated2_population_GA1, new_population_GA1);
993.          population_GA1 = new_population_GA1;
994.
995.          offspring.clear();
996.          merged_population_GA1.clear();
997.          mutated2_population_GA1.clear();
998.          new_population_GA1.clear();
999.
```

```
1000.         // Save individuals if they are perfect Quad-hexahedrons
1001.         get_perfect_Quad_Hexahedron(population_GA1, perfect_Quad_Hexahedron);
1002.         save_perfect_Quad_Hexahedron(perfect_Quad_Hexahedron, saved_Quad_Hexahedrons_map,
    saved_Quad_Hexahedrons);
1003.
1004.         // Check if new quad-hexahedron is found
1005.         vec_n.push_back(saved_Quad_Hexahedrons_map.size());
1006.         if (g == x*n) { uniqueQHs = unique(vec_n.begin(), vec_n.end()) - vec_n.begin();  x++;
    vec_n.clear(); }
1007.
1008.         g++;
1009.     }
1010.     // Check if found quad-hexahedrons are inside one of the BSD spaces
1011.     vector<bso::utilities::geometry::quad_hexahedron> Quad_Hexahedrons;
1012.     vector<vector<individual_GA1>> Quad_Hexahedrons_InsidePerSpace;
1013.     vector<vector<bso::utilities::geometry::quad_hexahedron>> Quad_Hexahedrons_perSpace;
1014.     isInsideBSD(saved_Quad_Hexahedrons, hexaSpaces, Quad_Hexahedrons_InsidePerSpace, Quad_Hexahe-
    drons_perSpace, Quad_Hexahedrons);
1015.
1016.     // Print relevant information about GA1
1017.     cout << "\n\n --- GA1: Quad-Hexahedrons  --- " << endl;
1018.     cout << "Population size N:        " << N << endl;
1019.     cout << "Total generations:        " << g << endl;
1020.     cout << "Quad-Hexahedrons found:   " << Quad_Hexahedrons.size() << endl;
1021.     cout << "Volume BSD:               " << total_volume_MS << endl;
1022.     cout << "\n";
1023.     cout << "\n \n" ;
1024.     for ( int i = 0; i < Quad_Hexahedrons.size(); i++)
1025.     {
1026.         cout << "quad: (" << i << ") " << endl;
1027.         for (int p = 0; p < 8; p++)
1028.         {
1029.             cout << Quad_Hexahedrons[i].getVertices()[p][0] << ", " << Quad_Hexahedrons[i].getVerti-
    ces()[p][1] << ", " << Quad_Hexahedrons[i].getVertices()[p][2] << endl;
1030.         }
1031.     }
1032.
1033.
1034.  // GA2 : Conformal model
1035.     // Input Parameters:
1036.     int M = 8;                                  // Population size
1037.     int probability_1_min = 10;                 // minimal probability of ones in individual GA2
1038.     int probability_1_max = 100;                // maximal probability of ones in individual GA2
1039.     double mutation_rate_population = 0.5;      // mutation rate of population (nr. of individuals to
    mutate)
1040.     double mutation_rate_individual = 1;        // mutate rate of individuals  (nr. of bits to flip)
1041.     int g2 = 0;
1042.     bool CFmodelIsNotFound = true;
1043.
1044.     // Initialize Population GA2
1045.     vector<individual_GA2> population_GA2;
1046.     initializePopulation_GA2( M, probability_1_min, probability_1_max , Quad_Hexahedrons, popula-
    tion_GA2 );
1047.
1048.     ofstream fv("fitness_vector.txt");
1049.     vector<individual_GA2> perfect_CFmodel;
1050.
1051.     // Start evolution (GA2)
1052.     while (CFmodelIsNotFound)
1053.     {
1054.         cout << "! --- generation " << g2 << endl;
1055.
1056.         // Selection
1057.         random_shuffle(population_GA2.begin(), population_GA2.end());
1058.         vector<individual_GA2> parent_GA2;
1059.         vector<vector<double>> fitness_vector;
1060.         selection_GA2(M, population_GA2, Quad_Hexahedrons, parent_GA2, fitness_vector);
1061.         for (int i = 0; i < M; i++)
1062.         {  fv << fitness_vector[i][0] << ", " << fitness_vector[i][1] << endl; }
1063.         for (int i = 0; i < parent_GA2.size(); i++)
1064.         {
1065.             cout << "{ " ;
1066.             for (int j = 0; j < Quad_Hexahedrons.size(); j++)
1067.             {
1068.                 cout << parent_GA2[i].quad_cluster[j] << "," ;
1069.             }
1070.             cout << " }" << endl;
```

```
        }

        // Check if perfect conformal model is found  (fitness vector = [0,0])
        for (int i = 0; i < fitness_vector.size(); i++)
        {
            if ( (fitness_vector[i][0] == 0 ) && ( fitness_vector[i][1] == 0 )) {    //&& (par-
    ent_GA2.back().fitness(Quad_Hexahedrons)[2] <= 0.5  ) ) {
                perfect_CFmodel.push_back(individual_GA2());
                perfect_CFmodel[0] = population_GA2[i];
                CFmodelIsNotFound = false;
            }
        }

        // Crossover (Uniform)
        random_shuffle(parent_GA2.begin(), parent_GA2.end());
        vector<individual_GA2> offspring_GA2;
        crossover_GA2_uniform(M, parent_GA2, Quad_Hexahedrons, offspring_GA2);

        // Merge
        vector<individual_GA2> merged_population_GA2;
        merge_GA2(M, parent_GA2, offspring_GA2, merged_population_GA2);

        // Mutation
        random_shuffle(merged_population_GA2.begin(), merged_population_GA2.end());
        vector<individual_GA2> mutated_population_GA2;
        mutation_GA2(M, mutation_rate_population, mutation_rate_individual,  merged_population_GA2, mu-
    tated_population_GA2);

        population_GA2.clear();
        population_GA2 = mutated_population_GA2;
        parent_GA2.clear();
        offspring_GA2.clear();
        merged_population_GA2.clear();
        mutated_population_GA2.clear();

        g2++;
        cout << endl << endl;
    }

    // Save Conformal model in "cf.txt" file, so it can be used and visualised
    ofstream resultCF("cf.txt");
    for (int q = 0; q < perfect_CFmodel[0].CFquads(Quad_Hexahedrons).size(); q++)
    {
        for (int p = 0; p < 8; p++) {
            resultCF << perfect_CFmodel[0].CFquads(Quad_Hexahedrons)[q].getVertices()[p][0] << "\n" <<
    perfect_CFmodel[0].CFquads(Quad_Hexahedrons)[q].getVertices()[p][1] << "\n" << per-
    fect_CFmodel[0].CFquads(Quad_Hexahedrons)[q].getVertices()[p][2] << endl;
        }
    }


    return 0;
}

```

“visualisation_GA.cpp”

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <cmath>

using namespace std;

// Visualisation of BSD and CF model obtained by the Genetic Algorithm (GA)


// Create multidimensional vector with corner-vertices of BSD or conformal model
vector<vector<vector<float>>> BSD()
{
    // Load predicted data
    vector<string> predictedCoordinatesString;
```

```
17.     string coordinate;
18.     ifstream predictC("cf.txt");                    // "bsd.txt" or "cf.txt" : corner-vertices of BSD or
    CF model
19.     while ( !predictC.eof() )
20.     {
21.         while ( getline(predictC,coordinate) )
22.         {
23.             predictedCoordinatesString.push_back(coordinate) ;
24.         }
25.     }
26.     // convert string to Int vector
27.     vector<float> predictedCoordinatesFl;
28.     for (int i=0; i<predictedCoordinatesString.size(); i++)
29.     {
30.         float num = atoi(predictedCoordinatesString.at(i).c_str());
31.         predictedCoordinatesFl.push_back(num);
32.     }
33.
34.     // Create multidimensional vector, each individual quad-hexahedron is listed in a vector
35.     vector<vector<vector<float>>> ncoordinates;
36.     int p = 0;
37.     for (int c = 0; c < predictedCoordinatesFl.size()/24; c++)              // number of quad-hexahe-
    drons
38.     {
39.         vector<vector<float>> cuboidVec;
40.         for (int j = 0; j < 24; j+=3)                                       // number of numerical val-
    ues per quad-hexahedron
41.         {
42.             vector<float> pointVec = {predictedCoordinatesFl[p+j+0], predictedCoordinatesFl[p+j+1], pre-
    dictedCoordinatesFl[p+j+2]};
43.             cuboidVec.push_back(pointVec);
44.         }
45.         ncoordinates.push_back(cuboidVec);
46.         p+=24;
47.     }
48.
49.     return ncoordinates;            // return corner-vertices of BSD or CF to use in VISUALISATION
50. }
51.
52. vector<float> PointCloud()
53. {
54.     vector<string> predictedCoordinatesString;
55.     string coordinate;
56.     ifstream predictC("pnts.txt");                  // "pnts.txt" : vertices in Point cloud
57.     while ( !predictC.eof() )
58.     {
59.         while ( getline(predictC,coordinate) )
60.         {
61.             predictedCoordinatesString.push_back(coordinate) ;
62.         }
63.     }
64.     // convert string to Int vector
65.     vector<float> predictedCoordinatesFl;
66.     for (int i=0; i<predictedCoordinatesString.size(); i++)
67.     {
68.         float num = atoi(predictedCoordinatesString.at(i).c_str());
69.         predictedCoordinatesFl.push_back(num);
70.     }
71.     // Create multidimensional vector, each individual cuboid is listed in a vector
72.     vector<vector<vector<float>>> ncoordinates;
73.     int p = 0;
74.     for (int c = 0; c < predictedCoordinatesFl.size()/24; c++)              // number of cuboids
75.     {
76.         vector<vector<float>> cuboidVec;
77.         for (int j = 0; j < 24; j+=3)                                       // number of numerical val-
    ues per cuboid
78.         {
79.             vector<float> pointVec = {predictedCoordinatesFl[p+j+0], predictedCoordinatesFl[p+j+1], pre-
    dictedCoordinatesFl[p+j+2]};
80.             cuboidVec.push_back(pointVec);
81.         }
82.         ncoordinates.push_back(cuboidVec);
83.         p+=24;
84.     }
85.     return predictedCoordinatesFl;            // return corner-vertices of Point cloud to use in VISUALI-
    ZATION
86. }
```

```
87.
88.
89.
90. //-------------------------------------------------------------------------------------------------------
91. // VISUALIZE Rectangles with OpenGL/glut                    ***        https://gist.github.com/hkule-
    kci/2300262
92. //-------------------------------------------------------------------------------------------------------
93. #include <stdio.h>
94. #include <GL/glut.h>
95. #include <GL/freeglut.h>
96. #define KEY_ESC 27 /* GLUT doesn't supply this */
97.
98.
99. // --- Initial parameters ---
100.  // mouse operation: rotate around object
101.      int mouseDown = 0;
102.      float xrot = 10.0f;
103.      float yrot = -10.0f;
104.      float xdiff = 10.0f;
105.      float ydiff = 10.0f;
106.  // translate shape
107.      float tra_x = 0.0f;
108.      float tra_y = 0.0f;
109.      float tra_z = 0.0f;
110.  // View
111.      float grow_shrink = 72.0f;                  // fovy & zoom in/out
112.      float resize_f = 1.0f;
113.  // Output screen
114.      #define SCREEN_WIDTH 640
115.      #define SCREEN_HEIGHT 480
116.
117.
118.  // --- DRAW ---
119.
120.  void drawPointCloud()
121.  {
122.      vector<float> coorN = PointCloud();
123.      int norm = 10;
124.      for (int c = 0; c < coorN.size(); c++)
125.      {
126.          coorN[c] = coorN[c] / norm;
127.      }
128.
129.      //glTranslatef(tra_x-0.3f, tra_y-0.3f, tra_z-0.3f);
130.      //glTranslatef(tra_x, tra_y, tra_z);
131.
132.      glPointSize(10.0f);  //10?
133.      glBegin(GL_POINTS);
134.      glColor3f(0.0f, 0.0f, 0.0f);
135.
136.      for (int c = 0; c < coorN.size(); c+=3)
137.      {
138.          glVertex3f(coorN[c+0],     coorN[c+2],     -coorN[c+1]);
139.      }
140.
141.      glEnd();
142.
143.
144.  }
145.
146.  void drawBuildingFILL()
147.  {
148.          vector<vector<vector<float>>> coorN = BSD();                    // call coordinates vector
    from BSD() Function
149.          int norm = 10;                                          // divide by 10000, all numbers between 0
    and 1 (normalized)
150.          for (int c = 0; c < coorN.size(); c++)
151.          {
152.              for (int p = 0; p < 8; p++)
153.              {
154.                  for (int j = 0; j < 3; j++)
155.                  {
156.                      coorN[c][p][j] = coorN[c][p][j] / norm;
157.                  }
158.              }
159.          }
```

```
160.          //glTranslatef(tra_x-0.3f, tra_y-0.3f, tra_z-0.3f);                        // translate shape to
    middle of the screen
161.          glTranslatef(tra_x, tra_y, tra_z);
162.          glBegin(GL_QUADS);
163.
164.          // Color(RED, GREEN, BLUE)    0,0,0 = BLACK
165.          //glColor3f(0.8f, 0.8f, 0.8f);       // INPUT: light grey
166.          glColor3f(0.6f, 0.7f, 0.7f);       // OUTPUT: middle light blue-grey
167.          //glColor3f(0.4f, 0.5f, 0.6f);       // PREDICTION: blue-grey
168.
169.          int nShape = 13;
170.          int x = 0; int y = 2; int z = 1;
171.          for (int c = 0; c < coorN.size(); c++)
172.          {
173.          // // TOP
174.              glVertex3f(coorN[c][6][x],     coorN[c][6][y],     -coorN[c][6][z]);
175.              glVertex3f(coorN[c][2][x],     coorN[c][2][y],     -coorN[c][2][z]);
176.              glVertex3f(coorN[c][1][x],     coorN[c][1][y],     -coorN[c][1][z]);
177.              glVertex3f(coorN[c][5][x],     coorN[c][5][y],     -coorN[c][5][z]);
178.          // BOTTOM
179.              glVertex3f(coorN[c][4][x],     coorN[c][4][y],     -coorN[c][4][z]);
180.              glVertex3f(coorN[c][0][x],     coorN[c][0][y],     -coorN[c][0][z]);
181.              glVertex3f(coorN[c][3][x],     coorN[c][3][y],     -coorN[c][3][z]);
182.              glVertex3f(coorN[c][7][x],     coorN[c][7][y],     -coorN[c][7][z]);
183.          // FRONT
184.              glVertex3f(coorN[c][5][x],     coorN[c][5][y],     -coorN[c][5][z]);
185.              glVertex3f(coorN[c][1][x],     coorN[c][1][y],     -coorN[c][1][z]);
186.              glVertex3f(coorN[c][0][x],     coorN[c][0][y],     -coorN[c][0][z]);
187.              glVertex3f(coorN[c][4][x],     coorN[c][4][y],     -coorN[c][4][z]);
188.          // BACK
189.              glVertex3f(coorN[c][7][x],     coorN[c][7][y],     -coorN[c][7][z]);
190.              glVertex3f(coorN[c][3][x],     coorN[c][3][y],     -coorN[c][3][z]);
191.              glVertex3f(coorN[c][2][x],     coorN[c][2][y],     -coorN[c][2][z]);
192.              glVertex3f(coorN[c][6][x],     coorN[c][6][y],     -coorN[c][6][z]);
193.          // LEFT
194.              glVertex3f(coorN[c][1][x],     coorN[c][1][y],     -coorN[c][1][z]);
195.              glVertex3f(coorN[c][2][x],     coorN[c][2][y],     -coorN[c][2][z]);
196.              glVertex3f(coorN[c][3][x],     coorN[c][3][y],     -coorN[c][3][z]);
197.              glVertex3f(coorN[c][0][x],     coorN[c][0][y],     -coorN[c][0][z]);
198.          // RIGHT
199.              glVertex3f(coorN[c][6][x],     coorN[c][6][y],     -coorN[c][6][z]);
200.              glVertex3f(coorN[c][5][x],     coorN[c][5][y],     -coorN[c][5][z]);
201.              glVertex3f(coorN[c][4][x],     coorN[c][4][y],     -coorN[c][4][z]);
202.              glVertex3f(coorN[c][7][x],     coorN[c][7][y],     -coorN[c][7][z]);
203.          }
204.          glEnd();
205.  }
206.  void drawBuildingLINE()
207.  {
208.          vector<vector<vector<float>>> coorN = BSD();                    // call coordinates vector
    from BSD() Function
209.          float norm = 10;                                        // divide by 10000, all numbers between 0
    and 1 (normalized)
210.          for (int c = 0; c < coorN.size(); c++)
211.          {
212.              for (int p = 0; p < 8; p++)
213.              {
214.                  for (int j = 0; j < 3; j++)
215.                  {
216.                      coorN[c][p][j] = coorN[c][p][j] / norm;
217.                  }
218.              }
219.          }
220.          glBegin(GL_QUADS);
221.          glColor3f(0.0f, 0.0f, 0.0f);     // black
222.
223.          int nShape = 13;
224.          int x = 0; int y = 2; int z = 1;
225.          for (int c = 0; c < coorN.size(); c++)
226.          {
227.          // TOP
228.              glVertex3f(coorN[c][6][x],     coorN[c][6][y],     -coorN[c][6][z]);
229.              glVertex3f(coorN[c][2][x],     coorN[c][2][y],     -coorN[c][2][z]);
230.              glVertex3f(coorN[c][1][x],     coorN[c][1][y],     -coorN[c][1][z]);
231.              glVertex3f(coorN[c][5][x],     coorN[c][5][y],     -coorN[c][5][z]);
232.          // BOTTOM
233.              glVertex3f(coorN[c][4][x],     coorN[c][4][y],     -coorN[c][4][z]);
```

```
234.              glVertex3f(coorN[c][0][x],     coorN[c][0][y],     -coorN[c][0][z]);
235.              glVertex3f(coorN[c][3][x],     coorN[c][3][y],     -coorN[c][3][z]);
236.              glVertex3f(coorN[c][7][x],     coorN[c][7][y],     -coorN[c][7][z]);
237.          // FRONT
238.              glVertex3f(coorN[c][5][x],     coorN[c][5][y],     -coorN[c][5][z]);
239.              glVertex3f(coorN[c][1][x],     coorN[c][1][y],     -coorN[c][1][z]);
240.              glVertex3f(coorN[c][0][x],     coorN[c][0][y],     -coorN[c][0][z]);
241.              glVertex3f(coorN[c][4][x],     coorN[c][4][y],     -coorN[c][4][z]);
242.          // BACK
243.              glVertex3f(coorN[c][7][x],     coorN[c][7][y],     -coorN[c][7][z]);
244.              glVertex3f(coorN[c][3][x],     coorN[c][3][y],     -coorN[c][3][z]);
245.              glVertex3f(coorN[c][2][x],     coorN[c][2][y],     -coorN[c][2][z]);
246.              glVertex3f(coorN[c][6][x],     coorN[c][6][y],     -coorN[c][6][z]);
247.          // LEFT
248.              glVertex3f(coorN[c][1][x],     coorN[c][1][y],     -coorN[c][1][z]);
249.              glVertex3f(coorN[c][2][x],     coorN[c][2][y],     -coorN[c][2][z]);
250.              glVertex3f(coorN[c][3][x],     coorN[c][3][y],     -coorN[c][3][z]);
251.              glVertex3f(coorN[c][0][x],     coorN[c][0][y],     -coorN[c][0][z]);
252.          // RIGHT
253.              glVertex3f(coorN[c][6][x],     coorN[c][6][y],     -coorN[c][6][z]);
254.              glVertex3f(coorN[c][5][x],     coorN[c][5][y],     -coorN[c][5][z]);
255.              glVertex3f(coorN[c][4][x],     coorN[c][4][y],     -coorN[c][4][z]);
256.              glVertex3f(coorN[c][7][x],     coorN[c][7][y],     -coorN[c][7][z]);
257.          }
258.          glEnd();
259.  }
260.
261.  // --- INITIALIZATION ---
262.  int init(void)
263.  {
264.              glClearColor(1.0f, 1.0f, 1.0f, 1.0f);                    // background color(1f, 1f, 1f) &
    opaque(transparency)
265.              //glEnable(GL_DEPTH_TEST);                                    // enable depth-test, which
    shape is in front of another?
266.              //glDepthFunc(GL_LEQUAL);                                     // type of depth-test: LESS or
    EQUAL: Passes if the fragment's depth value is less than or equal to the stored depth value.
267.              //glClearDepth(1.0f);
268.
269.      //glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);                  // wireframe view (GL_LINE), or
    GL_POINT or GL_FILL
270.      glShadeModel(GL_SMOOTH);                                      // shading: _SMOOTH or _FLAT
271.
272.      // glEnable(GL_CULL_FACE);
273.      // glCullFace(GL_BACK);
274.
275.      // glEnable(GL_BLEND);
276.      // glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
277.
278.              return 1;
279.  }
280.
281.  // --- DISPLAY ---
282.  void display(void)
283.  {
284.              glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
285.              glLoadIdentity();
286.
287.      // camera position:
288.              gluLookAt(  0.0f, 0.0f, 1.0f,           // EYE
289.                          0.0f, 0.0f, -0.5f,           // AT
290.                          0.0f, 1.0f, 0.0f    );      // UP, corresponds to y-axis
291.
292.              glRotatef(xrot, 1.0f, 0.0f, 0.0f);
293.              glRotatef(yrot, 0.0f, 1.0f, 0.0f);
294.
295.
296.      glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
297.      drawBuildingFILL();
298.      glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
299.      drawBuildingLINE();
300.      glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
301.      drawPointCloud();
302.
303.              glFlush();
304.              glutSwapBuffers();
305.  }
306.
```

```
307.  // --- RESHAPE ---
308.  void resize(int w, int h)
309.  {
310.      // MATRIXMODE( "specifies which stack is the target for subsequent matrix operations.")
311.      //             mode: MODELVIEW, PROJECTION, TEXTURE, COLOR
312.            glMatrixMode(GL_PROJECTION);               // To operate on the Projection Matrix
313.            glLoadIdentity();
314.
315.      // VIEWPORT(     x, y:           lower left corner of the viewport rectangle, in pixels. Initial =
    (0,0),
316.      //               width, height:  of the viewport. When a GL context is first attached to a window,
    width and height are set to the dimensions of that window
317.            glViewport(0, 0, w, h);
318.
319.      // Set up a perspective projection matrix
320.      // PERSPECTIVE(  fovy:   field of view angle,
321.      //               aspect: ratio of x (width) to y (height),
322.      //               zNear:  distance from viewer to near clipping plane (always positive),
323.      //               zFar:   distance from viewer to far  clipping plane (always positive)   )
324.            //gluPerspective(grow_shrink, resize_f*SCREEN_WIDTH/SCREEN_HEIGHT, 0.0f, 2.0f);
325.      // ORTHOGRPHIC( left, right, bottom, top, nearVal, farVal )
326.      glOrtho(-1.5, 1.5, -1.5, 1.5, 0, 2);
327.
328.            glMatrixMode(GL_MODELVIEW);
329.            glLoadIdentity();
330.  }
331.
332.  // --- VIEW OPTIONS ---
333.  // ZOOM VIEW:  IN (I) / OUT (O)
334.  void keyboard(unsigned char key, int x, int y)
335.  {
336.            switch(key)
337.            {
338.                      case 27 :
339.                                exit(1);
340.                                break;
341.
342.          // FORWARD
343.          case 'w':
344.                                tra_z += 0.1f;
345.                                break;
346.          // BACKWARD
347.                      case 's':
348.                                tra_z -= 0.1f;
349.                                break;
350.          // LEFT
351.                      case 'a':
352.                                tra_x -= 0.1f;
353.                                break;
354.          // RIGHT
355.                      case 'd':
356.                                tra_x += 0.1f;
357.                                break;
358.
359.          // Zoom OUT
360.                      case 'o':
361.                                grow_shrink++;
362.                                resize(SCREEN_WIDTH, SCREEN_HEIGHT);
363.                                break;
364.          // Zoom IN
365.                      case 'i':
366.                                grow_shrink--;
367.                                resize(SCREEN_WIDTH, SCREEN_HEIGHT);
368.                                break;
369.            }
370.            glutPostRedisplay();
371.  }
372.
373.  // ORBIT VIEW:   Rotate with left mouse button around object shape
374.  void mouse(int button, int state, int x, int y)
375.  {
376.            if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
377.            {
378.                      mouseDown = 1;
379.
380.                      xdiff = x - yrot;
381.                      ydiff = -y + xrot;
```

```
382.            }
383.            else
384.                    mouseDown = 0;
385.    }
386.    void mouseMotion(int x, int y)
387.    {
388.            if (mouseDown)
389.            {
390.                    yrot = x - xdiff;
391.                    xrot = y + ydiff;
392.
393.                    glutPostRedisplay();
394.            }
395.    }
396.
397.
398.    // --- MAIN FUNCTION ---
399.    int main(int argc, char *argv[])
400.    {
401.            glutInit(&argc, argv);
402.
403.            glutInitWindowPosition(0, 0);
404.            glutInitWindowSize(SCREEN_WIDTH, SCREEN_HEIGHT);
405.
406.            glutInitDisplayMode(GLUT_RGB);
407.
408.            glutCreateWindow("--- Building ---");
409.
410.            glutDisplayFunc(display);
411.        glutKeyboardFunc(keyboard);
412.            glutMouseFunc(mouse);
413.            glutMotionFunc(mouseMotion);
414.            glutReshapeFunc(resize);
415.
416.            if (!init())
417.                    return 1;
418.
419.            glutMainLoop();
420.
421.            return 0;
422.    }
423.
```

D.3 Modified C++ files in BSO-toolbox v1.0.0

The files, “ms_building.cpp”, “ms_building.hpp”, “ms_space.cpp”, and “ms_space.hpp”, are modified to include non-orthogonal designs in the toolbox. The files, “construction.cpp” and “structure.cpp”, are modified to solve error messages that occurred after the transfer to a new workstation. The file, “quad_hexahedron.hpp”, is modified by the author, to the purpose of this thesis. The modified C++ files are listed below. The files are compared with the original BSO-toolbox v1.0.0 and the differences are indicated.

2 bso/building_physics/properties/construction.cpp

```
67          mResistanceSide1 = rhs.mResistanceSide1;
68          mResistanceSide2 = rhs.mResistanceSide2;
69          mRelativeMeasurePoint =
    rhs.mRelativeMeasurePoint;
70  }
71
72  } // namespace properties
```

```
67          mResistanceSide1 = rhs.mResistanceSide1;
68          mResistanceSide2 = rhs.mResistanceSide2;
69          mRelativeMeasurePoint =
    rhs.mRelativeMeasurePoint;
70  +
71  +       return *this;
72  }
73
74  } // namespace properties
```

136 bso/spatial_design/ms_building.cpp

```
19  } // ms_building() (empty constructor)
20
21  ms_building::ms_building(std::string fileName)
22  { // initilization by string or text file
23          mLastSpaceID = 0;
24          std::ifstream input;
25          if (!fileName.empty())
    input.open(fileName.c_str());
35
36          std::string line;
37
38          while (!input.eof()) // Parse the input file
    line by line
39          {
40                  getline(input,line); // get next line
```

```
19  } // ms_building() (empty constructor)
20
21  ms_building::ms_building(std::string fileName)
22  + :       insertFileName(fileName)
23  { // initilization by string or text file
24  +       //std::cout << "In ms file is the
    insertedFileName: " << insertFileName << std::endl;
25  +       //std::cout << "In ms file is the
    insertedFileName: " << getInsertFileName() <<
    std::endl;
26          mLastSpaceID = 0;
27          std::ifstream input;
28          if (!fileName.empty())
    input.open(fileName.c_str());
38
39          std::string line;
40
41  +
42          while (!input.eof()) // Parse the input file
    line by line
43          {
44                  getline(input,line); // get next line
```

```
from the file
                        {
                                continue; // continue to next line
                        }
-                       else if (line.substr(0,2) == "R," || line.substr(0,2) == "r,")
                        {
-                               line.erase(0,2);
                                mSpaces.push_back(new ms_space(line));
                        }
                        else
                mSpaces.push_back(new ms_space(*i));
        }


        this->checkValidity();
} // ms_building() (copy constructor)

                std::vector<unsigned int> originIndices = {sc.getWSize(), sc.getDSize(), sc.getHSize()};
                std::vector<std::vector<double> > globalCoords(3);
                std::vector<std::vector<double> > dimensions(3);
-
                for (unsigned int i = 0; i < 3; i++)
                {
                        globalCoords[i] = std::vector<double>(originIndices[i]+1);
                        dimensions[i] = std::vector<double>(originIndices[i]);
-                       globalCoords[i][0] = 0.0;
-                       for (unsigned int j = 0; j < originIndices[i]; ++j)
-                       {
-                               if (i == 0) dimensions[i][j] = sc.getWValue(j);
-                               if (i == 1) dimensions[i][j] = sc.getDValue(j);
-                               if (i == 2) dimensions[i][j] = sc.getHValue(j);
-                               globalCoords[i][j+1] = globalCoords[i][j] + dimensions[i][j];
```

```
from the file
                        {
                                continue; // continue to next line
                        }
+                       else if (line.substr(0,2) == "R," || line.substr(0,2) == "r," || line.substr(0,2) == "N," || line.substr(0,2) == "n,")
                        {
+                               //line.erase(0,2); // line.erase now happens in ms_space for sDefmethod definition (space definitiond method)
                                mSpaces.push_back(new ms_space(line));
                        }
                        else
                mSpaces.push_back(new ms_space(*i));
        }

+       insertFileName = rhs.getInsertFileName();
+
        this->checkValidity();
} // ms_building() (copy constructor)

                std::vector<unsigned int> originIndices = {sc.getWSize(), sc.getDSize(), sc.getHSize()};
                std::vector<std::vector<double> > globalCoords(3);
                std::vector<std::vector<double> > dimensions(3);

                for (unsigned int i = 0; i < 3; i++)
                {
                        globalCoords[i] = std::vector<double>(originIndices[i]+1);
                        dimensions[i] = std::vector<double>(originIndices[i]);
+               }
+               for (unsigned int i = 0; i < originIndices[0]; i++) dimensions[0][i] = sc.getWValue(i);
+               for (unsigned int i = 0; i < originIndices[1]; i++) dimensions[1][i] = sc.getDValue(i);
+               for (unsigned int i = 0; i < originIndices[2]; i++) dimensions[2][i] = sc.getHValue(i);
+
+               // find the lowest cell indices (w,d,h) that index an active cell (i.e. they will represent the origin)
+               for (unsigned int i = 1; i < sc.getBRowSize(); i++)
```

```
                        }
                }

-               for (unsigned int i = 0; i < sc.getBSize(); ++i)
-               { // for each space i
-                       // find the indices of the minium and maximum cell index that is active for space i
-                       unsigned int maxW = 0, maxD = 0, maxH = 0, minW = originIndices[0],
-                       minD = originIndices[1], minH = originIndices[1];
-                       bool empty = true;
-                       for (unsigned int j = 1; j < sc.getBRowSize()+1; ++j)
-                       {
-                               if (sc.getBValue(i,j) != 1) continue;
-                               empty = false;
-                               unsigned int wInd = sc.getWIndex(j);
-                               unsigned int dInd = sc.getDIndex(j);
-                               unsigned int hInd = sc.getHIndex(j);
+               { // for each cell
+                       std::vector<unsigned int> indices = {sc.getWIndex(i), sc.getDIndex(i), sc.getHIndex(i)}; // temporarily store the indices (w,d,h)
+                       for (unsigned int j = 0; j < sc.getBSize(); j++)
+                       { // for each space
+                               if (sc.getBValue(j,i) == 1)
+                               { // if the cell i is active for space j
+                                       for (unsigned int k = 0; k < 3; k++)
+                                       { // check for each index
+                                               if (indices[k] < originIndices[k])
+                                               { // if it is lower than the lowest found so far to index an active cell
+                                                       originIndices[k] = indices[k]; // if it is, set the lowest found to it.
+                                               }
+                                       }
+                               }
                        }
                }

+               for (unsigned int i = 0; i < 3; i++)
+               { // for each index
+                       globalCoords[i][originIndices[i]] = 0.0; // set the origins coordinate value to zero
+                       for (unsigned int j = originIndices[i] + 1; j < globalCoords[i].size(); j++)
+                       { // for each consequent index value increment with the width of the dimensions described by the supercube
+                               globalCoords[i][j] = globalCoords[i][j-1] + dimensions[i][j-1];
+                       }
```

```
-
-                                if (minW > wInd) minW = wInd;
-                                if (minD > dInd) minD = dInd;
-                                if (minH > hInd) minH = hInd;
-
-                                if (maxW < wInd) maxW = wInd;
-                                if (maxD < dInd) maxD = dInd;
-                                if (maxH < hInd) maxH = hInd;
                    }                                                   }
-                        if (empty)                                      +
                                                                        +                   for (unsigned int i = 0; i < sc.getBSize(); i++)
                                                                        +                   { // for each space
                                                                        +                           // find the indices of the minium and maximum cell index that is active for space i
                                                                        +                           unsigned int max = 0, min = 0;
                                                                        +                           for (unsigned int j = 1; j <= sc.getBRowSize(); j++)
                        {                                                                   {
-                                std::stringstream errorMessage;         +                                   if (min == 0 && sc.getBValue(i,j) == 1) min = j;
-                                errorMessage << "\nError, while converting SC building into MS building\n"
+                                                                        +                                   if (sc.getBValue(i,j) == 1) max = j;
-                                << "encountered a space without cell assignments\n"
-                                << "(bso/spatial_design/ms_building.cpp)." << std::endl;
-                                throw std::runtime_error(errorMessage.str());
                        }                                                           }
-
                        //get the locations and dimensions from these indices
                        utilities::geometry::vertex location;                       utilities::geometry::vertex location;
-                       location << globalCoords[0][minW],                  +       location << globalCoords[0][sc.getWIndex(min)],
-                       globalCoords[1][minD],                              +       globalCoords[1][sc.getDIndex(min)],
-                       globalCoords[2][minH];                              +       globalCoords[2][sc.getHIndex(min)];
                        location.round(0);                                          location.round(0);
-
                        utilities::geometry::vector dimensions;                     utilities::geometry::vector dimensions;
-                       dimensions << globalCoords[0]                       +       dimensions << globalCoords[0]
```

```
[maxW+1],
-
globalCoords[1][maxD+1],
-
globalCoords[2][maxH+1];
                        dimensions -= location;
                        dimensions.round(0);

                                mLastSpaceID =
mSpaces.back()->getID();
                        }
                }
-
-               this->resetOrigin();
        }
        catch (std::exception& e)
        {
```

```
void ms_building::setZZero()
{
-       this->resetOrigin({2});

- } // setZZero()
-
- void ms_building::resetOrigin(const
std::vector<unsigned int>& indices /*= {0,1,2}*/)
- {
-       utilities::geometry::vector coordDifference =
{0,0,0};
-       for (const auto& index : indices)
-       {
-               double min = mSpaces[0]-
>getCoordinates()(index); // set an initial value to
the minimum

        // find the minimum value of the z-coordinates
in the building
-               for (auto i : mSpaces)

-               {
-                       double coord = i-
>getCoordinates()(index);
-                       if (min > coord) min = coord;
-               }
-
-               coordDifference(index) -= min;
-       }

-       if (coordDifference(2) > 0) coordDifference(2)
= 0;


        for (auto i : mSpaces) i->setCoordinates(i-
>getCoordinates() + coordDifference);
-       this->snapOn({{0,1},{1,1},{2,1}});
```

```
[sc.getWIndex(max)+1],
+
globalCoords[1][sc.getDIndex(max)+1],
+
globalCoords[2][sc.getHIndex(max)+1];
                        dimensions -= location;
                        dimensions.round(0);

                                mLastSpaceID =
mSpaces.back()->getID();
                        }
                }


        }
        catch (std::exception& e)
        {
```

```
void ms_building::setZZero()
{
+       double min = mSpaces[0]->getCoordinates()(2);
// set an initial value to the minimum

        // find the minimum value of the z-coordinates
in the building
+       for (auto i : mSpaces) if (i->getCoordinates()
(2) < min) min = i->getCoordinates()(2);
+       if (min <= 0) return;


+       utilities::geometry::vector coordDifference;

+       coordDifference << 0, 0, -min;

        for (auto i : mSpaces) i->setCoordinates(i-
>getCoordinates() + coordDifference);
+ } // setZZero()
```

```
- } // resetOrigin()

void ms_building::addSpace(const ms_space& space)
{
                        utilities::geometry::vertex p2 = i->getDimensions() + p1;
                        for (unsigned int j = 0; j < 3; j++)
                        {

coordValues[j].push_back(p1(j));

coordValues[j].push_back(p2(j));
                        }
                {

sc.mBValues.push_back(std::vector<int>(cubeSize + 1));
                        sc.mBValues.back()[0] = i->getID();
-
                        utilities::geometry::vertex p1 = i->getCoordinates();
                        utilities::geometry::vertex p2 = i->getDimensions() + p1;

                        for (unsigned int j = 1; j < sc.mBValues.back().size(); j++)
                        {
-
                utilities::geometry::vertex pCheck;
-                               pCheck[0] = coordValues[0][sc.getWIndex(j)] + coordValues[0][sc.getWIndex(j)+1];
-                               pCheck[1] = coordValues[1][sc.getDIndex(j)] + coordValues[1][sc.getDIndex(j)+1];
-                               pCheck[2] = coordValues[2][sc.getHIndex(j)] + coordValues[2][sc.getHIndex(j)+1];
-                               pCheck /= 2;


                                bool belongsToSpace = true;
                                for (unsigned int k = 0; k < 3; k++)
                                {
-                                       if (!(pCheck[k] > p1[k] && pCheck[k] < p2[k]))
                                        {

                belongsToSpace = false;
```

```
void ms_building::addSpace(const ms_space& space)
{
                        utilities::geometry::vertex p2 = i->getDimensions() + p1;
                        for (unsigned int j = 0; j < 3; j++)
                        {
+                               if (p1[j] < minMSPoint[j]) minMSPoint[j] = p1[j];

                coordValues[j].push_back(p1(j));

                coordValues[j].push_back(p2(j));
                        }
                {

                sc.mBValues.push_back(std::vector<int>(cubeSize + 1));
                        sc.mBValues.back()[0] = i->getID();

                        utilities::geometry::vertex p1 = i->getCoordinates();
                        utilities::geometry::vertex p2 = i->getDimensions() + p1;

                        for (unsigned int j = 1; j < sc.mBValues.back().size(); j++)
                        {
+
                utilities::geometry::vertex pCheck = minMSPoint;
+
                                for (unsigned int k = 0; k < sc.getWIndex(j); ++k) pCheck[0] += sc.getWValue(k);
                                for (unsigned int k = 0; k < sc.getDIndex(j); ++k) pCheck[1] += sc.getDValue(k);
                                for (unsigned int k = 0; k < sc.getHIndex(j); ++k) pCheck[2] += sc.getHValue(k);

                                bool belongsToSpace = true;
                                for (unsigned int k = 0; k < 3; k++)
                                {
                                        if (pCheck[k] < p1[k] || pCheck[k] >= p2[k])
                                        {

                belongsToSpace = false;
```

```
897                                                     break;
901                             }
902                         }
903                         sc.checkValidity();
904 -
905                         return sc;
906                 }
907                 catch (std::exception& e)
```

```
888                                                     break;
892                             }
893                         }
894                         sc.checkValidity();
895                         return sc;
896                 }
897                 catch (std::exception& e)
```

3 ■■■■■ bso/spatial_design/ms_building.hpp

```
15          std::vector<ms_space*> mSpaces;
16          mutable unsigned int mLastSpaceID;
17          void checkValidity() const;
18  public:
19          ms_building(); // empty constructor
20          ms_building(std::string fileName); //
    initilization by string or text file
21          ms_building(const ms_building& rhs); // copy
    constructor
22          ms_building(const sc_building& sc); // convert
    SC to MS
23          ~ms_building(); // destructor
24
25          void writeToFile(std::string fileName) const;
26          std::vector<ms_space*> getSpacePtrs() const;
27          ms_space* getSpacePtr(const ms_space& space)
    const;
36                  const bool includePartialSpaces =
    false) const;
37
38          void setZZero();
39  -       void resetOrigin(const std::vector<unsigned
    int>& indices = {0,1,2});
40          void addSpace(const ms_space& space);
41          void deleteSpace(ms_space* spacePtr);
42          void deleteSpace(ms_space& space);
```

```
15          std::vector<ms_space*> mSpaces;
16          mutable unsigned int mLastSpaceID;
17          void checkValidity() const;
18  +       std::string insertFileName;
19  public:
20          ms_building(); // empty constructor
21          ms_building(std::string fileName); //
    initilization by string or text file
22          ms_building(const ms_building& rhs); // copy
    constructor
23          ms_building(const sc_building& sc); // convert
    SC to MS
24          ~ms_building(); // destructor
25
26  +       const std::string getInsertFileName()
    const{return insertFileName;}
27          void writeToFile(std::string fileName) const;
28          std::vector<ms_space*> getSpacePtrs() const;
29          ms_space* getSpacePtr(const ms_space& space)
    const;
38                  const bool includePartialSpaces =
    false) const;
39
40          void setZZero();
41          void addSpace(const ms_space& space);
42          void deleteSpace(ms_space* spacePtr);
43          void deleteSpace(ms_space& space);
```

108 ■■■■■ bso/spatial_design/ms_space.cpp

```
...     @@ -1,6 +1,7 @@
1   #ifndef MS_SPACE_CPP
2   #define MS_SPACE_CPP
3
4   #include <sstream>
5   #include <string>
6   #include <vector>
58
59  ms_space::ms_space(std::string line)
60  {
```

```
1   #ifndef MS_SPACE_CPP
2   #define MS_SPACE_CPP
3
4   + #include <iostream>
5   #include <sstream>
6   #include <string>
7   #include <vector>
59
60  ms_space::ms_space(std::string line)
61  {
62  +       // substract sDefMethod
```

```
        // tokenize the line

        boost::char_separator<char> sep(","); // defines what separates tokens in a string
        typedef boost::tokenizer< boost::char_separator<char> > t_tokenizer; // settings for the boost::tokenizer
        t_tokenizer tokens(line, sep); // this is where the tokenized line will be stored
        t_tokenizer::iterator token = tokens.begin(); // set iterator to first token
        int number_of_tokens = std::distance( tokens.begin(), tokens.end()); // count the number of tokens int the line

        try
-       {
-               // get the geometry of the space
                utilities::geometry::vertex temp;

-               mID = utilities::trim_and_cast_int(*(token));
-               temp(0) = utilities::trim_and_cast_double(*(++token));
-               temp(1) = utilities::trim_and_cast_double(*(++token));
-               temp(2) = utilities::trim_and_cast_double(*(++token));
-               mDimensions = temp;
                        temp(0) = utilities::trim_and_cast_double(*(++token));
                        temp(1) = utilities::trim_and_cast_double(*(++token));
                        temp(2) = utilities::trim_and_cast_double(*(++token));
-               mCoordinates = temp;

-               // handle the tokens
```

```
+
+       sDefMethod = line.substr (0,1); // space definition method; orthogonal or non-orthogonal
+       line.erase(0,2); // Removes the sDefMethod part from the line variable
+
+
        // tokenize the line
+
        boost::char_separator<char> sep(","); // defines what separates tokens in a string
        typedef boost::tokenizer< boost::char_separator<char> > t_tokenizer; // settings for the boost::tokenizer
        t_tokenizer tokens(line, sep); // this is where the tokenized line will be stored
        t_tokenizer::iterator token = tokens.begin(); // set iterator to first token
        int number_of_tokens = std::distance( tokens.begin(), tokens.end()); // count the number of tokens int the line

+
        try
+       {// store values in string line in mDimensions and mCoordinates or pVertex with first value from line variable as mID
+               mID = utilities::trim_and_cast_int(*(token));
                utilities::geometry::vertex temp;

+               for(int i=0; i<((number_of_tokens-1)/3); i++)
+               {
+                       pVertex.push_back(new utilities::geometry::vertex);
+
                        temp(0) = utilities::trim_and_cast_double(*(++token));
                        temp(1) = utilities::trim_and_cast_double(*(++token));
                        temp(2) = utilities::trim_and_cast_double(*(++token));

+                       *pVertex[i] = temp;
+               }
+
+
+               if (sDefMethod == "N" || sDefMethod == "n")
+               {// handle the tokens for the non-
```

```
orthogonal cases
```

```
                        switch (number_of_tokens)
                        {
-                       case 7: // no space nor surface types have been defined
```

```
                                switch (number_of_tokens)
                                {
+                                       case 19: // 6 corner polyhedron
+                                       {
+                                               break;
+                                       }
+                                       case 25: // 8 corner polyhedron
+                                       {
+                                               break;
+                                       }
+                                       case 31: // 10 corner polyhedron
+                                       {
+                                               break;
+                                       }
+                                       default:
+                                       {
+                                       std::stringstream errorMessage;
+                                       errorMessage << std::endl
+ << "An invalid amount of input argument was encoutered when trying to initialize the " << sDefMethod << " method defined space with ID " << mID << "." << std::endl
+ << "When trying to parse the following input line: " << std::endl
+ << line << std::endl
+ << "The determined number_of_tokens are: " << number_of_tokens << " and should be: 25" << std::endl
+ << "(bso/spatial_design/ms_space.cpp)" << std::endl;
+                                       throw std::invalid_argument(errorMessage.str());
+
+                                       break;
+                                       }
+                               }
+                       }
+                       else if (sDefMethod == "R" || sDefMethod == "r") //number_of_tokens = 7
+                       {
+                               //std::cout << "dit is een try out" << std::endl;
+
+                               mDimensions = *pVertex[0];
+                               mCoordinates = *pVertex[1];
+                               pVertex.clear(); // A clearing
```

	of values from pVertex to avoid mis use
	+
	+ switch (number_of_tokens)
	+ { // handle the tokens for the orthogonal cases
	+ case 7: // orthogonal regtangal; no space nor surface types have been defined
{	{
break;	break;
}	}
- case 8: // only a space type has been defined	+ case 8: // orthogonal regtangal; only a space type has been defined
{	{
mSpaceType = *(++token);	mSpaceType = *(++token);
boost::algorithm::trim(mSpaceType);	boost::algorithm::trim(mSpaceType);
break;	break;
}	}
- case 13: // only surface types have been defind	+ case 13: // orthogonal regtangal; only surface types have been defind
- case 14: // both a space type and surface types have been defined	+ case 14: // orthogonal regtangal; both a space type and surface types have been defined
{	{
if (number_of_tokens == 14)	if (number_of_tokens == 14)
{	{
mSpaceType = *(++token);	mSpaceType = *(++token);
boost::algorithm::trim(mSpaceType);	boost::algorithm::trim(mSpaceType);
-	
}	}
mSurfaceTypes.clear();	mSurfaceTypes.clear();
{	{
std::stringstream errorMessage;	std::stringstream errorMessage;
errorMessage << std::endl	errorMessage << std::endl
- << "An invalid amount of input argument was encoutered when trying to initialize the space with ID " << mID << "." << std::endl	+ << "An invalid amount of input argument was encoutered when trying to initialize the " << sDefMethod << " method defined space with ID " << mID << "." << std::endl
<< "When trying to parse the following input line: " << std::endl	<< "When trying to parse the following input line: " << std::endl
<< line << std::endl	<< line << std::endl
<< "(bso/spatial_design/ms_space.cpp)" << std::endl;	<< "(bso/spatial_design/ms_space.cpp)" << std::endl;

```
141                                     throw
    std::invalid_argument(errorMessage.str());
142
143                                     break;
144                                 }
145 -               }
146         }
147         catch (std::exception& e)
148         {
155                 throw
    std::invalid_argument(errorMessage.str());
156         }
157
158                 try
159                 {
160                         checkValidity();
169
    << e.what() << std::endl;
170                         throw
    std::invalid_argument(errorMessage.str());
171                 }
172 } // ms_space()
173
174 ms_space::ms_space(const ms_space& rhs)
178         mID           = rhs.mID;
179         mSpaceType    = rhs.mSpaceType;
180         mSurfaceTypes = rhs.mSurfaceTypes;
181
182                 try
183                 {
184                         checkValidity();
192
    << e.what() << std::endl;
193                         throw
    std::invalid_argument(errorMessage.str());
194                 }
195 } // ms_space()
196
197 ms_space::~ms_space()
210
211 bool ms_space::checkValidity() const
```

```
189                                     throw
    std::invalid_argument(errorMessage.str());
190
191                                     break;
192                                 }
193 +                       } // switch statement
194 +               } // if statement for sDefMethod ==
    "R" or "r"
195         }
196         catch (std::exception& e)
197         {
204                 throw
    std::invalid_argument(errorMessage.str());
205         }
206
207 +       if (sDefMethod == "R" || sDefMethod == "r")
    //number_of_tokens = 7
208 +       {
209                 try
210                 {
211                         checkValidity();
220
    << e.what() << std::endl;
221                         throw
    std::invalid_argument(errorMessage.str());
222                 }
223 +       }
224 } // ms_space()
225
226 ms_space::ms_space(const ms_space& rhs)
230         mID           = rhs.mID;
231         mSpaceType    = rhs.mSpaceType;
232         mSurfaceTypes = rhs.mSurfaceTypes;
233 +       pVertex           = rhs.pVertex; // Tessa
    Defined
234 +       sDefMethod        = rhs.sDefMethod;
235
236 +       if (sDefMethod == "R" || sDefMethod == "r") //
    the checkValidity test is specifically for the R space
    type insertion method.
237 +       {
238                 try
239                 {
240                         checkValidity();
248
    << e.what() << std::endl;
249                         throw
    std::invalid_argument(errorMessage.str());
250                 }
251 +       }
252 } // ms_space()
253
254 ms_space::~ms_space()
267
268 bool ms_space::checkValidity() const
```

```
212     {
213 -           if (mDimensions.minCoeff() < 0)
214         {
215                 std::stringstream errorMessage;
216                 errorMessage << std::endl
219 << "(bso/spatial_design/ms_space.hpp)" << std::endl;
220                 throw
std::invalid_argument(errorMessage.str());
221         }
222     }
223
224     void ms_space::setCoordinates(const
utilities::geometry::vertex& coords)
225     {
226         try
227         {
228                 mCoordinates = coords;
229 -               checkValidity();
230         }
231         catch(std::exception& e)
232         {
278     utilities::geometry::quad_hexahedron
ms_space::getGeometry() const
279     { // return the geometry of the space
280         std::vector<utilities::geometry::vertex>
cornerPoints;
281                 cornerPoints.reserve(8);
282                 std::vector<unsigned int> vertexOrder
= {0,1,3,2,4,5,7,6};
283                 for (const auto& i : vertexOrder)
289                         }
290
cornerPoints.push_back(tempPoint);
291                 }
292         return
bso::utilities::geometry::quad_hexahedron(cornerPoints
);
293     } // getGeometry
```

```
269     {
270 +           bool check = mDimensions.minCoeff() < 0;
271 +           if (check)
272         {
273                 std::stringstream errorMessage;
274                 errorMessage << std::endl
277 << "(bso/spatial_design/ms_space.hpp)" << std::endl;
278                 throw
std::invalid_argument(errorMessage.str());
279         }
280 +           return check;
281     }
282
283     void ms_space::setCoordinates(const
utilities::geometry::vertex& coords)
284     {
285         try
286         {
287                 mCoordinates = coords;
288 +               //checkValidity();
289         }
290         catch(std::exception& e)
291         {
337     utilities::geometry::quad_hexahedron
ms_space::getGeometry() const
338     { // return the geometry of the space
339         std::vector<utilities::geometry::vertex>
cornerPoints;
340 +
341 +           if (sDefMethod == "R" || sDefMethod == "r")
342 +           {
343                 cornerPoints.reserve(8);
344                 std::vector<unsigned int> vertexOrder
= {0,1,3,2,4,5,7,6};
345                 for (const auto& i : vertexOrder)
351                         }
352
cornerPoints.push_back(tempPoint);
353                 }
354 +           }
355 +
356 +           else if (sDefMethod == "N" || sDefMethod ==
"n")
357 +           {
358 +               for (int i=0; i<pVertex.size(); i++)
359 +               {
360 +
cornerPoints.push_back(*pVertex[i]);
361 +               }
362 +           }
363         return
bso::utilities::geometry::quad_hexahedron(cornerPoints
);
364     } // getGeometry
```

```
294                                                     365
                                                        366 +
295  bool ms_space::getSpaceType(std::string& spaceType)  367  bool ms_space::getSpaceType(std::string& spaceType)
     const                                                   const
296  {                                                  368  {
297          spaceType = mSpaceType;                    369          spaceType = mSpaceType;
```

3 ■■■■■ bso/spatial_design/ms_space.hpp

```
12   private:                                           12   private:
13           utilities::geometry::vertex mCoordinates; //  13           utilities::geometry::vertex mCoordinates; //
     {x,y,z}                                                 {x,y,z}
14           utilities::geometry::vector mDimensions; //   14           utilities::geometry::vector mDimensions; //
     {width,depth,height}                                    {width,depth,height}
                                                        15 +         std::vector <utilities::geometry::vertex*>
                                                                pVertex; //Contains all vertixes of non-orthogonal
                                                                space {x,y,z}
                                                        16 +
15           unsigned int mID; // Identification number of  17           unsigned int mID; // Identification number of
     the space                                               the space
                                                        18 +         std::string sDefMethod; // A space definition
                                                                method; orthogonal or non-orthogonal
16           std::string mSpaceType; // a type which can be 19          std::string mSpaceType; // a type which can be
     defined by a user and which can subsequently be         defined by a user and which can subsequently be
     assigned to the space                                   assigned to the space
17           std::vector<std::string> mSurfaceTypes; // each 20          std::vector<std::string> mSurfaceTypes; // each
     surface can also be assigned a user defined type.       surface can also be assigned a user defined type.
     Convention: {+y,+x,-y,-x,+z,-                           Convention: {+y,+x,-y,-x,+z,-
     z}/{north,east,south,west,top,bottom}                   z}/{north,east,south,west,top,bottom}
18           void reset(); // resets the space to initial  21          void reset(); // resets the space to initial
     values                                                  values
```

3 ■■■■■ bso/structural_design/component/structure.cpp

```
158                                                     158
159          bool structure::checkBadRequest(std::string 159          bool structure::checkBadRequest(std::string
     variable) const                                         variable) const
160          {                                          160          {
                                                        161 +                bool badRequest = false;
161                  if (variable == "E"                162                  if (variable == "E"
     && !mEAssigned                    ||                    && !mEAssigned                    ||
162                                    variable == "poisson" 163                                  variable == "poisson"
     && !mPoissonAssigned ||                                 && !mPoissonAssigned ||
163                                    variable == "A"    164                                  variable == "A"
     && !mAAssigned                    ||                    && !mAAssigned                    ||
164                                    variable == "width" 165                                  variable == "width"
     && !mWidthAssigned       ||                             && !mWidthAssigned       ||
165                                    variable == "height" 166                                 variable == "height"
     && !mHeightAssigned      ||                             && !mHeightAssigned      ||
166                                    variable ==        167                                   variable ==
     "thickness" && !mThicknessAssigned)                     "thickness" && !mThicknessAssigned)
167                  {                                  168                  {
                                                        169 +                        badRequest = true;
168                          std::stringstream          170                          std::stringstream
```

```
                errorMessage;
169             errorMessage << "\nBad request for variable: " << variable << "\n"
170             << "From component with type: " << mType << "\n"
171             << "Was the request valid? Was it assigned to the component?\n"
172             << "(bso/structural_design/component/structure.cpp)" << std::endl;
173             throw std::runtime_error(errorMessage.str());
174         }
175     } // badRequest()
176
177     structure::structure()
```

```
                errorMessage;
171             errorMessage << "\nBad request for variable: " << variable << "\n"
172             << "From component with type: " << mType << "\n"
173             << "Was the request valid? Was it assigned to the component?\n"
174             << "(bso/structural_design/component/structure.cpp)" << std::endl;
175             throw std::runtime_error(errorMessage.str());
176         }
+177         return badRequest;
178     } // badRequest()
179
180     structure::structure()
```

7 ■■■■■ bso/utilities/geometry/quad_hexahedron.hpp

```
22          double getVolume() const;
23          bool isInside(const vertex& p1, const double& tol = 1e-3) const;
24          bool isInsideOrOn(const vertex& p1, const double& tol = 1e-3) const;
25      };
26
27  } // namespace geometry
```

```
22          double getVolume() const;
23          bool isInside(const vertex& p1, const double& tol = 1e-3) const;
24          bool isInsideOrOn(const vertex& p1, const double& tol = 1e-3) const;
+25
+26         // Sebas: add operator so quads can be sorted in map<>
+27         bool operator<(const quad_hexahedron& rhs) const
+28         {
+29             return getVertices() < rhs.getVertices();
+30         }
+31         double fitnessQ = 0;
32      };
33
34  } // namespace geometry
```