# Early Phase Performance Driven Design Assistance Using Generative Models

HARVARD UNIVERSITY
Graduate School of Design

THESIS ACCEPTANCE CERTIFICATE

The undersigned, appointed by the Doctor of Design Program, have examined a dissertation entitled

Early Phase Performance Driven Design Assistance Using Generative Models

Presented by
Spyridon Ampanavos

candidate for the Doctor of Design degree and hereby certify that it is worthy of acceptance.

*Signature* [illegible]

Ali Malkawi

*Signature* [illegible]
Panagiotis Michalatos (May 23, 2022 18:54 EDT)

Panagiotis Michalatos

*Signature* [illegible]

Andrew Witt

*Date*: May 20, 2022

# Early Phase Performance Driven Design Assistance Using Generative Models

A dissertation presented by

Spyridon Ampanavos

Dipl. Arch. Eng., National Technical University of Athens

MDes, Harvard Graduate School of Design

to

Harvard University Graduate School of Design

in partial fulfillment of the requirements

for the degree of

Doctor of Design

Harvard University

Cambridge, Massachusetts

May 2022

Dissertation Advisor: Professor Ali Malkawi Spyridon Ampanavos

# Early Phase Performance Driven Design Assistance Using Generative Models

## Abstract

Form-finding in the current performance-driven design methodology of architectural design is typically formulated as a design optimization problem. Although effective in engineering or late-stage design problems, optimization is not suitable for the exploratory design phase due to the time intensity and cognitive load associated with the processes involved in the formulation and solution of optimization problems. The iterative, diverging nature of early-phase design is incompatible with the i) cognitive load of parametric modeling and its limited affordances for conceptual changes, ii) time and resource intensity of simulations, iii) interpretability of optimization results.

This thesis suggests a framework for generating optimal performance geometries within an intuitive and interactive modeling environment in real-time. The framework includes the preparation of a synthetic dataset, modeling its probability distribution using generative models, and sampling the learned distribution under given constraints. The several components are elaborated through a case study of building form optimization for passive solar gain in Boston, MA, for a wide range of plot shapes and surroundings. Apart from the overall framework, this thesis contributes a series of methods that enable its implementation. A geometric system of orientable cuboids is introduced as a generalizable, granular modeling vocabulary. A method for efficient boundary condition sampling is suggested for the dataset preparation. A Variational Autoencoder (VAE) is extended for performance-aware geometry generation using performance-related loss functions. A series of techniques inspired by the data-imputation literature is introduced to generate optimal geometries under constraints. Last, a prototype is presented that demonstrates the abilities of a system based on the suggested framework.

# Acknowledgments

I would like to thank my advisor, Professor Ali Malkawi, for his mentorship, guidance, and intellectual and material support. I would also like to thank Professor Panagiotis Michalatos and Professor Andrew Witt for their feedback and advice at several stages of the dissertation, each time challenging me to push this work a step further.

I would like to thank the Onassis Scholarships Foundation for their financial support (Scholarship ID: F ZO 002/1 – 2018/2019), without which the completion of this dissertation would not have been possible.

A special thanks goes to Matheus Gadelha for helping me navigate the machine learning landscape at an early stage of this work. I also thank all my friends and colleagues at Harvard Center for Green Buildings and Cities for creating a supportive and inspirational environment, and especially Saurabh for being my on-site tech support when I had to work remotely, Bin for always being available for energy-related questions, Ellie for sharing her experience on the final stages of the dissertation process, and Xu for going through the presentation with me before the defense.

To my parents, I wouldn't have been in the position to start this degree if it was not for your constant support, enabling every step that got me here. To my wife, Sophie, you are the reason I managed to pull this through; thank you for all you have done.

# Table of Contents

# 1. Introduction

During the building design process, an architect has to reconcile several qualitative and quantitative objectives. Performance-driven design aims to assist in meeting the quantifiable goals related to a building's performance. To maximize its impact, the performance-driven design methodology needs to be applied starting from the early design phase[1].

In performance-driven design, optimization has been the dominant paradigm to assist the search for the best-performing solutions in a predefined design space. Contrary to its original purpose as a precise problem-solving tool, optimization has been increasingly gaining traction as an exploratory tool in the early design phase (Bradner et al., 2014; L. Caldas, 2001; Evins, 2013; Mueller & Ochsendorf, 2015; Nagy et al., 2017; Z. Tian et al., 2018; Turrin et al., 2011). However, outside of the research field, the use of optimization in the early design phase has been limited for reasons that relate to i) the nature of parametric models, ii) time and resource intensity of simulations, iii) interpretability of results.

The design process can be seen as consisting of three operations:  formulation of design goals, production of alternative designs, and evaluation of their compliance with the defined goals

[1] Paulson and MacLeamy have both elaborated on the impact of changes along the different phases of design (Paulson Jr, 1976; Gertraud Breitkopf, et al., 2004). Similarly, Morbitzer argues that simulation should be used throughout the design process (Morbitzer, 2003).

(Carrara et al., 1991). These operations relate in a non-linear way and may be updated through multiple iterations. Notably, the problem definition is not fixed but co-evolves with the solution (Dorst & Cross, 2001). In particular, the early design phase is a divergent, exploratory process, characterized by quick iteration. On the other hand, optimization operates on a specific design parametrization, evaluates performance using simulation, and produces results that may need to be further analyzed.

First, even though parametric models are widely adopted in architecture, their applicability in the early design phase has been questioned. Parametric modeling is a process foreign to traditional design; it requires a significant cognitive investment in creating abstractions more akin to mathematical thinking or software engineering practice (Bradner et al., 2014). Creating parametric models for the purpose of design exploration can be particularly challenging (Brown & Mueller, 2019) for reasons related both to parameter selection and a model's expressive power. In fact, design optimization using a single all-encompassing model has been deemed problematic (Holzer et al., 2007), as parametric models can only afford limited changes before breaking (Davis, 2013). Various frameworks have been suggested to assist (D. Yang et al., 2017) or automate (Brown & Mueller, 2019) design parameter identification for performance-driven design exploration, and there is ongoing research toward automated parametric graph generation (J. E. Harding & Shepherd, 2017; Toulkeridou, 2019). Nevertheless, optimization is bounded by the parameters and variable ranges of a fixed parametric model fitted within a single design concept.

Another major limitation for applying optimization in architecture is the time intensity of the processes involved (Attia et al., 2013; Z. Tian et al., 2018; Wortmann et al., 2015; S.-H. E. Lin &

Gerber, 2014; Soares et al., 2017). Environmental or structural simulations can be computationally expensive. Combined with an optimization process that employs a stochastic search method, such as evolutionary algorithms, the calculation time increases by multiple orders of magnitude. In the early design phase, where it is essential to quickly consider multiple design alternatives, the slow optimization speed disrupts the exploratory process.

Finally, when interpreting multi-objective optimization results, architects can have difficulties understanding the solution space (Ashour, 2015). Optimization returns a set of high-performing solutions with corresponding performances; however, the connection between design parameters and performance tradeoffs is not always apparent (Wortmann & Schroepfer, 2019), offering little intuition to the designer. Furthermore, a large, unstructured set of solutions can be hard to understand and makes selection difficult. Multivariate data visualization (Wortmann, 2017b) and post-processing techniques such as clustering (Sileryte et al., 2016; Wortmann & Schroepfer, 2019) and dimensionality reduction (J. Harding, 2016; Sileryte et al., 2016; D. Yang et al., 2017) have been suggested to facilitate this task.

This thesis aims to provide a framework for early phase performance-driven design assistance through the interactive recommendation of optimal geometries. The objective is to enable a system that generates optimal geometry suggestions in a way that is:

i) responsive: the system should be able to interactively adapt to any changes introduced by the designer, and the interaction should be real-time,

ii) intuitive: both the problem setup process and the acquisition of the results should be supported in an intuitive way,

iii) comprehensible: the generated solutions should be presented and structured in a way that promotes an understanding of cause and effect between geometry form and performance.

This research introduces a novel method for interactively generating optimally performing geometries in real-time without the need for parametrization. A generative model learns the data distribution of optimal geometries for a range of problems and design spaces using a synthetic dataset. Optimal geometries for specific problems and with the desired performance characteristics can then be generated by navigating the learned latent space. The framework presented includes the preparation of a synthetic dataset, modeling its probability distribution using generative models, and sampling the learned distribution under given constraints.

Chapter 2 first reviews existing paradigms of performance-driven design assistance with an emphasis on the early design phase, then focuses on the uses of machine learning methods in performance-driven design. Building simulation, sensitivity analysis, and optimization are identified as progressively interdependent processes with an incremental degree of suggestive power. Uses of meta-models and machine learning in performance-driven design are discussed. Last, generative models in architectural design are reviewed.

Chapter 3 elaborates on the framework and methods presented in this work. First, it introduces generative models for performance-driven design. Then, it defines a specific design problem of building form optimization for passive solar gain in Boston, MA, as a case study that enables the detailed development of the framework’s several components.

Chapter 4 details the synthetic data generation process. It presents a series of bottom-up, granular, interactable geometric systems targeting the problem of optimal passive solar gain. It introduces the simulation environment and specifies the performance goals. It defines the range of boundary conditions and discusses the sampling strategies used. Last, it presents the different dataset versions produced and evaluates the final results.

Chapter 5 introduces a performance-aware generative model in the form of a 3d, multi-channel Variational Autoencoder. Loss functions for the various data types in the training set are presented, and performance-specific loss functions are introduced. Aspects of the model training are evaluated.

Chapter 6 leverages the trained VAE to generate optimal geometries or geometry completions for specific boundary conditions. A series of sampling techniques inspired by the data imputation literature is introduced. The structure of the latent space and the continuous and differentiable nature of the VAE-generative model is leveraged to create performance-oriented geometry modifiers.

Chapter 7 presents a prototype that implements the methods of Chapter 6 to demonstrate and test the abilities of a system based on the suggested framework.

Finally, Chapter 8 concludes the thesis by discussing the results of this work, identifying areas for future research, and enumerating the main contributions.

# 2. Literature Review

This chapter begins by introducing the three main paradigms in performance-driven design: simulation, sensitivity analysis, and optimization, and discusses their role in the early design phase by reviewing recent work in each area. Then, it introduces meta-models through research that demonstrates how they enhance and augment the three performance-driven design paradigms. Next, it discusses recent work that addresses performance-driven design in ways that relate to but do not strictly fit in the main paradigms above. Last, it introduces Generative Adversarial Networks (GANs) and briefly reviews their uses in performance-driven design and form generation.

## 2.1. Simulations

Simulations form the basis of performance-driven design. However, during the first decades of building simulation software development, they were only used to verify complete designs (Morbitzer, 2003), which meant little to no involvement in the design process. If a design was later found to be inefficient, there was little space for changes, as significant design changes would be cost and time prohibitive. Some of the early research on building simulation in the architectural community focused on the problems of data integration (Augenbroe, 1994), comprehensive simulation-based analysis (Mahdavi, 1996), and collaborative aspects of performance simulation (Primikiri & Malkawi, 2001). Nevertheless, up to 2011, only 40 out of a list of 392 Building Performance Simulation tools were found to be targeting architects during the early design phase (Attia et al., 2012).

### 2.1.1. Systematic Simulations

One early approach to assisting in the decision-making process using simulation is reflected in the work of Shaviv (1999). Simulations were employed systematically to evaluate several alternative designs concerning energy efficiency. The alternatives varied in insulation, color, window shading, night ventilation, ceiling vents, and sunspace on the upper roof. However, these performance simulations still took place at a later stage of the design process, and decisions were limited to the envelope of a pre-determined building form.

### 2.1.2. Template Based Tools

Soon after, Morbitzer (2003) argued that simulation should be used throughout the whole design process, from the outline stage to the detailed design. The speed of model creation and simulation execution were identified as limiting factors preventing the acceptance of performance simulations by the designers. To overcome this problem, Morbitzer suggested a system that used benchmarks and pre-computed databases to evaluate designs quickly.

Focusing on the exploration of multiple and diverse alternatives during the initial design stage of facades for office buildings, Ochoa and Capeluto (2009) introduced NewFacades. This tool suggests energy-conscious designs based on prescriptive codes. The authors implemented a GUI through which the user selects the specifications from presets, and the system returns several alternatives with their evaluations.

Similarly, Attia et al. (2012) proposed a prototype tool to assist decision-making for designing net-zero energy buildings. The tool generates and evaluates alternatives and also performs sensitivity analysis. It heavily relies on templates for geometry generation, which severely limits its flexibility.

### 2.1.3. Real-Time Simulations

As parametric tools became popular, simulation was used to assist with the best parameter selection. However, the relatively slow simulation speed for performance evaluation combined with reasonably large design spaces, as defined by the parametric models, has been constraining this process. To overcome the issue of speed, Chronis et al. (2012) developed a case-specific tool for use at Foster and Partners that enables real-time exploration of incident solar radiation and daylight results through a database of pre-calculated 'key test cases'.

Other work has focused on speeding up the simulations themselves. Jones and Reinhart presented work that significantly speeds up the Global Illumination calculation inside the popular 3d modeling environment Rhinoceros-Grasshopper (Jones & Reinhart, 2014). They implemented the analyses using parallel logic, taking advantage of the increased capabilities of modern graphics hardware. In this way, they provided real-time feedback on a pre-defined parametric model's energy and structure performance to designers and showed that real-time feedback leads to higher performance designs (Jones & Reinhart, 2016).

In a more recent study, N. Brown (2020) used surrogate models to provide real-time results regarding the energy and structure performance of a pre-defined parametric model, receiving positive feedback from the participants.

## 2.2. Sensitivity Analysis

### 2.2.1. Parametric Model Simplification

Sensitivity analysis is a statistical method that has been used in Building Performance Simulation (BPS) to identify the most influential parameters to a building's performance (Lam & Hui, 1996). By indicating the size and direction of performance change for variations to the input, this method helps guide the design decision process. It directs attention toward specific parameters and enables the architect to explore the design space more efficiently (Hopfe et al., 2007; Lam & Hui, 1996).  In sensitivity analysis, a large number of value combinations are sampled for the free parameters, and the respective performances are acquired. This process can be time-consuming for complex simulations, raising the need for meta-models in place of the regular BPS models (W. Tian, 2013). Sensitivity analysis has been used extensively for building construction and operation parameter exploration, as indicated in the extensive list of examples in W. Tian (2013).

Recent research has focused on the efficiency of different sensitivity analysis techniques for building performance (Gagnon et al., 2018) or new techniques that make the method more suitable to specific energy-related problems. For example, Østergård et al. (2015) suggested an iterative parametric method for a holistic simulation approach, identifying favorable parameter regions for energy, thermal comfort, and daylight. The same team introduced interactive sensitivity methods that rank inputs with respect to energy performance, thermal comfort, and daylight while highlighting the most influential parameters (Østergård et al., 2017)

### 2.2.2. Design Space Reformulation

While sensitivity analysis is typically used to identify the significance of parameters, it has also been used to evaluate the performance potential of alternative parametrizations.

When setting up a parametric model, it is essential to allow for the desired variability in the resulting designs. On the one hand, the design space needs to be broad enough to include areas that conceptually and aesthetically align with the designer's intent and to increase the probability of containing high-performance areas. On the other hand, it needs to be tight enough to make its navigation easier, especially given the computational intensity of the simulations. Therefore, evaluating the potential of different parametric models early on, before a detailed exploration of the design space, can be of significant importance.

Sileryte et al. (2016) proposed a sensitivity analysis tool that integrates with Rhinoceros-Grasshopper (Rutten, 2007) to assess the suitability of a parametric model before the search for the optimal solutions begins. They used a Self-Organizing Map (SOM) and Hierarchical Clustering (HC) along with custom visualization tools to evaluate the results.

Extending this work, Yang et al. (2017) presented research on design exploration as a means towards the formulation of a good design concept. A good design concept, formalized as a design parametrization, is one that can lead to high-performance designs while complying with the qualitative criteria set by the designer. They integrated Rhinoceros-Grasshopper with ModeFRONTIER (*ModeFRONTIER*, 2017) software to perform sensitivity analysis and analyzed the input-output relationships using a SOM and HC. They presented a case study where three different conceptual designs for a skylight were compared by evaluating the energy and daylight objectives using their respective parametrized models. Through the analysis, the team identified promising variables that gave birth to a new, fourth design concept, which finally led to higher-performance designs than any of the three initial models.

In further development, Yang et al. (2018) proposed a Computational Design Exploration approach that aims to support concept reformulation through knowledge extraction from the process of sensitivity analysis. They presented a case study on a sports building.

## 2.3. Optimization

### 2.3.1. Algorithms and Applications

Optimization techniques for performance-driven design can be classified into three main categories: i) analytical methods, ii) direct search, and iii) meta-heuristics. Analytical methods have only found limited applications in this field in recent years because of their strict requirements for continuous and differentiable objective functions and simplifications that typically need to happen both to the geometry and the performance simulation models (Evins, 2013; Radford, 1988).

Direct search methods do not require gradients, and some of them can be used even when small discontinuities are present in the objective function. While direct search is the second most popular optimization method for sustainable building design (Evins, 2013), it has mostly been applied to single-objective problems of a predominantly engineering nature. Direct search algorithms have been assumed to be inadequate for complex problems with discontinuities in the performance (Wetter & Wright, 2004). However, this claim has been recently disputed (Wortmann et al., 2017). For an extensive review of applications of direct search methods, we direct the interested reader to (Kheiri, 2018).

The most popular optimization methods for performance-driven design in the last two decades belong to the category of metaheuristics, with evolutionary processes and, more specifically, Genetic Algorithms (GA) taking first place (Evins, 2013; Kheiri, 2018; Nguyen et al., 2014). The genetic algorithm NSGA-II (Deb et al., 2002) is the most common implementation for multi-objective problems (Evins, 2013). It has also been the default solver with popular parametric modeling software such as Dynamo (*Dynamo*, 2012/2022). Research on better optimization tools for performance-driven design is still ongoing. Most recently, Cubukcuoglu et al. (2019) introduced Optimus, a new optimization tool for Grasshopper using a self-adaptive differential evolution algorithm with an ensemble of mutation strategies (jEDE). They showed that Optimus provided better solutions than existing tools for several design problems.

Metaheuristics, such as genetic algorithms, are often inspired by natural processes. Like the direct search methods, they require no knowledge and pose no limitations to the objective function properties. They can handle both continuous and discrete variables, are suitable for non-linear objective functions, and work well with multiple objectives (Evins, 2013; Nguyen et al., 2014). Metaheuristics are often used for building shape optimization. The geometry generation is typically achieved using parametric models, although different approaches such as shape grammars and bottom-up systems have also been researched (L. G. ; S. Caldas, 2012; Evins, 2013; Pantazis & Gerber, 2018).

One of the first uses of a genetic algorithm (GA) for building performance optimization appears in the work of Caldas (2001). Caldas argues for design improvement and not strictly optimization since her goal was to find a population of good designs and not one best design, something that the GA enables. Caldas used a GA in a fenestration study, then extended her approach to a three-dimensional parametric model of a building made of boxes that could change size and roof taper angles.

Yi and Malkawi (2009) coupled Energy Simulation with Computational Fluid Dynamics (CFD) to optimize the form of a building  using a GA. To introduce geometric flexibility and allow more complex geometries, Yi (2008) introduced a hierarchical geometry representation.

Lin and Gerber (2014) identified one of the major problems preventing the adoption of multi-objective optimization outside of the research field to be the inability of existing systems to deal with the necessary geometric complexity. They developed a framework (EEPFD) and a tool that integrates modeling and simulation software and uses a GA to support the designers' decision-making early in the design process. The framework, which uses Revit (*Autodesk Revit*, 2000) Conceptual Masses energy analysis, optimizes for three objectives: spatial program, energy performance, and financial performance. It provides a tradeoff analysis, and the authors were planning to include sensitivity analysis as an initial step.

As the awareness and interest in automated methods and performance-based design increased, researchers have tried to make optimization methods more approachable to the designers' community. Shi and Yang (2013) reviewed existing tools and suggested an architect-friendly building performance optimization workflow that they accompany with case studies. Ashour (2015) identified a problem in communicating optimization results to the designer, suggested a workflow, and contributed a visualization tool in Grasshopper for the efficient exploration of the solution space. Konis et al. (2016) focused on the accessibility of design optimization and suggested a workflow in Grasshopper.

Wortmann et al. (2017) argued that the primary reasons for the widespread adoption of metaheuristics in building performance optimization have been the availability of the tools and dubious assumptions about the algorithms' nature, with their actual effectiveness often being overlooked. They compared common black-box methods, including several direct-search, metaheuristics, and surrogate-based algorithms, benchmarking speed of convergence and stability.

### 2.3.2. Non-quantifiable objectives

One concern for architects using multi-objective optimization processes is that there is no easy way to incorporate their expertise or subjective preferences as part of the objective function.

Caldas (2001) argues that a generative system based on optimization would be best used as an augmented design tool in an interactive loop with the architect-designer. This interaction is necessary to enable the consideration of non-quantifiable issues during the optimization process. Many researchers have worked on integrating the designer's expertise with the improved performance-based guidance that optimization offers. Evolutionary optimization is regarded as particularly suitable for use in this interactive setting due to its iterative nature and the progressive generation of higher-performing 'populations'.

For this purpose, Turrin et al. (2011) introduced ParaGen, a design tool that uses a GA for performance-driven design space exploration. ParaGen can optimize a parametric model from start to finish using the implemented algorithms or be used in an interactive setting. The designer can intervene after each cycle, choosing parents for the breeding phase or even selecting a single parent for mutation and further evolution.

Mueller and Ochsendorf (2015) presented an interactive evolutionary optimization method. They developed a proof of concept tool called structureFIT (Mueller, 2013) to combine quantitative structural performance with conceptual design intent, focusing on balancing formulated and unformulated objectives.

Tools developed for multi-objective optimization in Grasshopper, such as Octopus (Vierlinger, 2012), also enable interactive optimization by allowing the designers' input between generations.

### 2.3.3. Exploration of multi-objective optimization results

Multi-objective optimization gives a designer the flexibility to choose between solutions, prioritize different goals, and integrate non-quantifiable goals during this selection. However, the process is not always straightforward. A high dimensional parameter space and a high dimensional objective space make understanding the results challenging.

A commonly used solution is to apply clustering, such as K-means. With clustering, the high number of solutions is structured within categories from an often predetermined number of groups. One sample from each group can be chosen as a representative example of the group contents. Then the designer can evaluate whole groups based on their representatives (Wortmann & Schroepfer, 2019). Sileryte et al. (2016) suggested hierarchical clustering to explore design alternatives effectively. They demonstrated its use in a case study of a problem with 18 variables and 4 performance objectives related to energy utilization and construction cost.

Another way to understand the optimization results is to use data visualization to reveal the relationship between parameter changes and performance. Parallel coordinates is a visualization technique that has been employed for high-dimensional objective spaces (Sileryte et al., 2016; Wortmann & Schroepfer, 2019), where each objective is represented on one of the parallel axes. While there is technically no limit on how many dimensions can be accommodated, the visualizations become harder to understand as the number increases. Radial and star coordinate systems use a similar technique, but the axes are arranged radially. Star coordinates are more intuitive, as the use of a single point contains information about the performance in all axes (Wortmann & Schroepfer, 2019), A different approach is using dimensionality reduction to map a high dimensional performance space to two or three dimensions. A popular technique for this is the Self Organizing Map (SOM) (J. Harding, 2016; Sileryte et al., 2016; D. Yang et al., 2017).

### 2.3.4. Commercial Tools

Various tools that support optimization based on building performance simulation are used in the industry. Direct search algorithms are typically available for single-objective optimization problems. For example, generalized pattern search, discrete Armijo gradient, and a simplex of Nelder and Mead are available in GenOpt, linear programming, quadratic programming, integer programming, least squares, and Hooke-Jeeves algorithm through MATLAB directly or through GUIs such as Topgui (Attia et al., 2013). For multi-objective optimization problems, Particle-Swarm optimization, evolutionary algorithms, or genetic algorithms (NSGA-II) are widespread (Attia et al., 2013).

A similar pattern is identified by examining the available methods in ANSYS Design Explorer (DX). Direct search or gradient-based algorithms are implemented as single-objective optimizers (e.g., Non-Linear Programming by Quadratic Lagrangian). In contrast, genetic algorithms are available for multi-objective problems (e.g., Multi-Objective Genetic Algorithm) (*Design Exploration User's Guide*, 2013).

## 2.4. Meta-models

A meta-model is a simplified model of a model (Østergård et al., 2016). In BPS, meta-models, or surrogate models, can be used in the place of actual simulation models by approximating a mapping between inputs and outputs parameters.

### 2.4.1. Surrogate Models in Performance Simulation

Meta-models in performance-driven design are often used to reduce the computational cost of simulations. Such a reduction can be of significant importance in workflows involving sensitivity analysis or optimization. These processes may require hundreds to thousands of simulations, each taking several minutes to complete (Nguyen et al., 2014). Kumar et al. (2013) provide an early overview of the use of Neural Networks for building energy use prediction. Tseranidis et al. (2016) give an overview and a comparison of common data-driven models for performance estimation in civil structures, while Westermann and Evins (2019) provide a comprehensive review on surrogate models in sustainable building design up to 2018.

Much recent work on surrogate modeling for BPS has focused on artificial neural networks (ANN) and deep learning methods. Lorenz et al. (2018) and Radziszewski & Waczy (2018) trained ANNs to substitute daylight simulations for simple rectangular building geometries with parametrized windows. Alammar et al. (2021) compared two methods of substituting incident solar radiation simulations using ANN and decision trees for box-shaped buildings in various surrounding contexts. Westermann and Evins (2021) used Bayesian deep learning to incorporate uncertainties in a surrogate model. Westermann et al. (2021) released a web-based app that uses surrogate models to estimate cooling and heating loads for two building types with variable location, construction, and orientation parameters. Mokhtar et al. (2020) trained a conditional adversarial network to approximate pedestrian wind flow. Han et al. (2021) presented a generalizable method for incident solar radiation prediction using geometry voxelization and 3d convolutional neural networks (CNNs). Surrogate models are dependent on a pre-determined parametrization of the design space and the sampling range of the training dataset. Therefore, concerns about generalizability and re-use of surrogate models have recently motivated research in transfer learning (Pinto et al., 2022; Whalen & Mueller, 2022).

### 2.4.2. Surrogate Models in Optimization

Surrogate models are often used during optimization to replace costly functions, contributing to significant time savings. There are two ways for their application: i) static, where the surrogate model is first trained to approximate the simulation and then used in its place, and ii) adaptive, where the surrogate model is re-trained in between optimization iterations to better fit in the areas of interest (Westermann & Evins, 2019).

Examples of the static approach are workflows where an NSGA-II algorithm was used for optimization in conjunction with surrogate models that approximated thermal and energy simulations (Bre et al., 2020; Gossard et al., 2013).

An example of the dynamic approach is described by Sileryte et al. (2016) as a Responsive Surface Methodology-based surrogate model that is iteratively retrained during the optimization. Another example is the RBFOpt algorithm (Costa & Nannicini, 2018), which incorporates radial basis functions for approximating the objective evaluation with an optimization algorithm for effectively searching the design space. The RBFOpt algorithm was introduced to Grasshopper by Wortmann (2017a) through the Opossum plugin.

### 2.4.3. Classification of Design Space

In a different use of metamodels, researchers have tried to predict a classification of the designs' performance instead of an exact performance metric. An ANN has been used to learn a binary classification of the design space and guide the design exploration towards the more promising areas (Dabbeeru & Mukerjee, 2008). Similarly, other work applied a multi-class classification of the design space in a chair design problem, identifying high-performing areas in terms of the ergonomics (Reed, 2016).

#### 2.4.4. Meta-models for subjective evaluation

Advances in Machine Learning (ML) and the widespread availability of sophisticated but easy-to-use ML tools have pushed the boundaries of what can be approximated with meta-models. Phelan et al. (2017) used an ANN to assist with the meeting room allocations at WeWork. The neural network predicted the meeting room usage better than the company's previously used methods. Sjoberg et al. (2017) trained an ANN to learn an individual designer's preferences concerning the form for geometries produced using a parametric model. The ANN learns to score building designs following a subjective rating of several examples compiled into a dataset. Then, the ANN's predictions become part of the objective function used to optimize the parametrized geometry, in conjunction with more typical, simulation-based performance evaluations. With a similar goal, Musil and Wilkinson (2016) trained an ANN to predict user preferences on simple geometries, to automate solution selection between iterations of an optimization process.

## 2.5. Other Methods

Some research has tried to address performance-driven design from perspectives outside of the three main categories of direct simulation, sensitivity analysis, and optimization and different or sometimes tangential to their metamodel-enhanced variations.

Brown and Mueller (2017) focused on the problem of parametrizing a structure for enhancing its performance. They used dimensionality reduction to capture complex relationships between "dummy" parameters and project them to a low-dimensional space, which is easier for the designer to explore. They used principal component analysis (PCA) and canonical correlation analysis (CCA) to replace the original parameters with a small number of more “meaningful” ones from a performance perspective.

Conti and Kaijima (2018) suggested a Bayesian Network that performs bidirectional inference on parametric structural models. The Bayesian Network enables the execution of inverse inference, which means that given an objective, the most probable parameter ranges of the design model can be predicted.

Aksöz and Preisinger (2020) replaced runtime optimization with an ANN trained to predict the optimal angle for bracing elements of metal structures. Focusing the prediction analysis on a single, parametrized element modularized the whole system, achieving generalizability for any topology within the target structural system.

Chang and Cheng (2020) trained a pair of ANNs, NeuralSim and NeuralSizer, for the cross-section sizing of structural designs. NeuralSim is a surrogate model for structural simulations. After its training, it was used in the loss function of NeuralSizer, due to its differentiable nature, which enables the computation of gradients.

Ampanavos et al. (2021) demonstrated the use of a CNN for approximating structural layouts from sketch-level building plans. When trained on an appropriate dataset, the method has the potential to produce optimal solutions within an environment that is well-suited for the early design phase.

## 2.6. Machine Learning Generative Models

A generative model is an ML model that can learn an estimate of a distribution by observing a set of examples from that distribution, i.e., a training set  (Goodfellow, 2017). Once fully trained, sampling a generative model approximates sampling from the original data distribution. For example, a generative model trained on a dataset of faces will generate new faces when sampled[2].

Some generative models work by learning a mapping of the original data to a lower-dimensional space, called the latent space. For example, the Variational Autoencoder (Kingma & Welling, 2014) (VAE) explicitly learns an encoder and a decoder function that maps the original data to and from a latent space. Naturally, similar data points will be located nearby within the latent space. This characteristic allows for smooth interpolation of data samples by traversing the latent space or even for the composition of new data with specified properties through latent space vector arithmetic, as demonstrated by Wu et al. (2016) in the domain of three-dimensional objects.

In architecture, several attempts have been made to use generative models in the creative phase (Huang & Zheng, 2018; H. Liu et al., 2019; Mohammad, 2019; Zhang & Blasetti, 2020). Most such works used Generative Adversarial Networks (Goodfellow et al., 2014) (GANs), motivated by some impressive results in the field of computer vision (Brock et al., 2019; Goodfellow et al., 2014; Isola et al., 2017; Karras et al., 2018). Some research related to architectural form has used GANs for morphological studies mixing 2D (Chaillou, 2020; Del Campo, 2019; Mohammad, 2019) or 3D (Zhang & Blasetti, 2020) plans, geometries, and urban morphologies (Fedorova, 2021). Other

[2] See for example the Progressive GAN model (Karras et al., 2018) trained on the CelebA dataset (Z. Liu et al., 2015).

work has suggested early-design assistance, introducing novel GANs that generate 2D (Nauata et al., 2020) or 3D (Chang et al., 2021) geometries from space connectivity graphs as compositions of orthogonal forms. Image-to-image translation GANs were used to approximate wind flow (Mokhtar et al., 2020), to populate plans with mechanical systems (Sato et al., 2020), and to substitute topology optimization results (Bernhard et al., 2021), among others. De Miguel (2019) used a VAE to model a distribution of structural design typologies. Danhaive and Mueller (2021) suggested a performance-conditioned VAE to reduce the dimensionality of a structural design's parametric model and assist the performance-driven design space exploration.

## 2.7. Chapter Conclusions

Building performance simulation, sensitivity analysis, and optimization have been architects' major, essential tools in performance-driven design. Shortcomings in simulation speed have been addressed through surrogate modeling (Sileryte et al., 2016; Tseranidis et al., 2016; Westermann & Evins, 2019; Wortmann et al., 2015; D. Yang et al., 2016). Design agency in optimization has been addressed directly using interactive optimization (Mueller, 2013; Mueller & Ochsendorf, 2015; Turrin et al., 2011) and indirectly by using ML to quantify and generalize subjective evaluations (H. Liu et al., 2019; Musil & Wilkinson, 2016; Sjoberg et al., 2017). Design exploration has been addressed through multi-variate visualizations and dimensionality reduction for data visualization or model re-parametrization (Brown & Mueller, 2019; Danhaive & Mueller, 2021; J. Harding, 2016; Sileryte et al., 2016; Wortmann, 2017b; D. Yang et al., 2017). Such work has focused on design with pre-determined parametric models that express a single design concept. Some work addressing simulation has used more flexible geometry representations in the form of pixels or voxels for surrogate modeling (Bernhard et al., 2021; Han et al., 2021; Mokhtar et al., 2020), but this flexibility is not trivially generalizable to optimization or other form-guiding processes. Last, several researchers have leveraged ML generative models to suggest novel techniques for architectural form generation, but that work has not been connected to building performance.

# 3. Method

## 3.1. Framework

This thesis introduces an ML-based framework for performance-driven design assistance through the interactive suggestion of optimal geometries. Previous work on performance-driven design has focused on facilitating the exploration of a well-defined design space for a single problem or simulation context. In contrast, a system based on the suggested framework uses a generalizable geometry representation that makes no assumptions about the design concept or topology and can be used for a wide range of problem definitions, i.e., simulation contexts.

During a typical optimization process, a design space is first defined using a parametric model. To find the best-performing solutions, samples from the design space are mapped to their corresponding performances using simulation. The search for the best-performing solutions advances through iterative performance evaluations coordinated by a stochastic optimization algorithm. The constructed relationships between optimization parameters and performance are often visualized using a fitness landscape[3]. Figure 3.1 shows an example of a fitness landscape for a geometry optimization problem, where the design space is encoded by two geometric parameters. In this example, multiple local optima have been identified (blue dots) and comprise the set of optimal solutions.

[3] General background on the term and use of fitness landscapes can be found in (Reeves, 2005) and (Rutten, 2014).

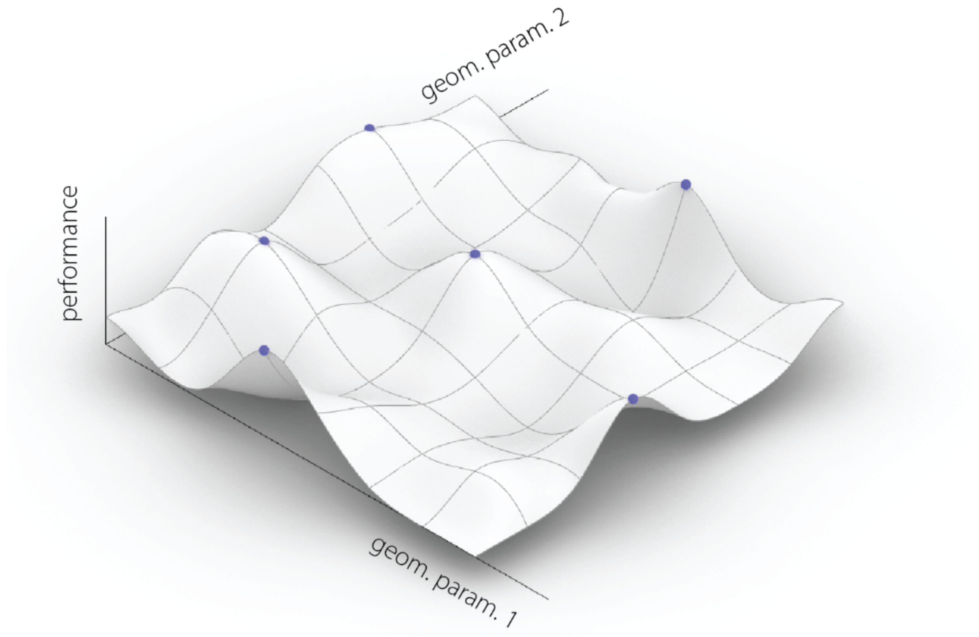


*Figure 3.1 Example of a fitness landscape for a geometry optimization problem. The best solutions are marked in blue.*

This thesis aims to replace the optimization process during design by shifting the search from the design space to a well-structured, continuous, performance-aware, compressed space that encodes the boundary conditions and optimal geometric solutions for a family of problems. Figure 3.2 shows an example of such a compressed space with two dimensions. Four problems occupy distinct areas in the compressed space. Optimal solutions for a problem are generated by sampling the corresponding areas, then mapping the samples to the geometry space. In addition, the structure and the continuous nature of this space enable goal-driven geometry tuning and exploration, by following appropriate trajectories in the compressed space, such as p or ∇v in Figure 3.2.

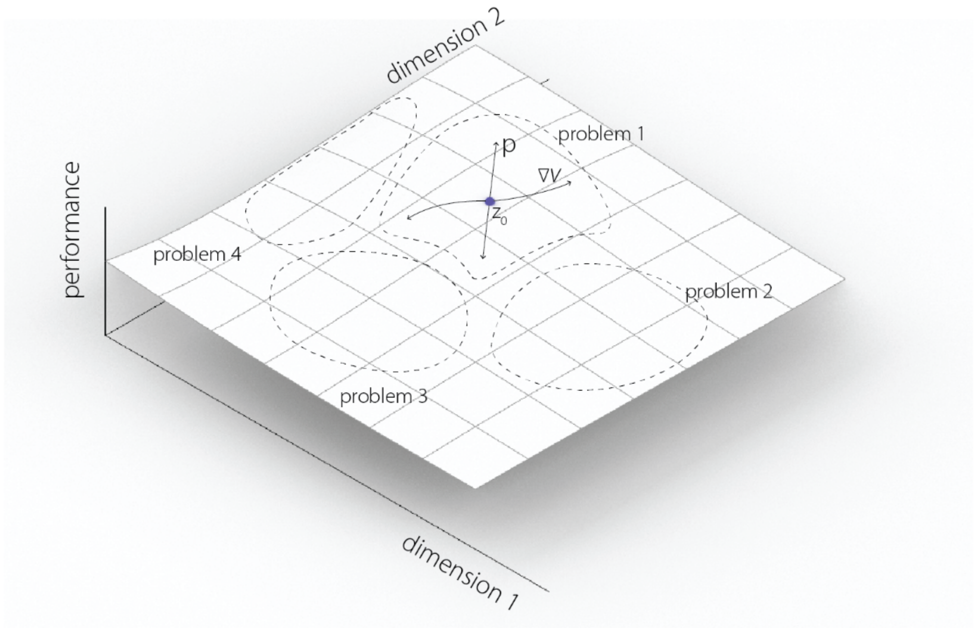


*Figure 3.2 Example of a compressed space encoding problem boundary conditions and their optimal geometric solutions. Vector p encodes performance changes. The gradient $\nabla v$* creates a trajectory of monotonic volume changes.

This thesis suggests that this compressed space can be constructed as the latent space of a generative model. In particular, a Variational Autoencoder (VAE) is used to learn a mapping E(x) between the joint space of problem definitions and optimal geometric solutions and the model's latent space, as well as the inverse mapping D(z), as shown in Figure 3.3.

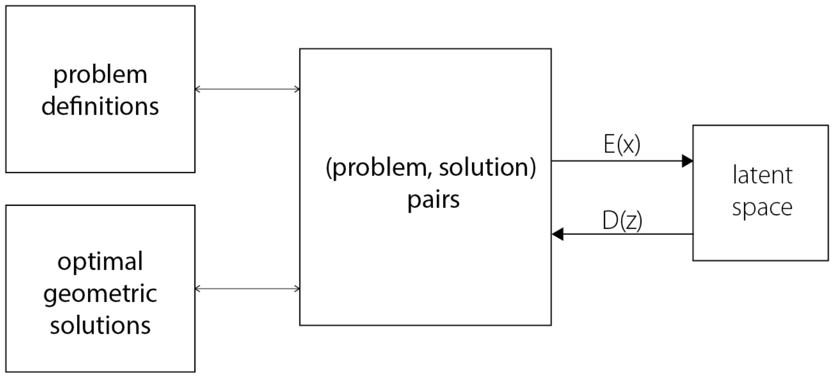


*Figure 3.3 Encoding and decoding problem definitions and optimal solutions to and from a latent space.*

The overall framework involves two phases, detailed in Figure 3.4. The first phase refers to the offline preparation of the generative model which is used during design time. It only needs to happen once for a family of problems.  It includes the dataset generation and model training. The dataset is comprised of synthetic data; first, a number of problem instances are sampled from a range of boundary conditions, then a set of geometries is generated for each boundary condition using an appropriate geometry representation and a conventional optimization process. During model training, a performance-aware VAE is used to learn the probability distribution of boundary conditions-optimal geometries pairs.

The second phase refers to the use of the model for design assistance and includes methods for inference and exploration of optimal geometric solutions. During inference, the geometry generation task is framed as a data imputation problem, where the boundary condition and the optimal geometry are the observed and missing data, respectively. The missing data are generated using the trained VAE. Then, performance-driven exploration is achieved by navigating the VAE's latent space. Differentiable geometry attribute approximations and vector arithmetics in the latent space enable a fast, goal-driven tuning and exploration of the geometry solutions. Figure 3.2, for example, shows how a sample $z_0$ that encodes a solution for a building performance problem can be modified using the performance-specific vector p to explore different performing geometries. Similarly, the size of an obtained solution can be tuned by following the volume gradient $\nabla v$.

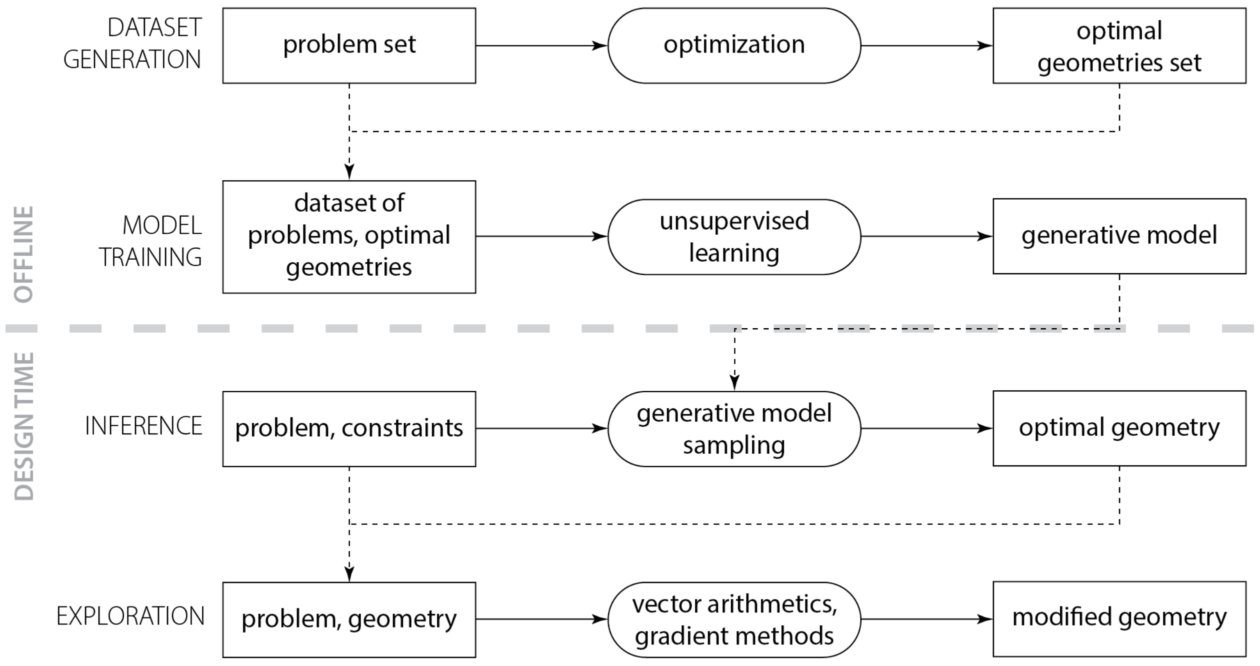


*Figure 3.4 The framework for performance-driven geometry generation and exploration.*

## 3.2. Problem Scope

A case study is presented to allow further development of the framework and evaluate the effectiveness of the several methods and overall feasibility.

In a typical scenario for designing a new building, the designer would have information including the location, the plot shape and size, the surrounding buildings, and the program of the building. In performance-driven design, maximizing the performance of interest is of primary concern. Then, in the early design phase, where the focus is on form finding, the problem would be expressed as finding a geometry for the building that maximizes the desired performance, given the simulation context (Figure 3.5).

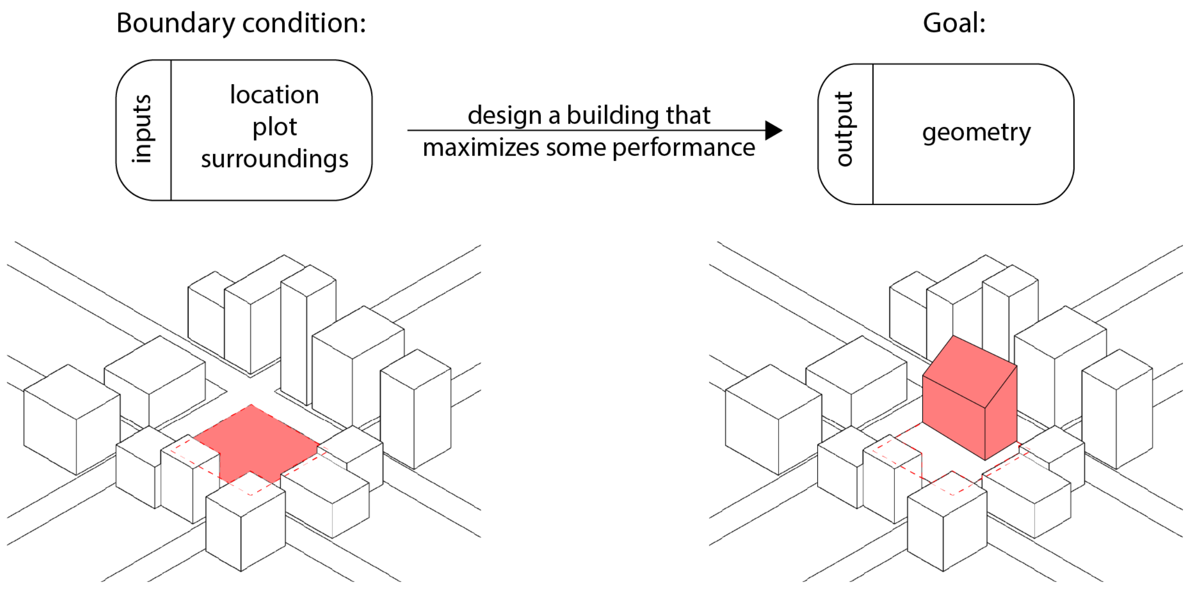


*Figure 3.5 . Optimization: context and expectation for the design of a new building.*

In this work, the goal is to generate a geometric form for a building that optimizes the passive solar gain throughout the year for the location of Boston, MA. The area coverage and the building volume of the geometries are considered parameters that need to be controllable, reflecting constraints from the building code and the architectural program.

Figure 3.6 outlines the process followed. First, to generate the dataset, various geometric systems were explored, examining their ability to produce geometries with the desired morphological characteristics and performance when used with an optimization process. In addition, constraint sampling was reformulated as an optimization objective, increasing the dimensionality but decreasing the total number of optimization problems to be solved. Last, a grid-based system was used to sample various building sites.

Then, a generative model was trained using the problem definitions and optimization results of the synthetic dataset. This step introduced a novel loss function for a performance-aware VAE that improved the model’s ability to reconstruct the geometry data with accurate building performance.

Next, a set of methods for constrained sampling of the generative model was introduced and compared to determine the best candidates to enable design assistance. Likewise, a set of methods for performance-driven geometry exploration was introduced and compared.

Last, the trained model and the selected methods were incorporated into a prototype tool for 3d-modeling.

It is worth noting that the process is not executed in a linear way, but it includes several loops, where results at several stages cause revisions to prior components in an iterative way. These reverse connections are indicated with the red arrows in Figure 3.6.

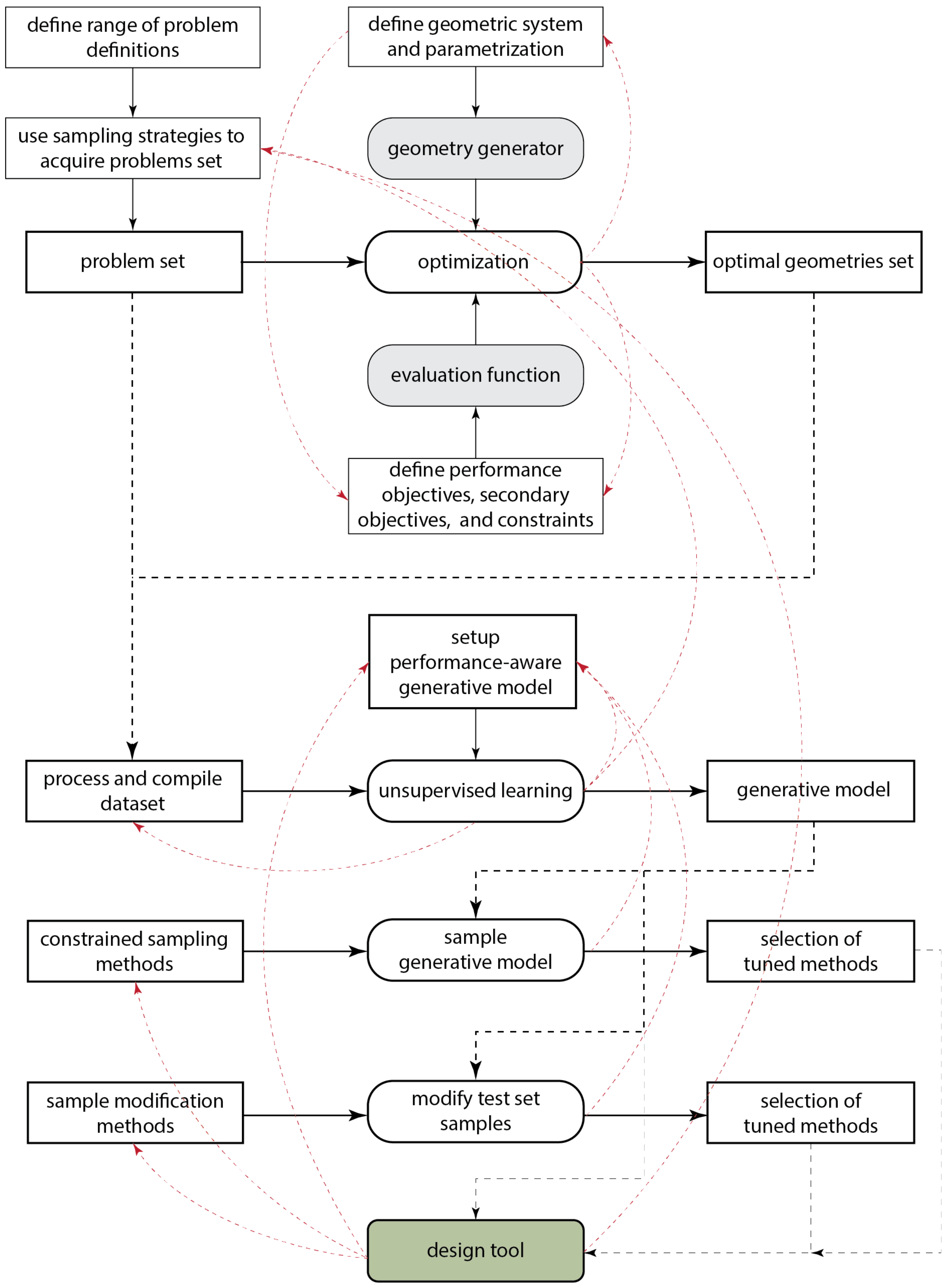

define range of problem definitions
define geometric system and parametrization
use sampling strategies to acquire problems set
geometry generator
problem set
optimization
optimal geometries set
evaluation function
define performance objectives, secondary objectives, and constraints
setup performance-aware generative model
process and compile dataset
unsupervised learning
generative model
constrained sampling methods
sample generative model
selection of tuned methods
sample modification methods
modify test set samples
selection of tuned methods
design tool

*Figure 3.6 Process outline.*

# 4. Optimal Geometry Generation

The first step of the suggested framework involves the preparation of a dataset that will be used to model the probability distribution of optimal geometries for the selected range of boundary conditions. A sufficiently large set of diverse building geometries with the desired performance is needed, for a sufficiently dense sampling of the range of the possible contexts. However, openly available building data are very limited, do not typically include performance assessment and if they do, the nature of the performance metrics and differences in methods used do not allow for direct comparisons. In addition, they usually do not come with three-dimensional models of the buildings. Therefore, the limited availability and wide inconsistencies in geometric representations and performance assessments make it impossible to compile such a dataset using data collection methods.

As a result, a synthetic dataset needs to be generated. The synthetic dataset is comprised of a set of problem definitions (boundary conditions) and the corresponding set of optimal geometries for the defined performance metrics. The overall process of the dataset generation is outlined in Figure 4.1. The three essential components are highlighted in yellow, blue, and red. At the core of the process is a geometry generator, comprised of a flexible, performance appropriate geometric system, and a geometry parametrization that interfaces with the optimizer (Figure 4.2 Left). The evaluation function includes a set of performance, geometric, and sampling objectives and constraints and utilizes an appropriate simulator (Figure 4.2 Center). The compilation of the problem set requires the definition of the simulation boundary conditions range, as well as a sampling strategy for selecting individuals from that range (Figure 4.2 Right).

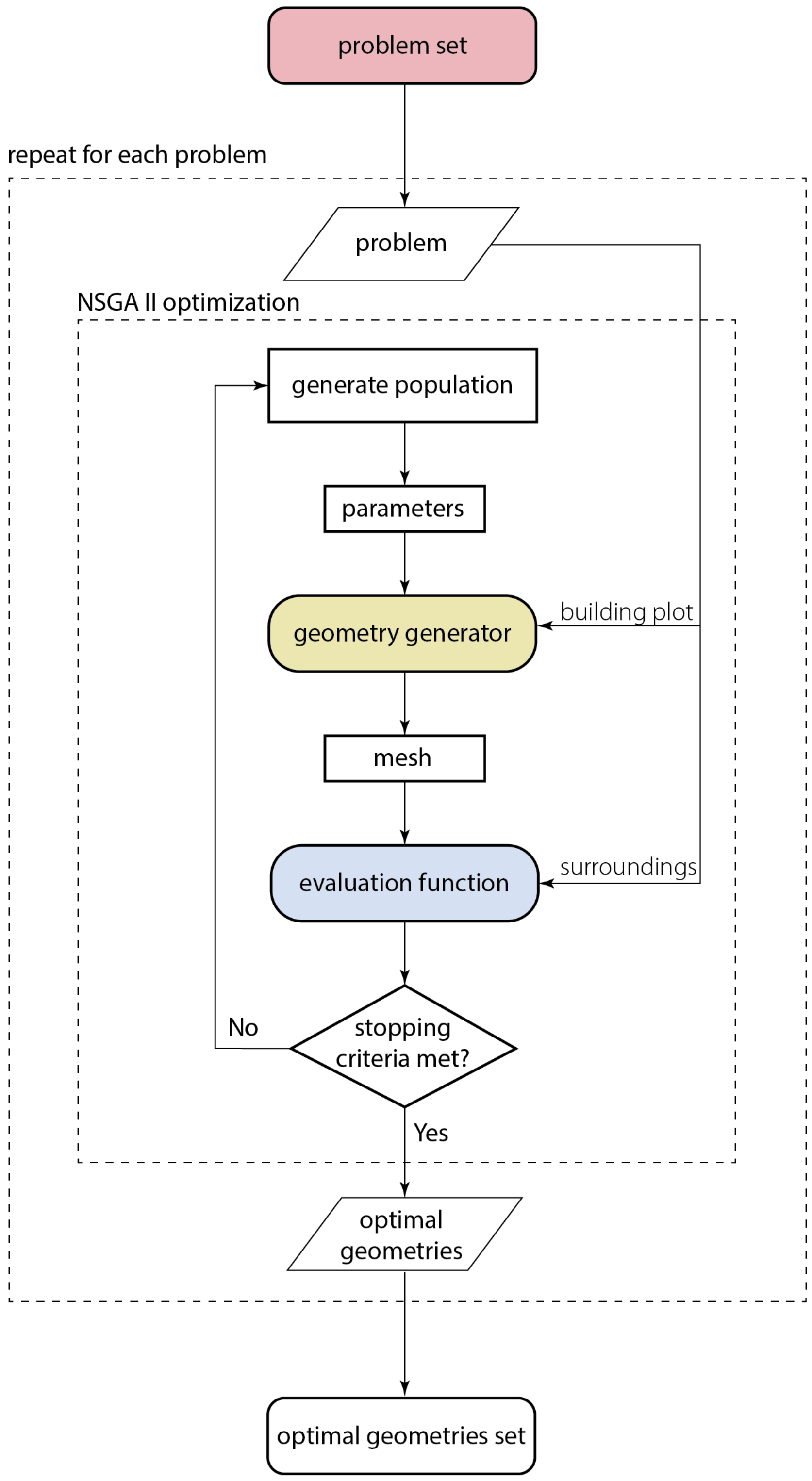


*Figure 4.1 The synthetic dataset generation process.*

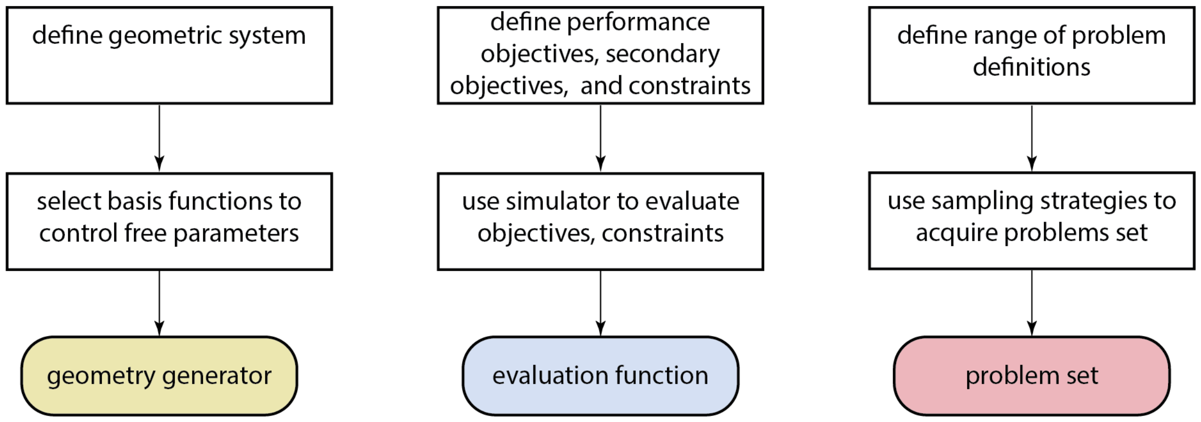

define geometric system
select basis functions to control free parameters
geometry generator
define performance objectives, secondary objectives, and constraints
use simulator to evaluate objectives, constraints
evaluation function
define range of problem definitions
use sampling strategies to acquire problems set
problem set

*Figure 4.2 Defining the three essential components for the dataset generation.*

This chapter first introduces the performance objectives and the simulations that enable their evaluation. Then it describes the development of the geometry generator and compares different geometric systems. Next, it introduces a series of geometric optimization objectives that complement the geometric system. In addition, a set of sampling-related objectives are defined as a substitute for geometric constraints, reducing the total number of optimization problems. The next section describes the range and sampling of boundary conditions, i.e. plot shapes and surrounding obstructions. Four different dataset versions considered in this work are described. Last, the performance of the resulting geometries is evaluated against baselines.

## 4.1. Performance Objective

### 4.1.1. Goals

In this case study, the goal is to generate geometric forms for buildings that optimize solar gain throughout the year. More specifically, solar gain is considered beneficial during the heating season since it has the potential to contribute to the building’s heating and reduce the usage of heating systems. In contrast, during the cooling season, solar gain is considered harmful because it can increase the energy demand for cooling. Therefore, to optimize solar gain, the incident radiation should be maximized during the cooling season and minimized during the heating season.

### 4.1.2. Metrics

Solar gain directly depends on the amount of solar radiation that a building receives. By definition, the total incident radiation increases together with the building envelope and by extension, with the size of the building. In architectural design, the size of a building is typically given in the form of the architectural program and should not be modified significantly as a means to meet performance goals. Therefore, in order to remove the building size as a parameter from the search for optimally performing geometries, the mean incident radiation intensity is used instead of the total incident radiation as the performance metric of interest.

### 4.1.3. Performance objective formulation

The optimization objectives $Jcool_{min}$ and $Jheat_{max}$ for the cooling and the heating period, respectively, are described in equations ( 4.1 ) and ( 4.2 ), where $I_h(face_i, geometry, obstructions)$ is the incident radiation intensity on the *i*-th mesh face, shaded by the *geometry* and *obstructions* during the hour *h*. The *geometry* is a mesh with $n_{faces}$ faces, where $FaceArea_i$ is the area of the *i*-th face. The *obstructions* are the immediate surrounding buildings, which shade the area of the building site.

The maximization objective $Jheat_{max}$ can be converted to an equivalent minimization objective $Jcool_{min}$ by multiplying its value by -1 as shown in ( 4.3 ). Then, the two objectives can be combined into a single minimization objective *Jsolar* using scalarization, and the optimization problem takes the form of equation ( 4.4 ).

$$Jcool_{\,min}\,(geometry, obstructions) = {}^{1}\Big/_{\sum_{i=0}^{n_{faces}} FaceArea_i} \sum_{i=0}^{n_{faces}} \left( FaceArea_i * \sum_{h=coolingSeasonStartDate}^{coolingSeasonEndDate} I_h(\,face_i, geometry, obstructions) \right) \tag{4.1}$$

$$Jheat_{\,max}(geometry,\ obstructions) = {}^{1}\Big/_{\sum_{i=0}^{n_{faces}} FaceArea_i} \sum_{i=0}^{n_{faces}} \left( FaceArea_i * \sum_{h=\mathrm{heat}ingSeasonStartDate}^{heatingSeasonEndDate} I_h(\,face_i, geometry, obstructions) \right) \tag{4.2}$$

$$Jheat_{\,min}(geometry, obstructions)\ = -Jheat_{\,max}(geometry, obstructions) \tag{4.3}$$

$$\min_{geometry}\ Jsolar(geometry, obstructions) = Jcool_{\,min}(geometry,\ obstructions) + \ Jheat_{\,min}(geometry,\ obstructions) \tag{4.4}$$

The importance of the two objectives is assumed proportional to the duration of their respective time frames. This factor is already embedded in ( 4.4 ) since $Jcool_{min}$ and $Jheat_{min}$ are calculated from the sum of the hourly intensities for the whole duration of each season. As a result, no further weighting of the two terms $Jcool_{min}$ and $Jheat_{min}$ is used. Note that in practice, the sum of the hourly intensities is approximated using a cumulative sky matrix, as detailed in the next section.

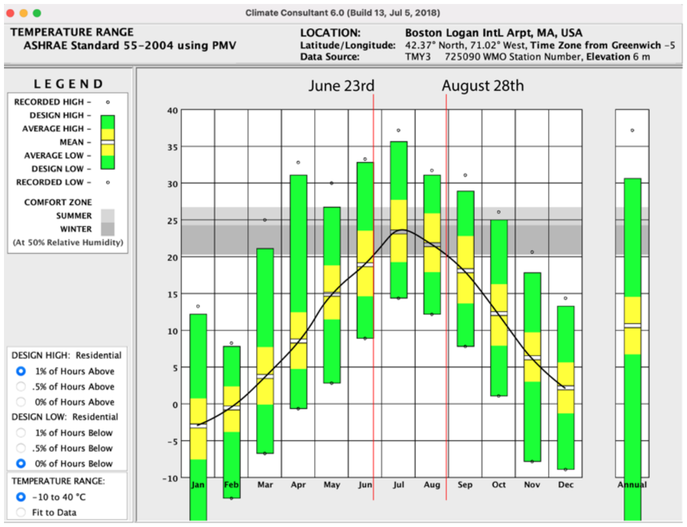


*Figure 4.3 Monthly temperature ranges for Boston and mean temperature curve. The red vertical lines indicate the intersection of the mean temperature curve with the threshold t=20°C.*

## 4.1.4. Simulation

In order to determine the exact timeframes of the heating and cooling periods, a temperature analysis was run for the location of Boston. The monthly temperature ranges were plotted using Climate Consultant (Ligget & Milne, 2018), and a cutoff threshold t = 20°C was defined. A continuous mean temperature curve was drawn by assigning the monthly mean temperature to the 15th day of each month and assuming a smooth interpolation of temperatures in between (Figure 4.3). Based on this graph, the dates of June 23rd and August 20th, when the mean temperature curve has a value of exactly 20$^{o}$C, were obtained as the transition dates from heating to cooling season and inversely.

Solar radiation calculations were initially performed using the open-source plugin Ladybug (Roudsari et al., 2013), inside the visual programming platform Grasshopper3d in McNeel's Rhinoceros 3d modeling software. The *IncidentRadiation* component was used with a Tregenza dome cumulative sky matrix to simplify the annual solar calculations. The total incident radiation is calculated as the sum of the direct radiation and the diffuse radiation ( 4.5 ). The algorithm, written in Python, implements backward ray-tracing using the CAD environment's methods for ray intersections. It does not take into account reflections of solar energy.

$$I_{total} = I_{direct} + I_{diffuse} \tag{4.5}$$

Due to limitations related to speed, communication with external processes, batch processing, and hardware parallelization, an equivalent, stand-alone simulation module was developed following the same algorithm. The cross-platform module was developed in Python. It makes extensive use of NumPy (Harris et al., 2020) array operations to replace loops and improve speed. Ray intersections are performed through the Trimesh library (Dawson-Haggerty et al., 2021) with Pyembree (Scopatz, 2015/2021) as the backend. This allows the use of multiprocessing during optimization, parallelizing the execution of simulations, and reducing the total computation time dependent on the number of available processors. The cumulative sky matrix was obtained in advance using Ladybug's *SkyMatrix* inside Grasshopper and exported to a JSON file using a custom Python script. Alternatively, a sky matrix can be obtained directly using an EPW file and a script capable of parsing the relevant information.

The solar irradiation simulation module was validated against Ladybug's *IncidentRadiation* component. The comparison included the calculation of incident solar radiation for 400 geometries partially shaded by a fixed context geometry for two different annual time intervals, corresponding to heating and cooling season (Appendix A). The mean percentage error was 2.53% for the heating season and 2.86% for the cooling season, which was considered satisfactory for the needs of this work.

## 4.2. Geometric System

This section details the development of a flexible geometric system with the ability to generate diverse geometries. Even though such a system can be technically seen as a parametric model, there are some fundamental differences. Typical parametric models in architecture are built using high-level concepts and geometric operations, for example, an adjustable height or twist of a tower. The number of parameters must be kept small and their effect easily traceable and understood to enable a meaningful manipulation from the designer.

The geometric system researched in this section is not targeted to be directly used by the designer, so concerns about interpretability or ease of manipulation do not apply at this level. Instead, the focus is on the granularity which can provide geometric flexibility, expressive power, suitability for the solar gain problem, and the ability to be optimized using an appropriate algorithm. Therefore, the number and intuitive nature of the system's free parameters are not concerns from a user perspective and do not impose any limitations on the system.

Following, a series of different geometric systems that were considered are presented with their advantages and disadvantages.

### 4.2.1. Heightmaps

A heightmap is a simple and intuitive way to describe a 2.5-D geometry, commonly used in computer graphics and geographic information systems (GIS). Heightmaps can be used to describe a building geometry by assigning a height on each point of a two-dimensional grid. In this work, a heightmap representing elevation points of a building roof in resolutions of 5X5 and 7X7 was considered. Figure 4.4 demonstrates such a heightmap on a 5X5 grid, with colors indicating the incident radiation intensity. This representation was used during an initial stage of the framework development (Ampanavos & Malkawi, 2021); however, it was later abandoned mainly due to its inability to represent fully three-dimensional geometries such as buildings with cantilevers, or negatively sloped walls, as well as limitations related to the topology of the geometry. Next, a series of three-dimensional representations with a higher expressive ability were considered.

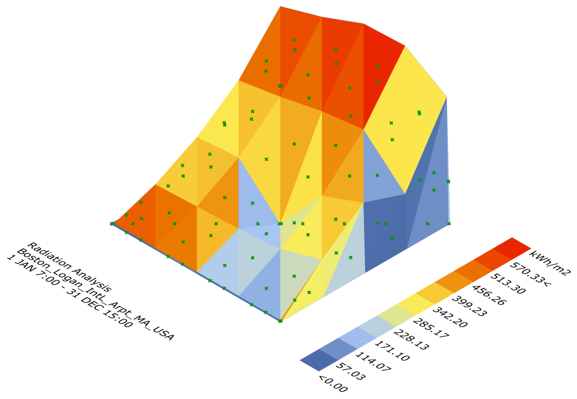


*Figure 4.4 Example of a geometry generated by a 5X5 heightmap. The grid points are linearly interpolated creating a triangulated mesh. The colors indicate heating period radiation intensity.*

### 4.2.2. Surfels

A system of elementary surfaces on a three-dimensional grid was created and named surfels. The surfels representation was inspired by a computer graphics rendering method that uses surface elements (surf-els) to efficiently render complex geometries (Pfister et al., 2000). In this work, the geometric system of surfels is comprised of a set of elementary square mesh faces on a three-dimensional grid. First, a three-dimensional grid is generated covering the complete site area and extending to the maximum allowed height. At each point of the grid, an elementary surface (surfel) is placed in the form of a square mesh face. Each surfel has four degrees of freedom: it can be uniformly scaled within the boundary of a grid cell, and it can rotate freely in space. Figure 4.5 shows an example of a surfel grid on a square plot shaded by two obstructions. Surfels colors represent the solar gain intensity per face $I_{obj}$, which following Equation ( 4.4 ) is defined as the difference of the heating period radiation intensity $I_{heating}$ from the cooling period radiation intensity $I_{cooling}$, detailed in Equation ( 4.6 ).

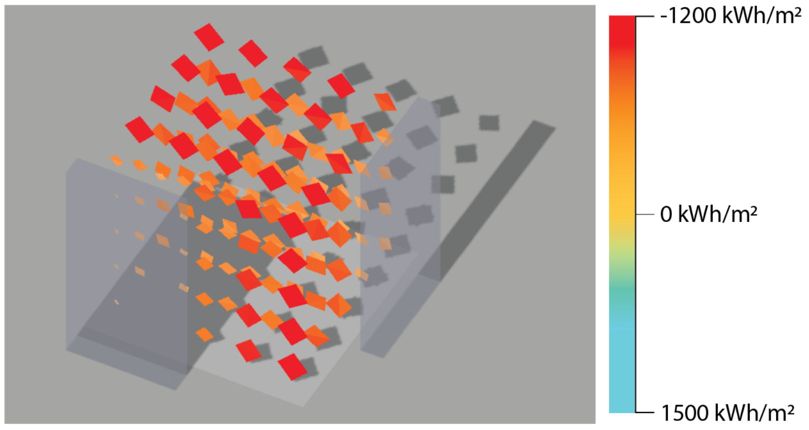


*Figure 4.5 Example of surfels populating a 5X5X5 grid. Colors represent the solar gain intensity per face.*

$$I_{obj} = I_{cooling} - I_{heating} \qquad (4.6)$$

Each surfel receives shadows from the other surfels, as well as the shading geometries of potential surrounding buildings. During optimization with a mean radiation objective, surfels are expected to acquire orientations that maximize their performance and scale up to the maximum or scale down to disappearance according to their potential to improve the overall performance.

The maximized surfels dictate the envelope shape of the building. Depending on the selected resolution of the grid, each surfel can be interpreted as a façade element with a single-floor height, a larger façade element that extends to multiple floors, or even a smaller façade element, with multiple surfels along the Z-axis required to cover the complete floor height.

Regarding the rotation parameters, three different ways were considered and implemented to represent a surfel’s orientation: axis-angle, quaternions, and Euler angles. Quaternions have better numerical properties and do not have singularities, but in the end, the Euler angles were selected due to their ability to be constrained within certain ranges in a more intuitive way. Specifically, extrinsic Euler angles using the ZXZ convention were used.

The complete surfels representation was developed in Python, using NumPy for matrix operations, and converting the computed vertices and faces to Trimesh geometries for direct interface with the simulator.

During experimentation, grids with a resolution of [5, 5, 5], [7, 7, 7], and [9, 9, 9] were used. Lower resolutions do not offer enough variability to meaningfully apply an optimization problem, while higher resolutions were too computationally expensive for this stage of the experimentation. It soon became obvious that a stochastic optimization process controlling a large number of individual geometries (between 125 and 729 square surfaces) tends to create results that appear ‘noisy’. Figure 4.6 shows such an example of an optimization result. The irregularity of these forms prevents them from being easily translated to building surfaces, considering the constraints that exist in the physical world.

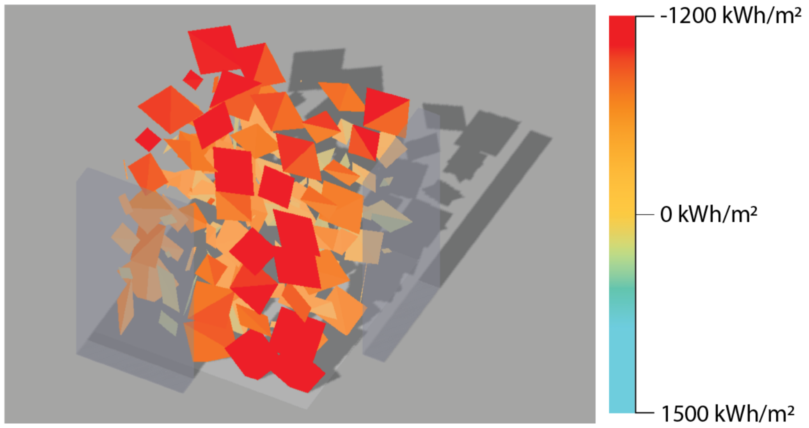


*Figure 4.6 Surfels grid optimized for solar gain using 500 independent free parameters (125 surfaces X 4 degrees of freedom).*

Previous work on the geometric optimization of structural shells has identified a similar problem of optimization processes generating highly irregular forms (Michalatos & Kaijima, 2014). That work suggested a solution taking a signal processing perspective, using eigenvectors to modify individual nodes' parameters. Other work has used 2D Fourier Series to reduce the number of free parameters and create smooth, continuous geometries for a solar gain optimization problem (Kämpf & Robinson, 2010). Following these precedents, the variation of the elementary geometries was constrained through the use of spatial functions. A series of parametrized three-dimensional waves were used as a relaxed generalization of the Fourier Series. The frequency of each of the X, Y, and Z dimensions (f*x, fy, fz*), the phase ($\varphi$), and the intensity (*scale*) of the waves are all controlled individually. Equations ( 4.7 ) and ( 4.8 ) detail the calculation of each of the four parameters (scale and 3d rotation) of a surfel$_i$, $i \in \{0, \ldots, gridResolution^3 - 1\}$ given the number of wave functions $n_{waves}$, and the five system-wide free parameters for each wave *j {scalej, fxj, fyj, fzj,* $\varphi j$ *}* $\in$ *[0, 1]*. The resulting surfel parameter depends on the surfel's location *(posX, posY, poxZ)*, where *posX, posY, posZ* $\in$ *[0, gridSide]*. The maximum frequency hyperparameter *maxf* is fixed system-wide and was set to *maxf = 1.5* after initial experimentation. The factor *w* is a normalizing factor dependent on the dimensions of the total grid *(gridSide)*, defined in Equation ( 4.8 ). The total number of waves was set to two *(nwaves = 2)*, after considering alternatives of one, two, or three waves.

$$Param_{element_i} = \sum_{j=0}^{n_{waves}} scale_j * \cos\left(fx_j * maxf * w * posX_{element_i} + fy_j * maxf * w * posY_{element_i} + fz_j * maxf * w * posZ_{element_i} + \varphi_j * 2 * \pi\right) \quad (4.7)$$

$$w = {}^{2*\pi}/_{gridSide} \quad (4.8)$$

In conclusion, instead of modifying each individual surfel geometry along the four degrees of freedom, a set of wave functions controls each degree of freedom for all the elementary geometries of the system, reducing the number of free optimization parameters to 40 ($n_{degreesOfFreedom}$X $n_{waves}$ X $n_{waveParameters}$ = 4 X 2 X 5 = 40), regardless of the selected resolution of the grid. The result is smoother transitions from each surfel to its surroundings, forming groups of surfels that can easily be interpreted as building facades, as shown in Figure 4.7. The trapezoids marked with dashed lines indicate how the surfels can form larger façade segments.

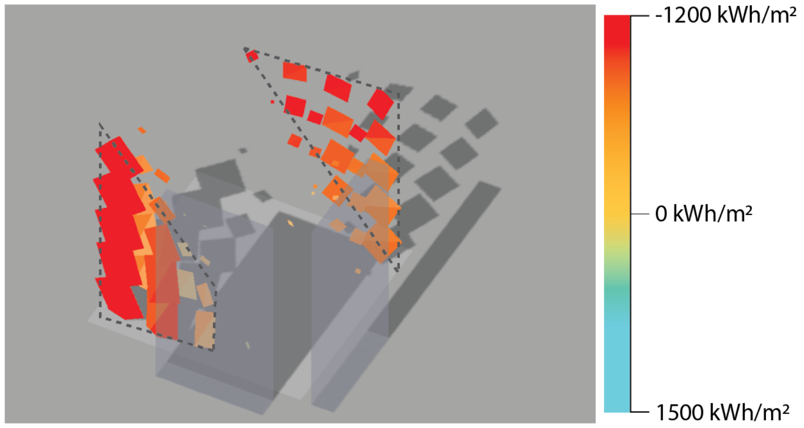


*Figure 4.7 Surfels grid optimized for solar gain using 40 free parameters. The 500 geometric parameters are controlled by two 3-dimemsional waves.*

In practice, the surfels system was able to generate façade segments but not complete building envelopes. The optimal 3d orientation for a flat surface was found to be always south-facing for the location of Boston, MA, with small angle deviations for different shading configurations of other surfels and surrounding buildings. A study on shading configurations also showed limited sensitivity of the system to the positions of East and West shading obstructions. Therefore, other representations that are volumetric by nature were considered as better alternatives given the geometric constraints of a building shape.

### 4.2.3. Orientable Cuboids

The orientable cuboids representation was developed as the volumetric extension of the surfels. A three-dimensional square grid is fit on the site and an orthogonal cuboid is placed on each grid point. Each cuboid can be scaled along x, y, and z and can be freely rotated in space. The rotations were restricted to 90° around the X and Y axes and 180° around the Z axis.

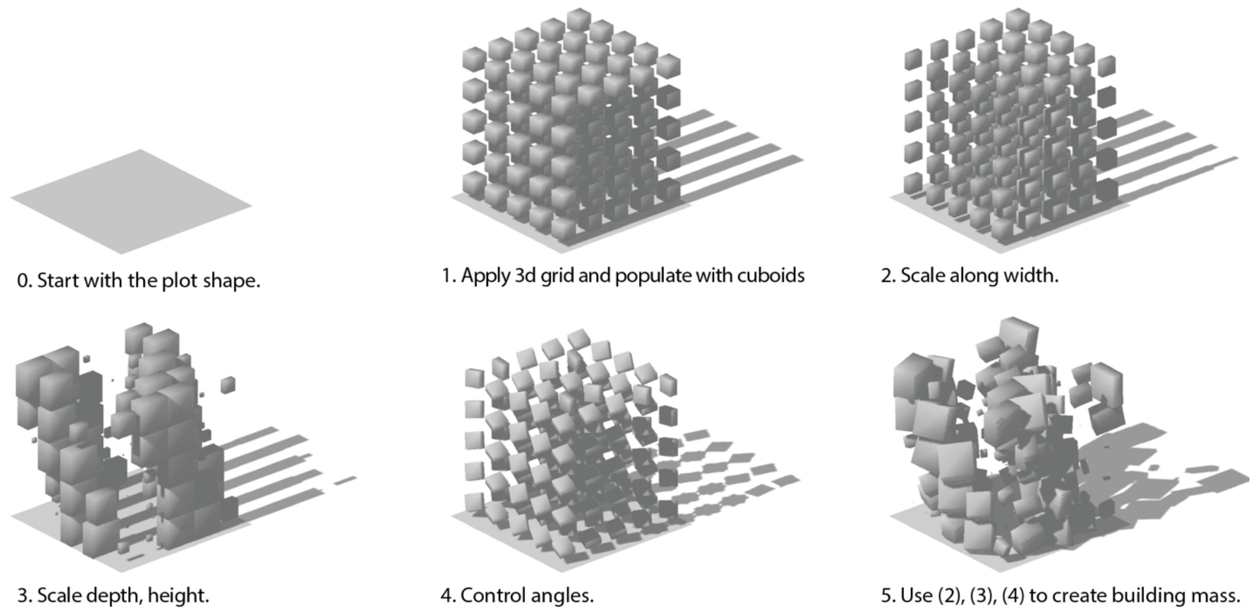

0. Start with the plot shape.
1. Apply 3d grid and populate with cuboids
2. Scale along width.
3. Scale depth, height.
4. Control angles.
5. Use (2), (3), (4) to create building mass.

*Figure 4.8 Orientable cuboids. Steps 1-5 demonstrate how the system is used to generate building volumes through 3d scaling and rotations of the cuboids.*

Similar to the surfels system, each degree of freedom of a single cuboid is handled at the system level by a series of *n* three-dimensional waves using equation ( 4.7 ), where n was set to two ($n_{waves}$=2). The resolution of the grid can be adjusted in a way such that each cuboid roughly corresponds to the size of a room. A 5X5X5 grid was used in the experiments. Like the surfels, the cuboids were also implemented in Python.

A series of modifications were applied to the initial formulation to facilitate the correspondence of cuboids aggregations to building volumes. First, the height of each cuboid was fixed as equal to the grid step. In this way, each cuboid, or room, always fills the full floor height. Second, to enable the formation of connected spaces for adjacent cuboids with a scaled width and height less than the grid step, an anisotropic scaling was introduced. Figure 4.9 details how each of the previously defined width and length scaling parameters W and L were replaced by a pair of parameters Wa, Wb and La, Lb, each controlling one direction of the width or height scaling around the grid cell center. Third, to better define the distinction between built and unbuilt space, a hard threshold *th* was used to adjust the size of the cuboids. Cuboids with at least one of their sides scaled below th=0.35 were scaled down to zero, in practice being removed from the resulting geometry. To further bias the system towards distinguishing built from unbuilt cuboids, all the scaling parameters passed through a sigmoid function. Last, to further enable connectivity, especially for rotated cuboids, the maximum scaling was increased to 1.6 times the grid step, allowing substantial overlaps between cuboids.

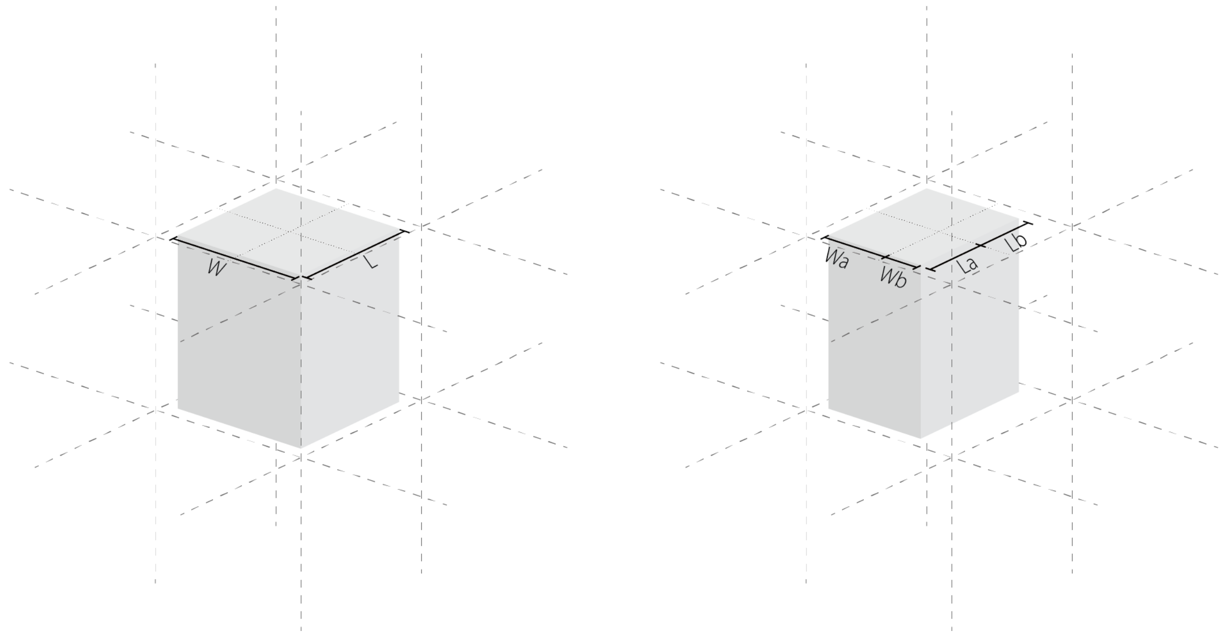


*Figure 4.9 Left: Width and Length of the cuboid are symmetrically scaled around the grid cell center. Right: Anisotropic scaling of the width and length scaling of the cuboid.*

Figure 4.10 illustrates the effect of these changes on an optimization result where all other parameters have been kept the same. The geometry on the left is defined using the initial setup of the cuboids system, while the geometry on the right has cuboids with constant height, anisotropic scaling of the cuboids' width and length, sigmoid-modified scaling parameters, and smaller cuboids filtered out using a threshold th=0.35. The marked areas show how similarly scaled cuboids along the X and Y axes are disconnected on the left solution but connected and forming a larger geometry on the right. Smaller cuboids such as those on the upper south and lower north areas of the solution on the left have disappeared from the updated solution on the right.

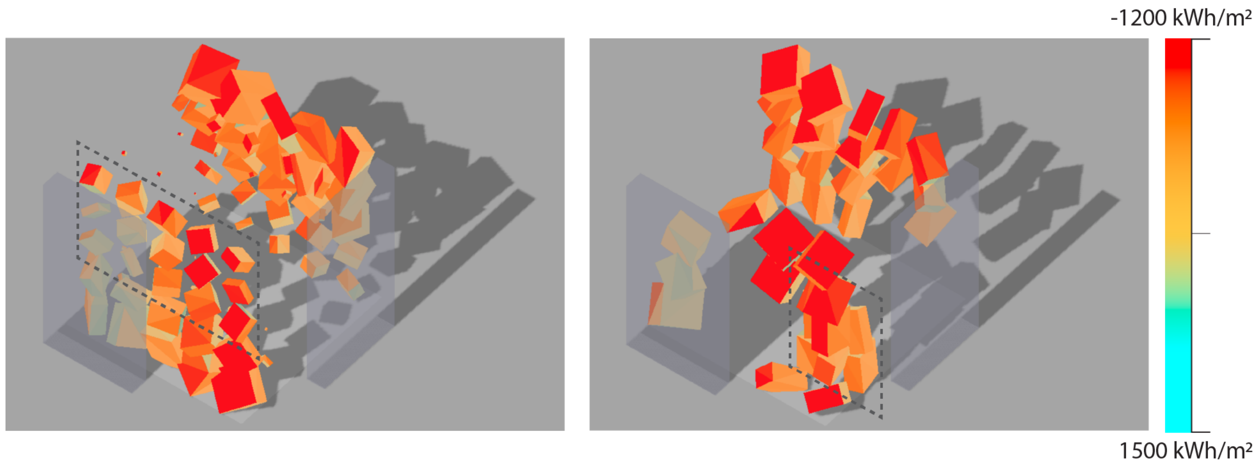


*Figure 4.10 South-east isometric view of optimized geometries using the initial (left) and updated (right) cuboid definitions.*

## 4.3. Single Problem Optimization

Once the geometric system has been selected, the free parameters defined, the performance objective formulated, and the simulation software to evaluate the objectives configured, an optimal geometry can be obtained using any stochastic optimization algorithm. A Genetic Algorithm was used for experiments with a single objective and the evolutionary algorithm NSGA II was used for experiments with multiple objectives. The optimizations were performed using the Python library Pymoo (Blank & Deb, 2020) with the library's default parameters for each algorithm.

### 4.3.1. Geometric Objectives

To steer the morphology of the generated geometries towards more desirable forms during the optimization process, a set of secondary, geometry-related optimization objectives was added to the problem. In total, three metrics related to the geometric form were used.

First, regarding the variation of the cuboids' parameters along the three spatial axes, it became obvious that the rotation and the scaling parameters should be subject to different constraints. The scaling parameters are defining the built and unbuilt spaces and the rate of change should naturally be high close to the transition border. On the other hand, a high rate of change for the rotation parameters of a set of cuboids describing the built space results in highly irregular geometries that create unnecessary and undesirable spatial complexity. To steer the search of the design space away from such solutions, a rotational variance metric *RotVar* was defined as the mean of the local rotational variance of all cuboids, detailed in Equation ( 4.9 ), where $n_{cuboids}$ is the total number of cuboids in the grid. The local rotational variance is calculated as the weighted variance of the rotation parameters of the neighborhood of a single cuboid, detailed in Equation ( 4.10 ), where *cuboid*$_j$, *j={1, …7}*, same as *cuboid*$_m$, *m={1, …, 7}*, is the neighborhood of a *cuboid* defined as the *cuboid*'s six immediate neighbors and itself, and *rotation*$_k$, *k={1, 2, 3}* represents the three rotational parameters.

$$RotVar = {}^{1}\!/_{n_{cuboids}} \sum_{i=1}^{n_{cuboids}} LocalRotVar(Cuboid_i) \qquad (4.9)$$

$$
\begin{aligned}
&LocalRotVar(Cuboid\,) \\
&\quad = {}^{1}\!/_{21} \sum_{k=1}^{3} \sum_{j=1}^{7} \Bigg( cuboid_{j_{rotation_k}} \\
&\quad - {}^{1}\!/_{7} \sum_{m=1}^{7} cuboid_{m_{rotation_k}} * \left( cuboid_{m_{scale_{x1}}} + cuboid_{m_{scale_{x2}}} \right) \\
&\quad * \left( cuboid_{m_{scale_{y1}}} + cuboid_{m_{scale_{y2}}} \right) \Bigg)^{2} \qquad (4.10) \\
&\quad * \left( cuboid_{j_{scale_{x1}}} + cuboid_{j_{scale_{x2}}} \right) \\
&\quad * \left( cuboid_{j_{scale_{y1}}} + cuboid_{j_{scale_{y2}}} \right)
\end{aligned}
$$

Figure 4.11 shows two optimization results from an example problem with two objectives: minimization of radiation objective ( 4.4 ) and rotational variance objective ( 4.9 ). The geometry on the left has been selected to favor the radiation objective, while the geometry on the right favors the rotational variance objective. Both geometries present a smoother transition of orientations between neighboring cuboids than the geometries in Figure 4.10, which have been optimized using the radiation objective alone.

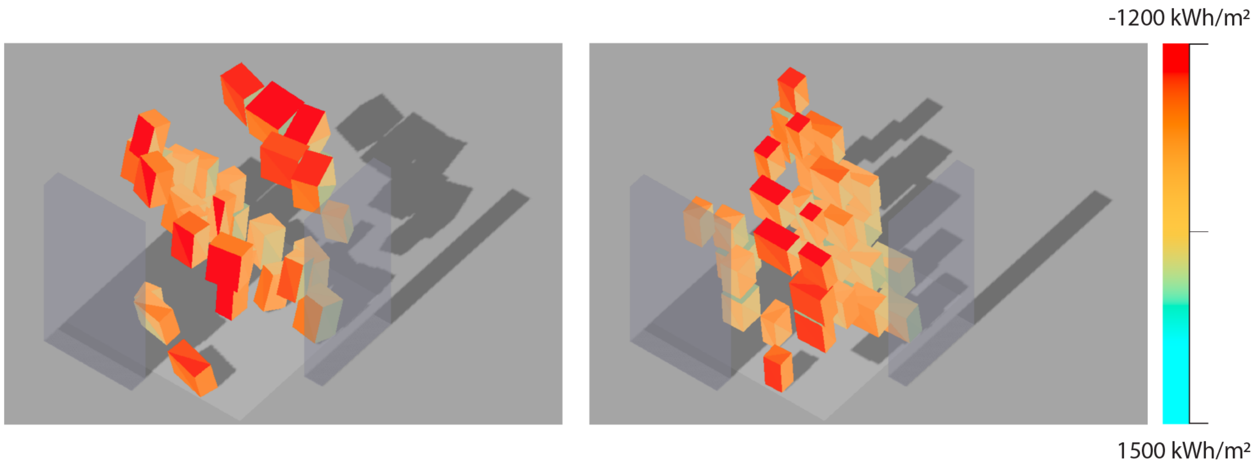


*Figure 4.11 South-east isometric of optimized geometries examples using both radiation and rotational variance objectives.*

Regarding the cuboids' connectivity, while the anisotropic scaling that was introduced earlier enables continuity of space between smaller cuboids, the solutions need to be biased in order to take advantage of this ability. This is evident in Figure 4.10 Right, where there is no such bias and as a result, the geometry includes some connected cuboids (south-east corner) but also many disconnected ones (north side). A preference for the adjacent cuboids' connectivity can be expressed as part of the optimization objective in order to manifest in the generated geometries. Therefore, a proximity metric *FacesProximity* was defined, computed for all existing neighbors on the XY plane for every cuboid defined as built after the threshold is applied. Equations ( 4.11 ) and ( 4.12 ) describe the FacesProximity penalty, where $Cuboid_i$ is the *i*-th cuboid from the subset of built cuboids after the threshold *th* has been applied, and $neighborVertex_{ijk}$ is the *k*-th vertex of the *j*-th immediate neighbor of the *i*-th cuboid on the XY plane.

$$FacesProximity = -1/n_{\text{cuboids}-built} \sum_{i=0}^{n_{cuboids-built}} LocalFacesProximity(Cuboid_i) \quad (4.11)$$

$$LocalFacesProximity(V_i) = 1/n_{neighbors} \sum_{j=0}^{n_{neighbors}} \max\{SignedDistance(V_i, neighborVertex_{ijk}), k = 1, \dots 4\} \quad (4.12)$$

Figure 4.12 shows two example results from an optimization problem using the radiation objective ( 4.4 ) and the faces proximity objective ( 4.11 ). Compared to Figure 4.10 and Figure 4.11, these geometries have better connectivity of adjacent cuboids; all neighboring cuboids seem to be at least partially in contact, making it easier to interpret them as continuous, connected spaces of a building.

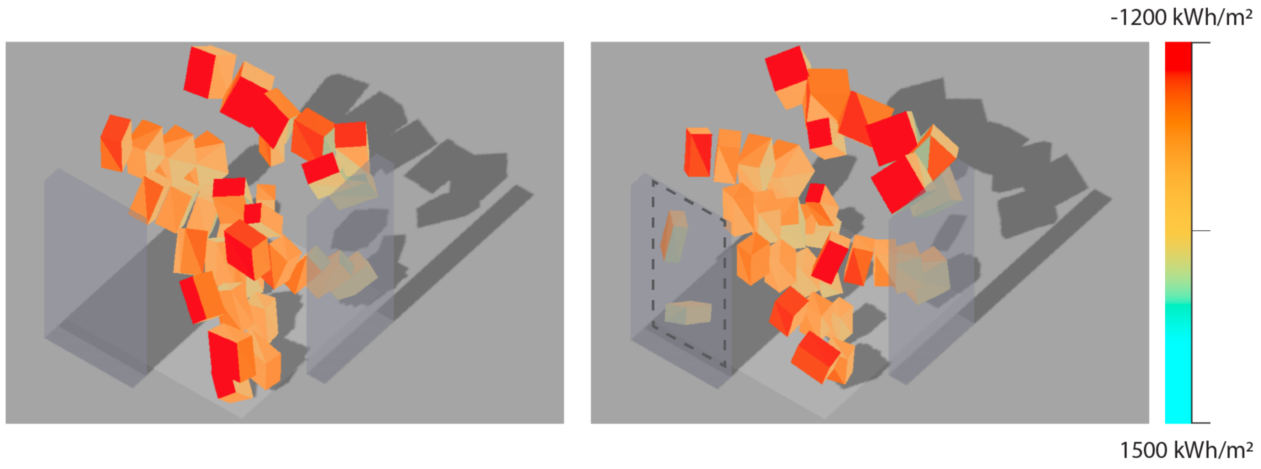


*Figure 4.12 South-east isometric of optimized geometries examples using both radiation and face proximity objectives.*

Even though the *FacesProximity* objective can steer adjacent cuboids closer together, it does not affect the presence of single cuboids with no neighbors – for example, those marked in Figure 4.12 Right, which remains a source of unrealistic – or unusual – building designs. In order to avoid single, disconnected cuboids and further promote larger aggregations of adjacent cuboids, a *MeanValence* metric was defined in Equation ( 4.13 ) based on the number of existing (built) immediate neighbors for each built cuboid.

$$MeanValence = -{}^{1}/_{n_{cuboidsbuilt}} \sum_{i=0}^{n_{cuboidsbuilt}} NumberOfNeighbors(Cuboid_i) \qquad (4.13)$$

Figure 4.13 shows two optimization results for a problem with radiation ( 4.4 ) and cuboid valence ( 4.13 ) objectives. Contrary to Figure 4.11 and Figure 4.12 there are no single, disconnected cuboids in these geometries.

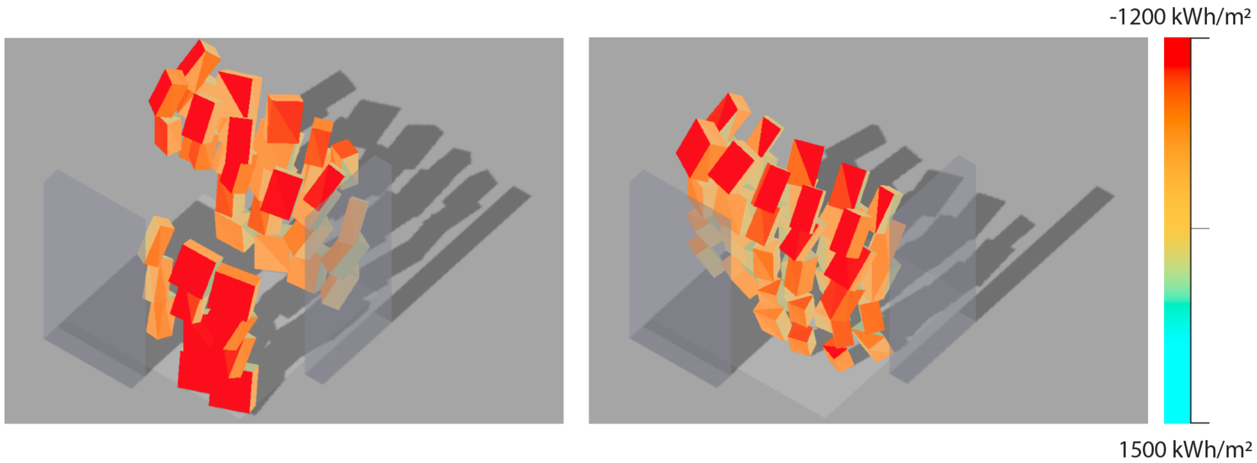


*Figure 4.13 South-east isometric of optimized geometry examples using both radiation and cuboid valence objectives.*

Next, the three metrics, that describe morphological attributes of a geometry instance using the cuboids system were combined into a single objective *Jgeom* as shown in Equation ( 4.14 ). The weights $f_{rot}$, $f_{prox}$ and $f_{val}$ are hyperparameters subject to further tuning and, for these experiments, were set to $f_{rot} = 1.0$, $f_{prox} = 1.0$, $f_{val} = 1/6.0$. Minimizing this objective steers the solutions towards more desirable forms with less irregular rotations, better connected and less dispersed cuboids, as illustrated in Figure 4.14.

$$J_{geom} = f_{rot} * \ RotVar + f_{prox} * \ FacesProximity + f_{val} * MeanValence \qquad (4.14)$$

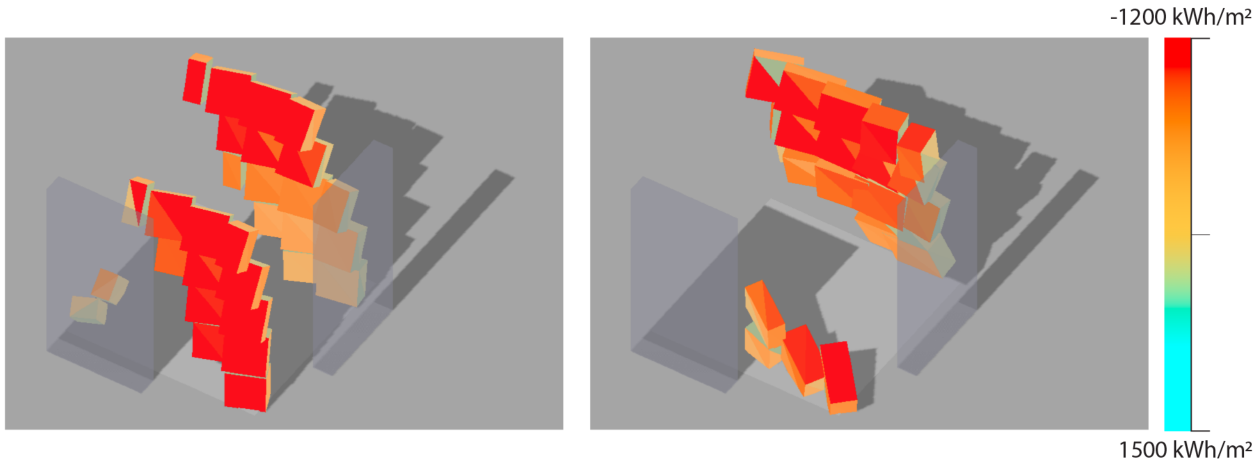


*Figure 4.14 South-east isometric of optimized geometry examples using both the radiation and the combined geometric objectives.*

### 4.3.2. Optimization Constraints

Apart from the performance and geometry objectives of Equations ( 4.4 ) and ( 4.14 ), additional information related to the architectural program, location, plot shape, surroundings of the site, and potential building constraints such as the ones regulated by code need to be used to determine the geometric solutions.

The architectural program determines the size of the building and can be expressed as the total volume of the geometry. During the initial phase of experimentation, the building volume was used as a constraint for the optimization problem, with a tolerance of up to 10% of the site volume. The constraint $G_{vol}$ is described in Equation ( *4.15* ), where *MaxVol* is the maximum available volume of the site, *TargetVol* is the required volume derived from the architectural program, and *GeneratedVol* is the volume of the generated geometry instance to be evaluated.

$$G_{vol} = |TargetVol - GeneratedVol| - 0.01 * MaxVol, \quad G_{vol} \leq 0 \qquad (4.15)$$

A similar constraint can be used for the area coverage, as dictated by code or design preference ( 4.16 ). Regarding other building constraints, a maximum allowed height is considered ( 4.17 ). The plot shape together with the maximum allowed height define the available three-dimensional space where the cuboids will be generated. Finally, the location, plot shape, and surroundings are direct inputs to the solar simulation software.

$$\begin{gathered} G_{areaCoverage} = GeneratedAreaCoverage - MaxAreaCoverage, \\ G_{areaCoverage} \leq 0 \end{gathered} \qquad (4.16)$$

$$G_{height} = GeneratedHeight - MaxHeight, \qquad G_{height} \leq 0 \qquad (4.17)$$

## 4.4. Dataset Optimization

The previous section described how optimal geometries for a single, fully defined problem can be generated. This section focuses on creating a set of problem definitions and obtaining optimization results for them in an efficient way. Given the already formulated objectives, a problem brief describes the simulation boundary condition and the optimization constraints. An appropriate sampling of both is necessary for compiling an inclusive synthetic dataset.

### 4.4.1. Optimization Constraints Sampling

In the final dataset, geometries with varying volumes and areas need to be generated for each of the conditions defined above. The total built volume is important for two reasons. First, as already discussed, it is a metric related to the architectural program. In order to build knowledge about handling different building sizes, optimal geometries from the whole range of possible volumes need to be generated. Second, the aim of the framework is to build a system that suggests optimally performing geometries interactively at different stages of the design exploration. This means that regardless of the final building size, optimally performing geometries of various sizes will need to be generated.

In addition, the built area coverage is another important attribute, enabling compliance with potential building constraints and design preferences, but also allowing more flexibility during the form-finding exploration even with partially completed solutions.

One way to satisfy the requirement for this range of geometric solutions for a single site would be to use the constraints $G_{vol}$ and $G_{area}$, described in Equations ( 4.15 ) and ( 4.16 ), or perhaps to relax these constraints and convert them to secondary optimization objectives $J_{vol}$ and $J_{area}$ – for example, minimizing the squared difference of the generated quantity from the target one as shown in Equations ( 4.18 ) and ( 4.19 ). Then, multiple optimizations would be performed for each single site, each with different values for TargetVolume and TargetArea.

$$J_{vol} = (TargetVolume - GeneratedVolume)^2 \tag{4.18}$$

$$J_{area} = (TargetArea - GeneratedArea)^2 \tag{4.19}$$

In this work, instead, the geometric attributes are used as additional optimization objectives $J_{samplingVol}$ and $J_{samplingArea}$, of a multi-objective optimization problem, formulated in such a way that the set of optimal solutions obtained from a single optimization process contains geometries with attributes from the whole range of interest. That is, instead of running multiple optimizations with a different target volume and target area for each building site, a single, multi-objective optimization is run that returns equivalent solutions.

The formulation of the sampling objectives $J_{samplingVol}$ and $J_{samplingArea}$ relies on the fact that a typical multi-objective algorithm obtains a Pareto set of solutions approximating the Pareto front of the problem. The Pareto front is the set of non-dominated solutions and manifests the tradeoffs between the different optimization metrics. Therefore, to obtain a set of solutions for geometries with various volumes and areas, a tradeoff between these quantities and the primary objective of solar gain performance needs to be identified first. Intuitively, for a fixed site, a higher potential for solar gain performance is expected for smaller geometries, since there is more flexibility in taking advantage of the surrounding obstructions to improve performance. Location-specific factors also affect the results. In Boston, where the heating season is much longer than the cooling season, a geometry with a higher area coverage has better potential to use the winter sun and obtain optimal overall solar gain performance.

A series of experiments supported these hypotheses. Figure 4.15 shows examples of multi-objective optimization results using solar gain and volume objectives for a single simulation context after 100 generations. The top results were acquired using volume minimization, the bottom ones using volume maximization objective. On the right, each point of the scatterplots represents the performance of a single geometric solution. Green color represents the whole optimization history, i.e., every geometry considered during the optimization process. Brown color represents the Pareto set obtained in the last generation. Red color is used to represent the performance of the example solution rendered on the left. In each scatterplot, the optimal solutions would occupy the bottom left corner of the plot. In the rendered geometries (left), warmer colors indicate better solar gain performance, while cooler colors indicate worse.

As shown in Figure 4.15 Top, when a volume minimization objective was used together with the solar gain objective all the solutions obtained had a minimal volume. The Pareto set is almost a horizontal line (Figure 4.15 Top Right in brown color), with geometries of total volume between zero and 3m$^3$ that have a very wide range of solar gain performances. This set is far from the desired solutions for the problem. In contrast, when a volume maximization objective was used, the optimal solutions included geometries with volumes from the whole possible range (Figure 4.15 Bottom), with solar gain performance tradeoffs related to the geometry size.

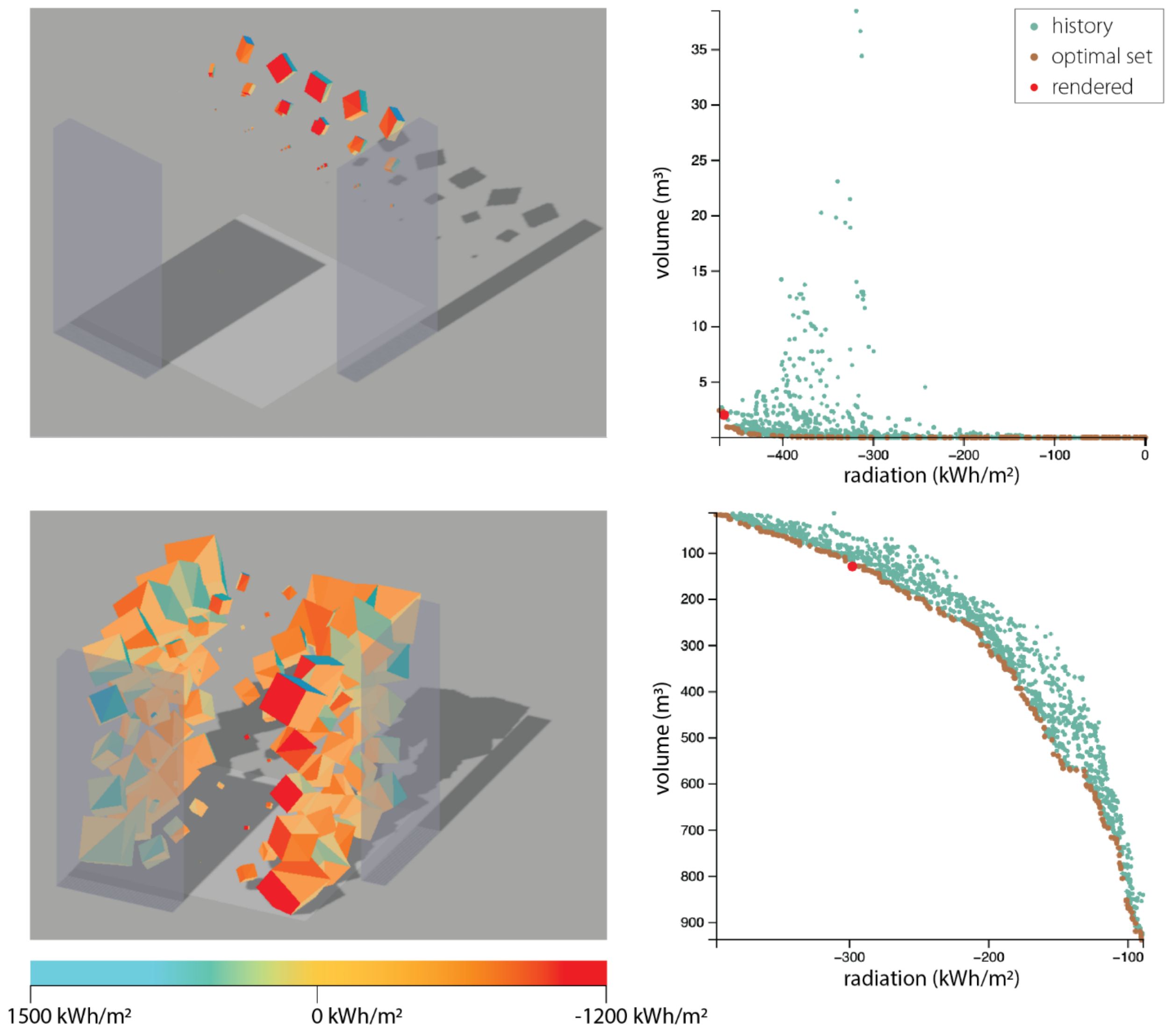


*Figure 4.15 Examples of multi-objective optimization results using solar gain (X-axis) and volume (Y-axis) objectives for a single simulation context. In the top row the volume is maximized, in the bottom row the volume is minimized.*

An inverse trend was observed regarding the area coverage. In the example of Figure 4.16, when the area coverage was used as a maximization objective, only geometries with coverage between 78% and 88% were obtained (Figure 4.16 Top). However, when the area coverage was instead minimized, the solutions spanned a wider range between 20% and 35% (Figure 4.16 Bottom).

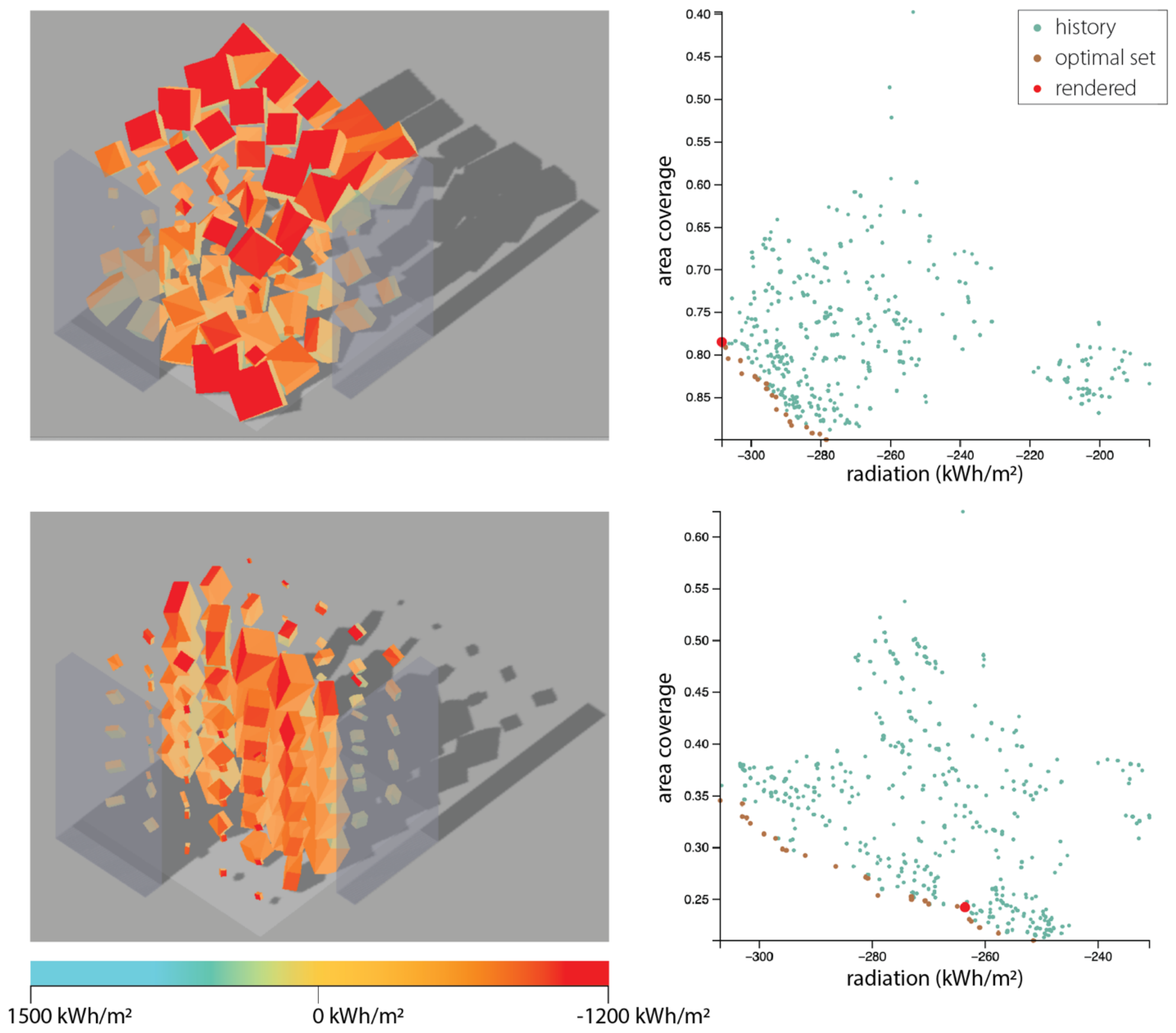


*Figure 4.16 Examples of multi-objective optimization results using solar gain (X-axis) and area coverage (Y-axis) objectives for a single simulation context. In the top row the area coverage is maximized, in the bottom row the area coverage is minimized.*

Last, Figure 4.17 shows examples of optimization results where all three objectives of solar gain, volume, and area coverage were used. The three performance objectives cannot be easily visualized in a scatterplot, so a parallel coordinates graph has been used. Each solution is represented by a line, indicating the performance along the three axes. Optimal solutions would occupy the bottom area on each axes. The tradeoffs between the three objectives cause the optimized solutions to spread along the whole range of possible volumes (0 – 1000m$^3$ in a 10mX10m plot with 10m maximum height) and areas (0 – 100%), as shown in Figure 4.17. These results confirmed that a single, multi-objective problem can provide a diverse sampling of the volume and area constraints for the problem of optimal solar gain.

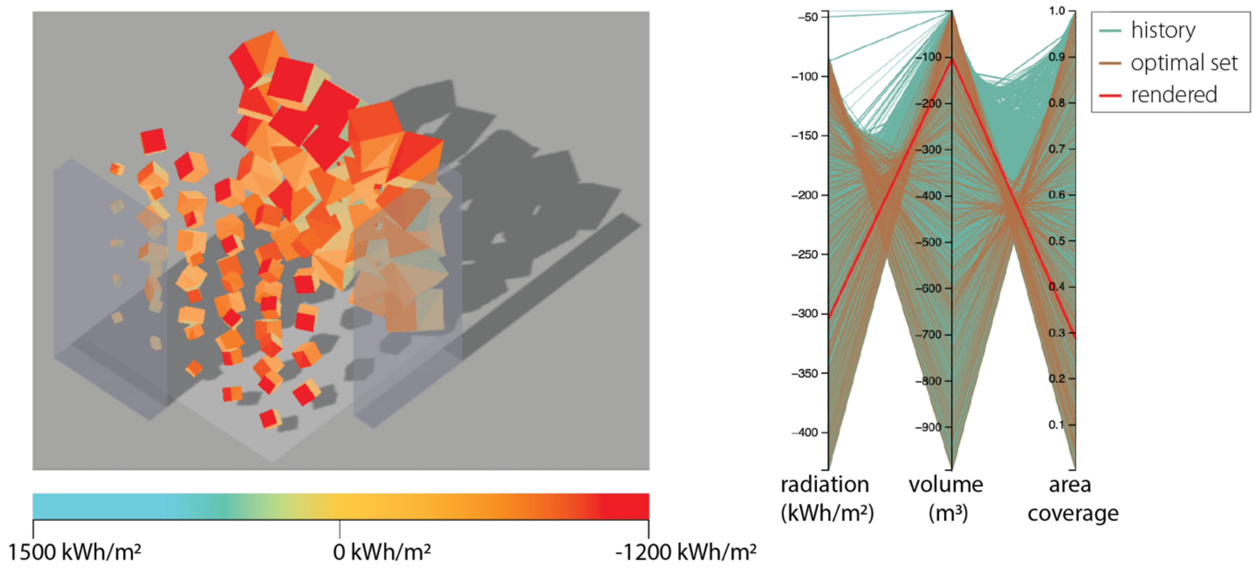


*Figure 4.17 Examples of multi-objective optimization results using solar gain, volume, and area coverage objectives for a single simulation context.*

The final volume and area objectives are given in Equations ( 4.20 ) and ( 4.21 ), respectively. The area coverage calculation was implemented using ray casting with a 35X35 grid.

$$J_{samplingVolume} = -Volume \tag{4.20}$$

$$J_{samplingArea} = AreaCoverage \tag{4.21}$$

### 4.4.2. Boundary Conditions Sampling

The simulation boundary condition consists of the location, plot shape, and surrounding buildings. In this case study, only a single location is considered, therefore this section describes the plot shape and surrounding buildings sampling method. By extension, any time the term "boundary condition" appears in this dissertation, it will refer to these two elements.

Given a selected location, various sites can be sampled from the real world using GIS data. Alternatively, synthetic data can be generated in a more generalized way. In this work, the sites are generated using a grid system. First, the maximum plot considered is defined as a square with a side of twenty-five meters (25m X 25m). A 5 X 5 grid is placed in this square (Figure 4.18a). A subset of the cells of this grid comprises the building plot. First, a number of cells are selected by randomly assigning a binary label to each cell (Figure 4.18b). Then, to avoid shapes that are excessively complex or disconnected, the convex hull of the selected cells' center points is generated (Figure 4.18c). The final building plot consists of the cells whose centers are contained within this convex hull. Padding of size one is added on all sides of the grid, resulting in a total grid of 7 X 7 cells (Figure 4.18d). Any cell that is not part of the building plot may contain part of the surrounding buildings.

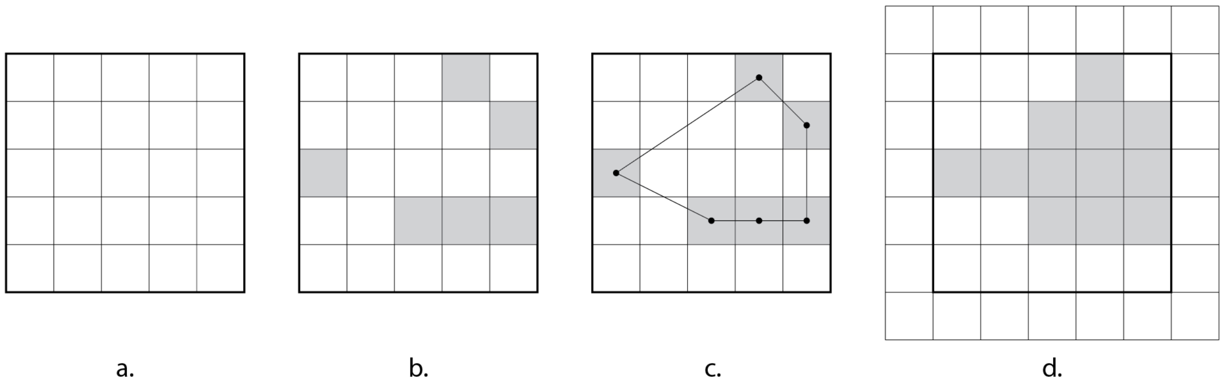


*Figure 4.18 Steps for plot shape sampling.*

The surrounding buildings are abstracted and represented on the same grid in the form of an incident solar radiation obstructions heightmap. Grid cells outside the convex hull are assigned heights between zero and the maximum height h ∈ [0, $\text{height}_{max}$]. Heights can be assigned using different strategies. For example, to reflect the distribution of real-world heights in a specific location, existing building heights can be sampled using GIS data and form a distribution. Then, the obstructions heightmap can be sampled from this probability distribution.

In this work a generalized strategy was used to generate heightmaps, either sampling from a uniform distribution or using a zone-based strategy. In the first case, each grid cell *i* is randomly assigned a height $h_i$ ~ U(0, $\text{height}_{max}$). In the zone-based case, first, four obstructions zones are defined on the north, south, east, and west sides of the 7 X 7 grid, as shown in Figure 4.19. Then, a subset of zones is selected, and each zone *j* is assigned a height $h_j$ ∈ [0, $\text{height}_{max}$]. Next, each grid cell *I* is assigned a height $h_i$ by adding some noise *e* to the corresponding zone height $h_j$ (Equation ( 4.23 )).

$$\mathrm{h_i} \sim U(h_j - e, h_j + e) \tag{4.22}$$

Conflicts at corner cells are resolved at random on the zone level. In either case, grid cells that belong to the building plot are assigned zero heights.

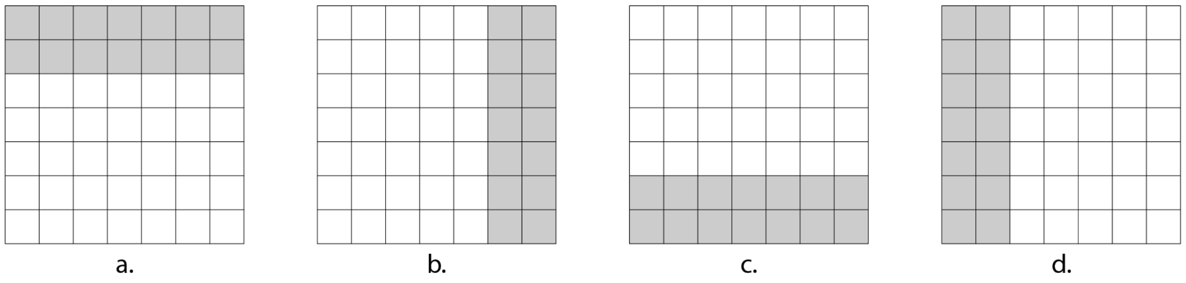


*Figure 4.19 Zones for obstruction configurations: a subset of zones from (a, b, c, d) are combined to form a single configuration.*

Figure 4.20 shows examples of resulting boundary conditions where the obstructions heightmap has been sampled using a uniform distribution. Figure 4.21 shows examples of resulting boundary conditions where the obstructions heightmap has been generated using the zone-based strategy.

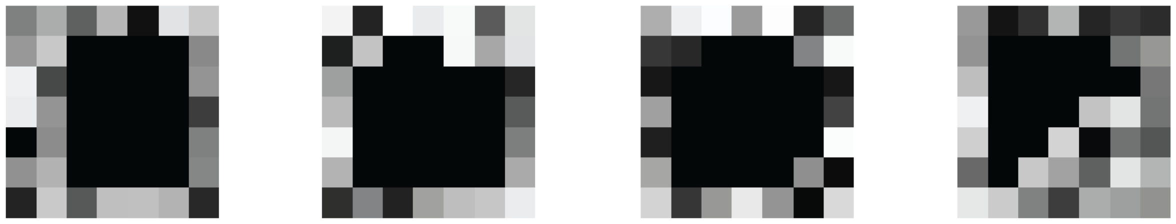


*Figure 4.20 Examples of boundary conditions with uniformly sampled obstructions.*

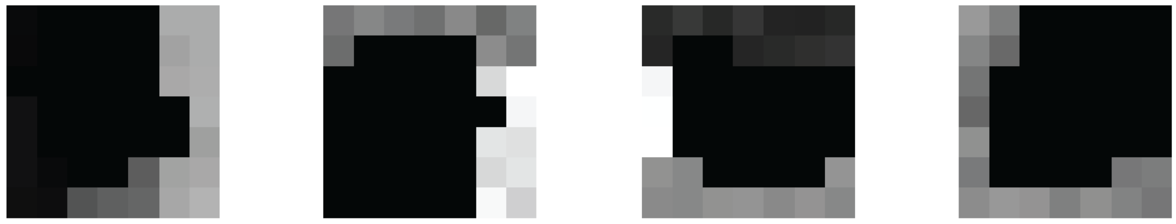


*Figure 4.21 Examples of boundary conditions with zone-based obstructions.*

A complete problem set is generated by combining each plot shape with a number of obstruction heightmaps. Figure 4.22 outlines the process of creating a set of *m X n* boundary conditions, by sampling *n* obstruction heightmaps for each of *m* plot shapes.

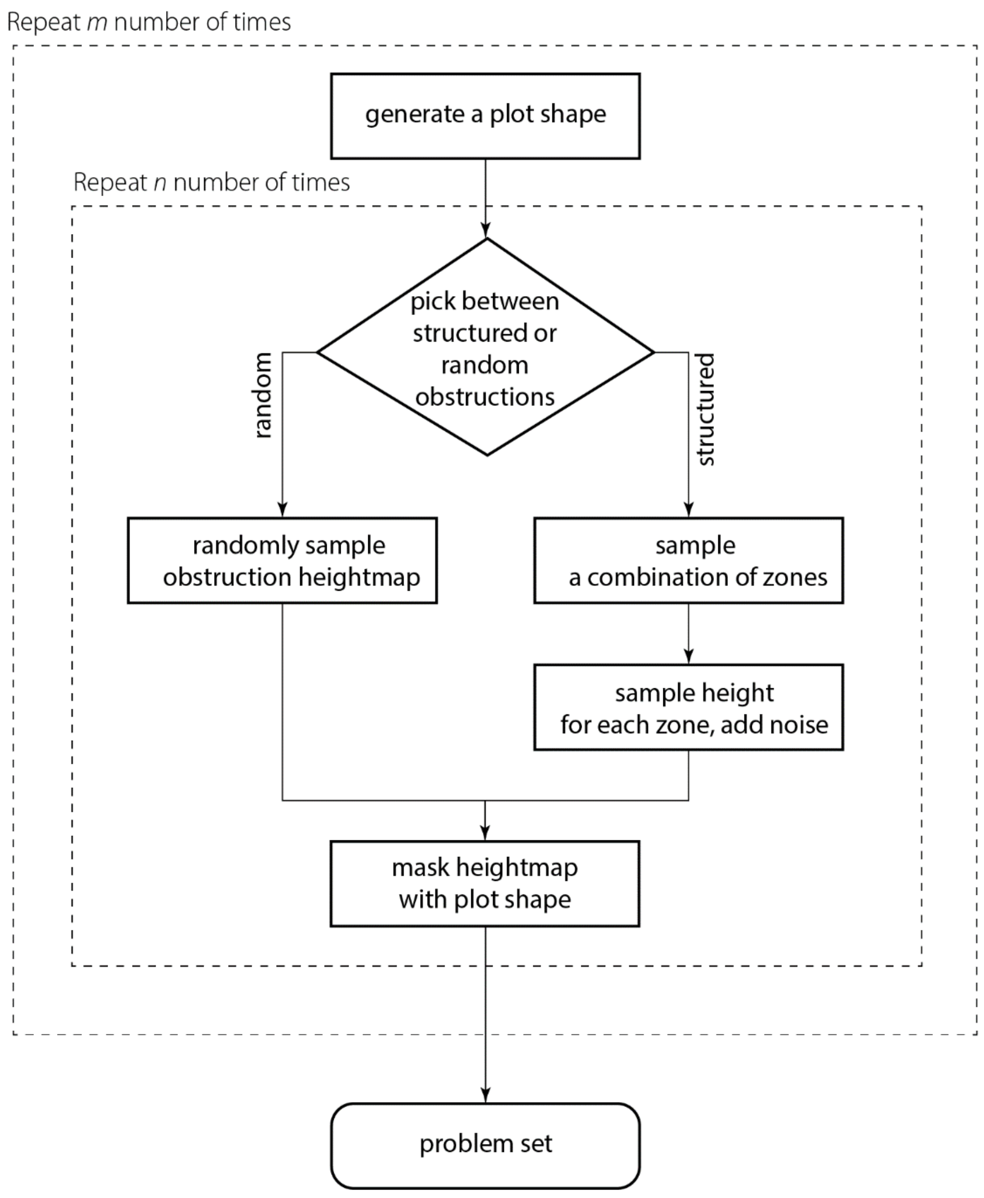

Repeat *m* number of times
generate a plot shape
Repeat *n* number of times
pick between structured or random obstructions
random
structured
randomly sample obstruction heightmap
sample a combination of zones
sample height for each zone, add noise
mask heightmap with plot shape
problem set

*Figure 4.22 Generation of a problem set, containing m X n boundary conditions.*

Finally, during the geometry generation, the cuboids grid is placed inside the maximum plot square of 25m X 25m and up to the maximum height. Cuboids that fall outside the building plot are removed from the final geometry before it is evaluated. Since a 5 X 5 X 5 grid is used for the cuboids system, the projection of the cuboids grid coincides with the 5 X 5 grid inside the maximum plot square. This was a choice to provide clarity during the system design and convenience in the next stage of data formatting; however, it is also possible to use different-sized grids for the two systems.

### 4.4.3. Dataset optimization

To produce the final dataset, a total of 300 plot shapes were sampled using the grid method described in Boundary Conditions Sampling. Using these plot shapes, four different versions of datasets were created. Two conflicting considerations were the driving forces during this iterative process of dataset generation. First, the quality and granularity of data samples should be kept as high as possible to enable the training of an accurate and generalizable model which can be used for inference, as described in the next two chapters. Second, the computational cost should stay within reasonable limits for reasons of general efficiency and to allow the completion of this research in a timely manner. The four dataset versions are: v2.0, v2.01, v2.5, and v2.6[4].

*Dataset v2.0*

In the first version (v2.0), the 300 plot shapes were matched with one configuration of surrounding buildings each (m=300, n=1), using only the uniform sampling strategy, and resulting in a total of 300 unique boundary conditions. Each boundary condition was optimized using four objectives: solar gain ( 4.4 ), geometric objective ( 4.14 ), volume sampling ( 4.20 ), and area sampling ( 4.21 ). In addition, the area coverage was constrained below 70% and the volume

[4] The version numbered v1.0 was generated using a different underlying system and used in a pre-study (Ampanavos & Malkawi, 2021).

between 10% and 60% of the total site volume, using the constraints of Equations ( 4.23 ) and ( 4.24 ), to keep the generated geometries within reasonable limits.

$$G_{volume} = \max \begin{pmatrix} GeneratedVolume - 0.6 * SiteVolume, \\ 0.1 * SiteVolume - GeneratdVolume \end{pmatrix}, \quad G_{volume} \leq 0 \tag{4.23}$$

$$G_{area} = GeneratedArea - 0.7 * SiteArea, \quad G_{area} \leq 0 \tag{4.24}$$

All the objectives and constraints were computed after the application of the threshold th=0.35 to filter out smaller cuboids, as described in 4.2.3 Orientable Cuboids. During the calculation of the incident solar radiation, each cuboid-based geometry is passed to the simulator as a mesh object made up of disjoint boxes, each box having 12 triangular mesh faces. This means that "interior" mesh faces – faces in adjacent or intersecting cuboid groups that do not belong to the overall building envelope – were still present and contributed to the objective calculation. This decision was based on the assumption that their contribution would not significantly affect the optimization results, and motivated by a need to avoid the extra computational cost of further geometry processing. Each of the 300 optimizations was run for 200 generations with a population of 300 individuals. The optimization results for the 300 boundary conditions of this dataset were obtained in 7 days, 15 hours, 24 minutes, and 28 seconds of compute time using multiprocessing on a Windows machine with an Intel® Core™ i5-6500 CPU @ 3.20GHZ. The average processing time for a single optimization run was approximately 37 minutes.

*Dataset v2.01*

Next, the 300 boundary conditions of v2.0 were optimized with an additional objective $J_{areaHeight}$ ( 4.25 ), defined as the sum of the products of each mesh face's area (*FaceArea*) with the Z coordinate of the face's center (*FaceCenterZ*). The goal of this objective is to diversify the height distribution of the final solutions, which otherwise tend to extend all the way to the maximum.

$$J_{areaHeight} = \sum_{i=1}^{n_{faces}} FaceArea_i * FaceCenterZ_i \tag{4.25}$$

Additionally, a post-processing step was added to the cuboid-generated geometries. A Boolean union operation was applied to the disjoint boxes of each solution before the calculation of any objective or constraint. This step ensured that faces on the interior of the resulting building did not contribute to the solar gain objective. During this process, the Boolean operation contributed an excessive computational load. Two different Boolean backends were used through the Trimesh library: Blender (Blender Foundation, 2021) and OpenSCAD (Kintel & Wolf, 2021). OpenSCAD was up to ten times slower[5]. The cause of the time intensity originates with Trimesh library's Boolean implementation, which starts and terminates the 3rd party software during each single Boolean operation execution. To alleviate this problem, a C++ implementation of a mesh union operation was developed using the open-source library CGAL (*CGAL*, 2021) and compiled for use with Python. This offered a small speed improvement compared to using Blender through Trimesh. Each of the 300 optimizations was run for 200 generations with a population of 160 individuals. The optimization results were obtained in 20 days, 15 hours, 59 minutes, and 54 seconds of compute time, using multiprocessing on a Windows machine with an Intel Xeon CPU

[5] The greatest time increase was when executed on MacOS.

E5-2687W v3. The average processing time for a single optimization run was approximately 99 minutes.

To compare the solutions of the two datasets, the performance objectives of the optimal solutions acquired for v2.0 were recomputed after applying the Boolean union operation used in dataset v2.01 (Appendix B). Next, the optimized solutions of the two datasets that use the same simulation contexts were compared. A visual comparison of the solar gain-geometry volume distributions and a comparison of the hypervolumes with respect to the solar gain, volume, and area objectives did not show a clear advantage of using the union operation during optimization. The hypervolume for dataset v2.0 was found on average to be greater than that of dataset v2.01 by a small margin (MPE: 5%). Consequently, the next dataset iterations did not use the union operation, to avoid the increased computational cost.

### *Dataset v2.5*

For dataset v2.5, the 300 plot shapes of v2.0 were filtered for duplicates, and a subset of 228 unique plot shapes was selected. Each plot shape was combined with four surrounding building cases (m=228, n=4). The uniformly sampled obstructions of v2.0 were maintained, and three new cases were added using the zones of Figure 4.19. The subset of zone configurations that were used included either no zones or one of the four zone combinations: ab, bc, cd, da. Each zone *j* was assigned a height $h_j \in \{0.5 * h_{max}, 0.9 * h_{max}\}$, and each grid cell was sampled using Equation ( 4.23 ) with $e=0.1*h_{max}$. Each of the resulting 684 simulation contexts was optimized for 200 generations with a population of 200 individuals, using the same objectives, constraints, and general process as the one described for the dataset v2.0. Optimization results for 658 files were obtained in 6 days, 19 hours, 52 minutes, and 44 seconds (μ=15 minutes) on a Windows machine with an Intel Xeon CPU E5-2687W v3. Another 27 optimization results were obtained in 1 day, 2 hours, 20 minutes, and 45 seconds (μ=59 minutes) on a Windows machine with an Intel® Core™ i5-6500 CPU @ 3.20GHZ.

*Dataset v2.6*

Last, the dataset v2.6 was based on the 228 plot shapes of dataset v2.5, but the 3 zone-based configurations were replaced by 6 new configurations sampled from the following 11 options: either no obstructions, or any two or three-zone combination. The cells of each zone were sampled using Equation ( 4.23 ) with $e=0.05*h_{max}$, and each zone height was randomly sampled between zero and the maximum height $h_j \sim U(0, h_{max})$. The 1368 simulation contexts were optimized following v2.5. The optimization results were obtained in 14 days, 9 hours, 39 minutes, and 16 seconds of compute time (μ=15 minutes), on a Windows machine with an Intel Xeon CPU E5-2687W v3.

All optimizations were performed using the evolutionary algorithm NSGA II. With the exception of population size and maximum number of generations, all optimization parameters were kept at the Pymoo library's default values. The last generation of each optimization was selected to represent the set of optimally performing geometries for each problem.

The datasets v2.0, v2.5 and v2.6 were generated progressively, in response to limitations that were revealed after evaluating the processes of VAE learning and inference, detailed in the next two chapters.

Figure 4.23, Figure 4.24, and Figure 4.25 depict some examples of optimized geometries from dataset v2.6 and their corresponding performance graphs for three different boundary conditions. The parallel coordinates graphs on the right show the evaluated objectives for each geometry considered during the optimization. The optimal solutions (in brown color) for each boundary condition include geometries with volume and area coverage that span the whole allowed range. In addition, each boundary condition includes geometries of diverse morphology, as demonstrated with the rendered samples, also indicated by the breadth of the geometric objective evaluations.

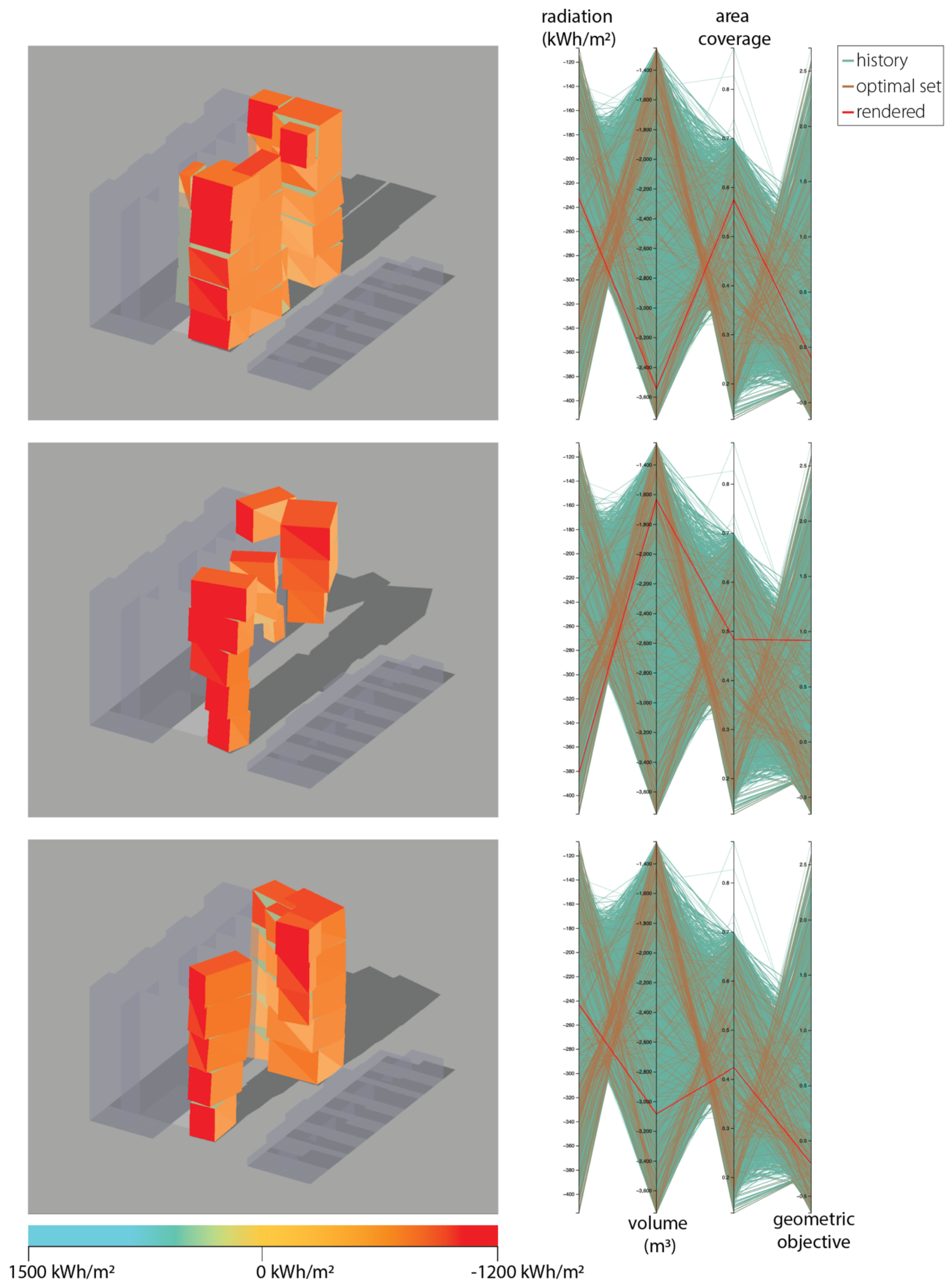


*Figure 4.23 Left: South-east isometric of optimization examples from dataset v2.6 from a single boundary condition (cond000).*

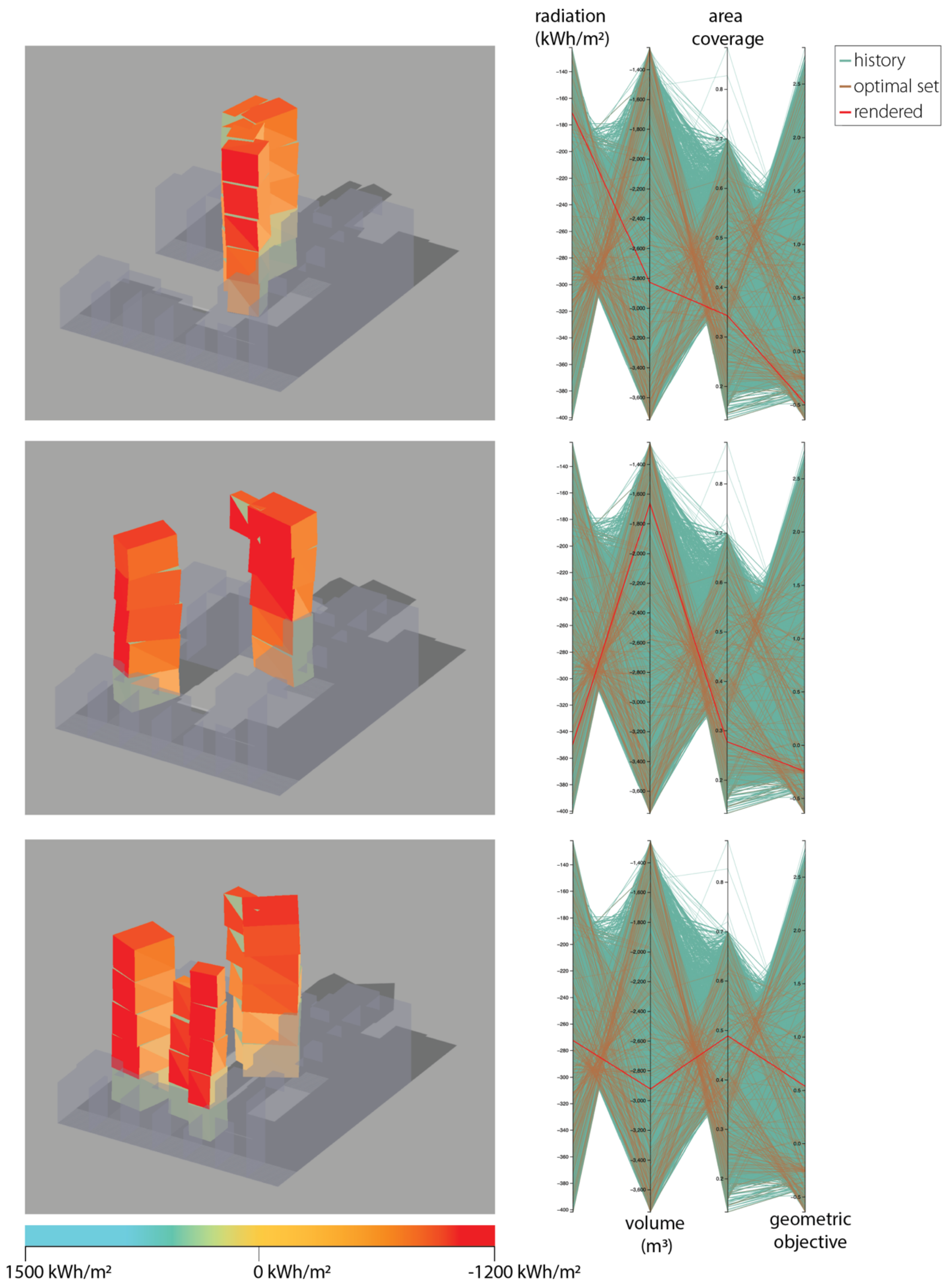


*Figure 4.24 South-east isometric of optimization examples from dataset v2.6 for boundary condition 'cond001'. The color-coding is the same as for Figure 4.23.*

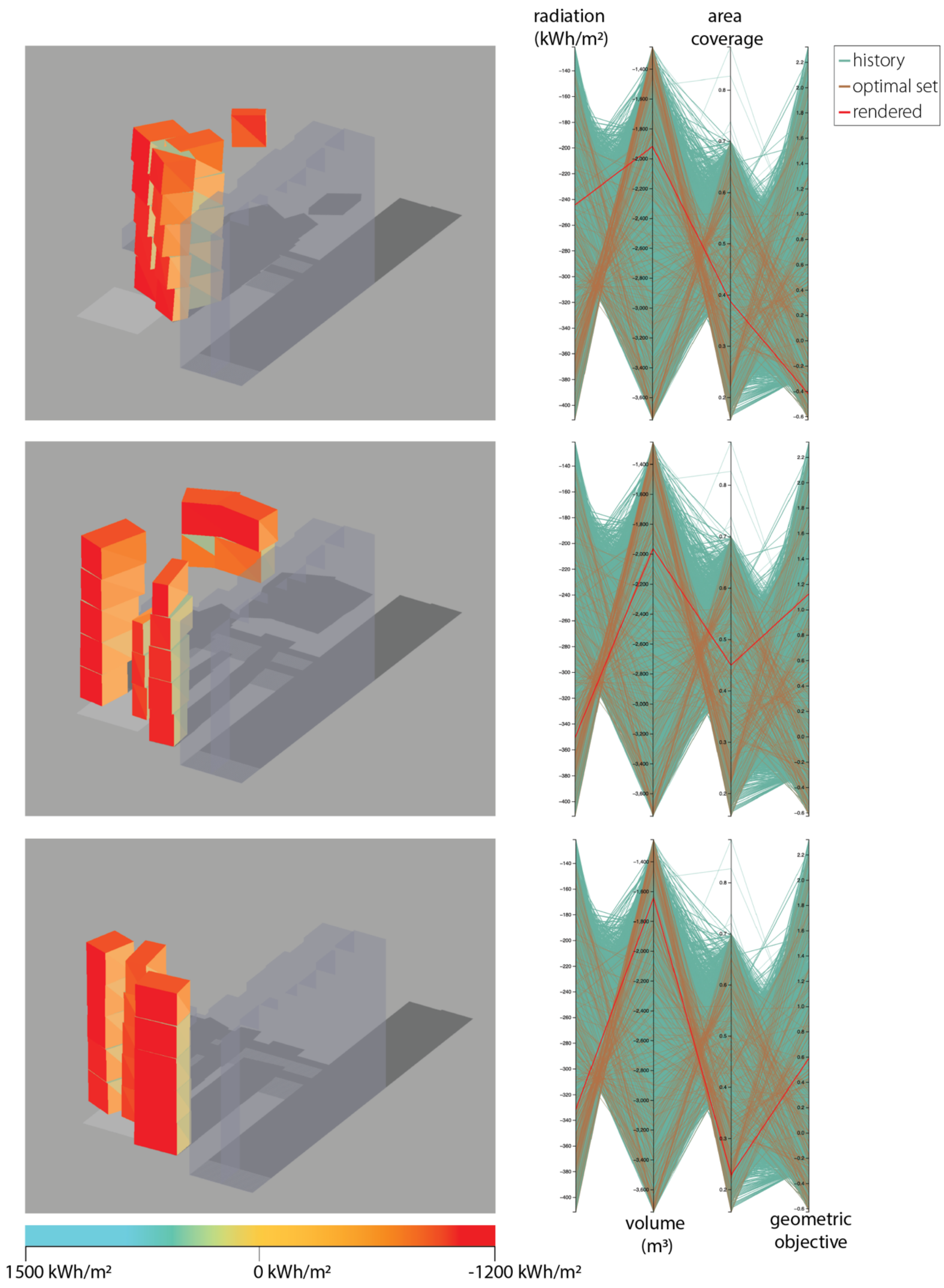


*Figure 4.25 South-east isometric of optimization examples from dataset v2.6 for boundary condition 'cond004'. The color-coding is the same as for Figure 4.23.*

## 4.5. Performance Evaluation

In order to further evaluate the quality of the results produced from the optimization process with respect to solar gain, a series of additional simulations were performed. First, a random subset from the optimized data was selected. In total 80 samples were drawn, coming from 10 unique boundary conditions. Any overlapping/interior segments of the mesh faces were removed by performing a Boolean union operation for all the individual cuboids.  Next, the mesh geometries were imported to the Grasshopper 3d environment where a series of incident radiation simulations were run for the cooling and heating periods. Mesh faces identified as floor slabs were removed from the simulated geometries. The simulations were performed using Ladybug's LB Incident Radiation component, with the same cumulative sky matrix and date ranges as the ones described in Performance objective formulation. The total incident radiation, the radiation intensity per face, and the individual face areas were recorded for each simulation.

The simulation results are compared against two sets of baselines. Each baseline geometry is generated as a cube with a volume equal to that of the respective optimized geometry. In the first baseline set, the cube geometry is placed centrally within the reference context, as demonstrated in Figure 4.26 Right. In the second set, no shading geometries are considered. The solar radiation simulations for the baselines were performed following the same process as for the optimized geometries.

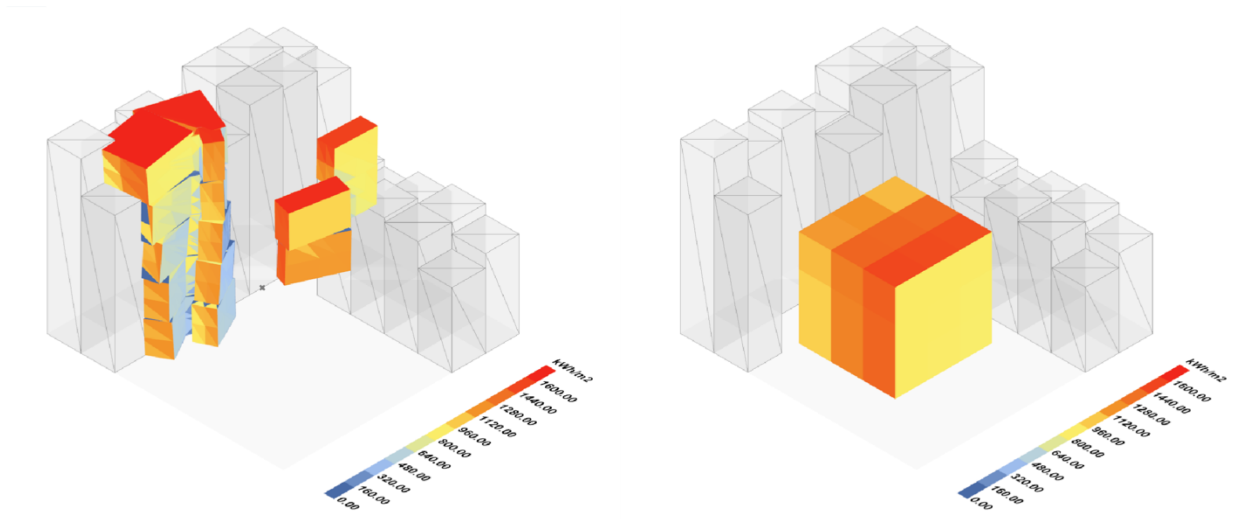


*Figure 4.26 Left, an optimized geometry, colored with incident radiation intensity values for the winter period. The shading geometries are outlined and shaded in gray. Right, a corresponding baseline with the same volume. Simulations were performed using Ladybug.*

Figure 4.27 shows the distribution of radiation intensity per square meter (m2) for a representative example from the simulated geometries. During winter, the optimized geometry includes faces with radiation intensity higher than both the shaded and the free-standing baselines, indicating a higher performance. During summer, the maximum radiation intensity on the optimized geometry is approximately the same as for the free-standing baseline. The area under the curve represents the total area of the geometry. The optimized geometry envelope area is larger than the baseline area, which contributes to a higher total radiation received during both winter (2,046,015kWh > 1,298,938kWh) and summer (529,821kWh > 354,789kWh) periods.

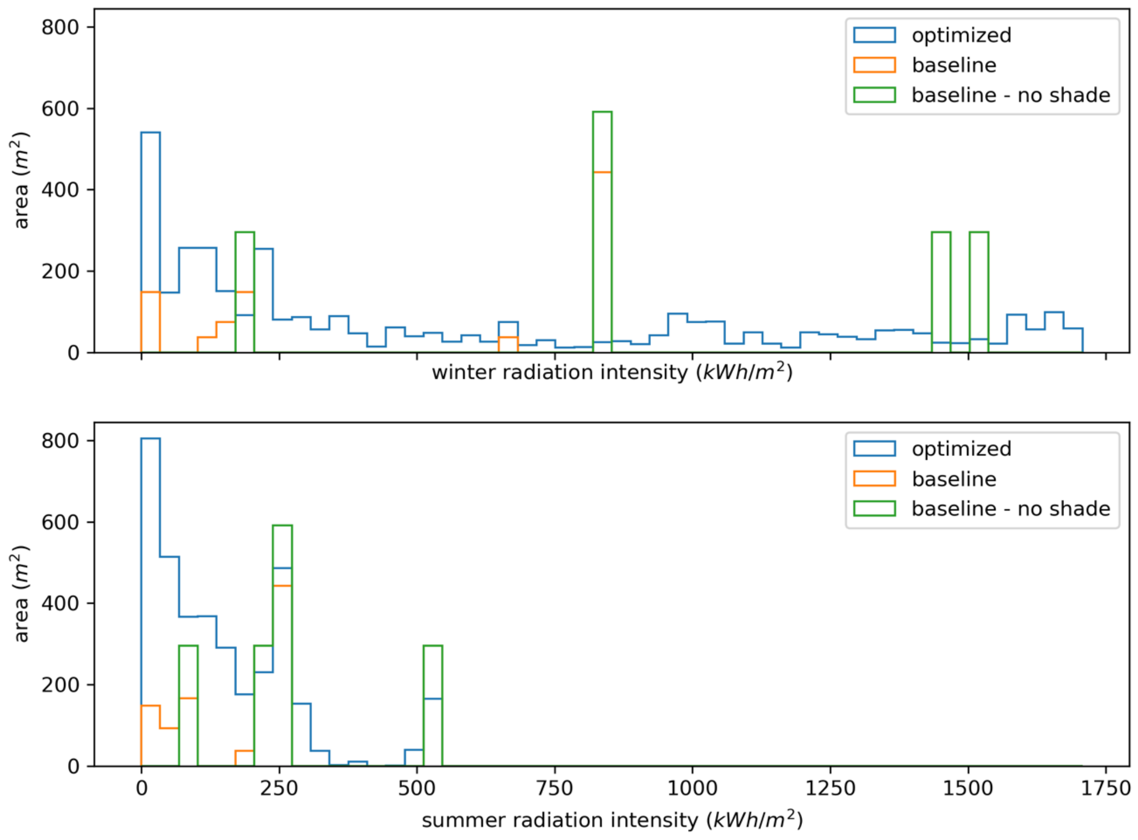


*Figure 4.27 Histograms of incident radiation intensity for heating (Top) and cooling (Bottom) periods for a single optimized geometry from dataset v2.6 and the two corresponding baselines.*

A similar pattern is observed for all 80 simulated samples compared to their corresponding baselines, as shown in Figure 4.28. During both summer and winter, the total incident radiation on the optimized geometries always exceeds that of the shaded baselines, and in most cases, it exceeds the radiation received by the free-standing baselines as well. While this is desired for the winter period, it creates adverse conditions during the summer period. The reason why this happens, even though the optimization objective considers both winter and summer periods, is that in the selected location, Boston, the heating period is much longer, resulting in higher absolute values for the heating objective than for the cooling objective. Therefore, the optimization process is primarily driven by the heating objective, while the cooling objective acts as a regularizer.

Indeed, a comparison of the beneficial, winter radiation to the harmful, summer radiation shows that the increase of beneficial radiation is much larger than the increase of harmful radiation (Figure 4.29). On average, the total incident radiation increased by 586,926 kWh for the winter period, an 83% increase compared to the shaded baselines. On the other hand, during summer, the total radiation increased only by 146,075 kWh on average, which is a 75% increase compared to the shaded baselines.

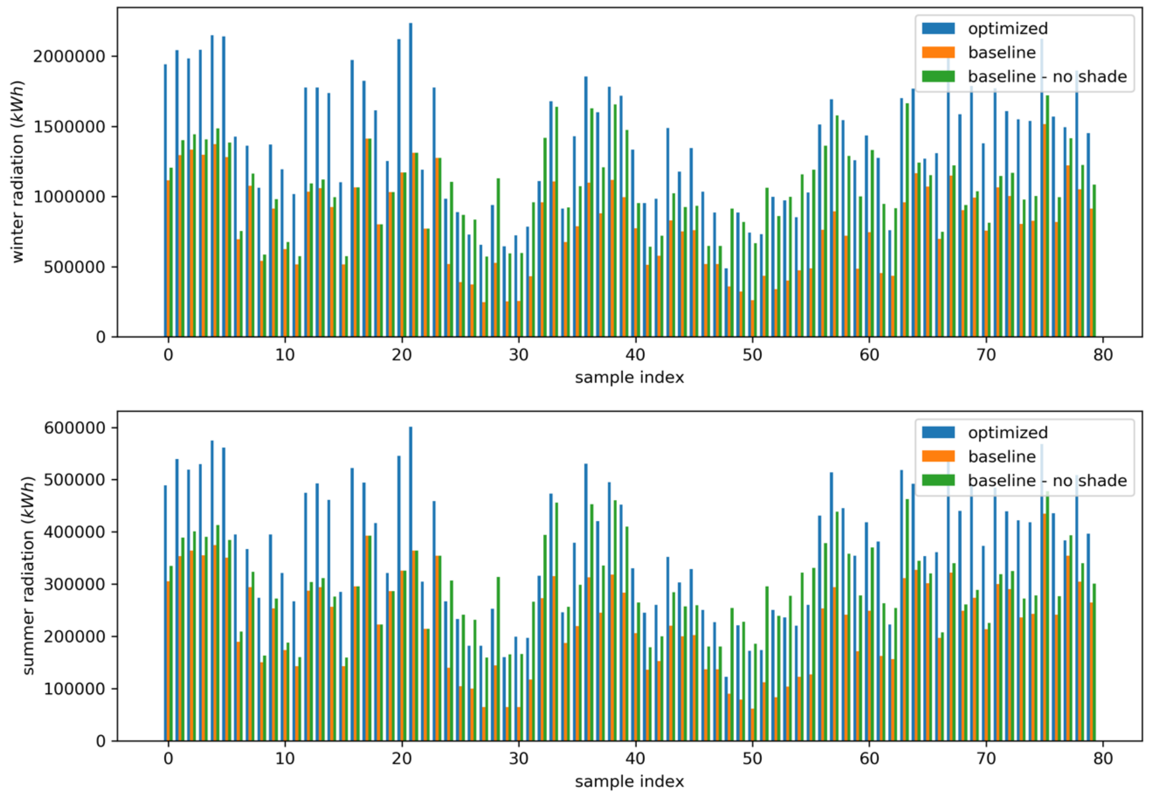


*Figure 4.28 Bar charts comparing the total incident radiation during the heating (Top) and cooling (Bottom) periods for 80 randomly selected optimized geometries from dataset v2.6, and their corresponding baselines.*

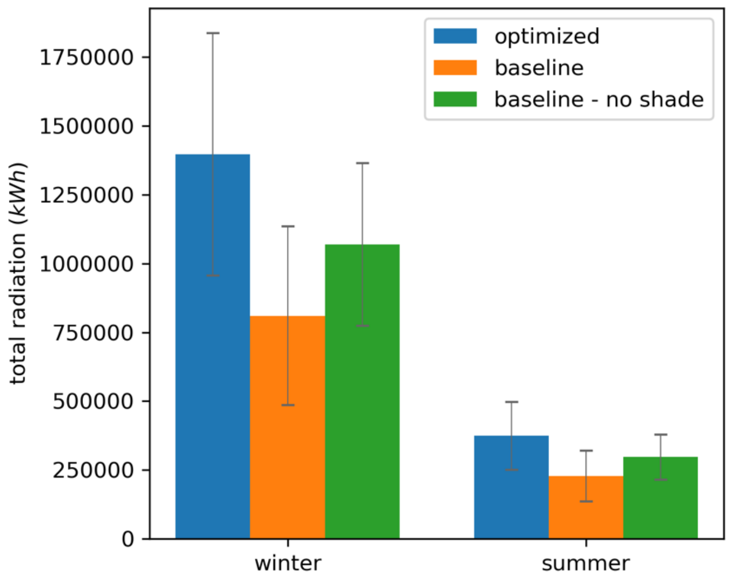


*Figure 4.29 Mean total incident radiation during heating (Left) and cooling (Right) periods for the 80 selected samples from dataset v2.6, and their corresponding baselines.*

# 5. Learning the Optimal Distribution

This chapter describes the process of creating a ML generative model to approximate the distribution of the optimized data produced in Chapter 4. For this task, a VAE is used, because of its natural ability to create a latent space. Mapping the data to a reduced, structured, and continuous space facilitates constrained sampling, which is necessary for the quick generation of geometric solutions, detailed in the next chapter.

This chapter starts with a brief introduction to the VAE and its training process. The next two sections discuss data formatting and model architecture. Right after, loss functions for the different data types in the dataset are introduced. A set of loss terms related to performance aspects of the geometries are suggested and incorporated into the loss function. The next section discusses the training process, comparing the different dataset versions and detailing aspects of the model's hyperparameter tuning. More specifically, the learning rate, the weights for the various loss terms, and the effect of the performance-related losses are detailed. Finally, the trained model is evaluated on the test set reconstruction using error metrics, building performance simulations, and visual comparison of examples.

Figure 5.1 outlines the whole process depicting its iterative nature. At various stages, training results inform the dataset generation, data formatting, model architecture, loss functions, and hyperparameters.

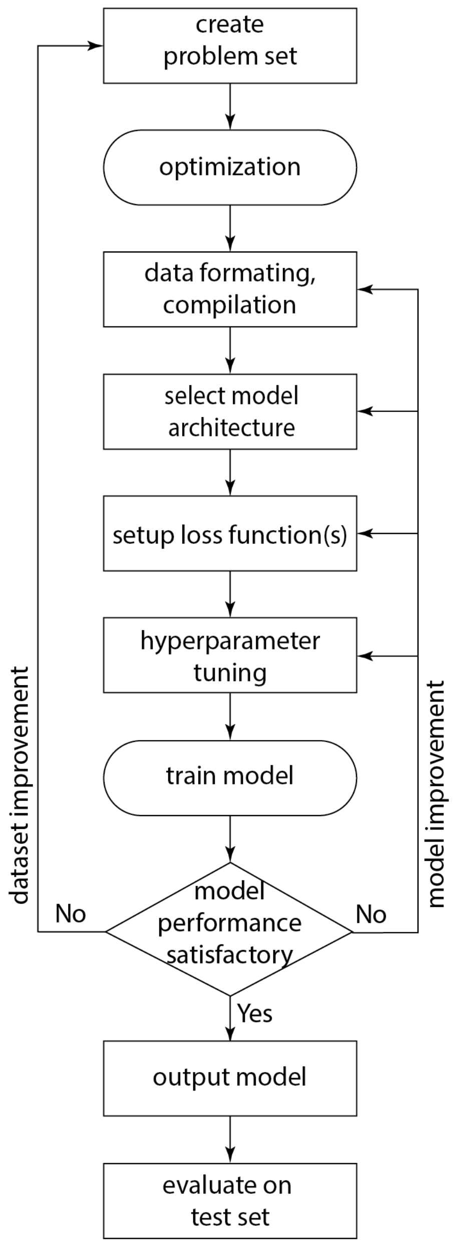


*Figure 5.1 The iterative process of model training.*

## 5.1. VAE – Intro

The VAE (Kingma & Welling, 2014) is a probabilistic model consisting of an encoder function E(x)=z and a decoder function D(z)=x. The encoder E maps the original data to a lower-dimensional space, the latent space, by estimating a probability distribution. The decoder function D performs the inverse task, mapping samples from the latent space to the original data space. Both the encoder E and the decoder D are approximated by neural networks trained in a self-supervised way, on the task of data reconstruction. Equation ( 5.1 ) shows the three steps for data reconstruction through the VAE[6], where x is the data, $z_\mu$ and $z_\sigma$ are the mean and standard deviation of a Gaussian distribution, and x' is the reconstruction of x.

$$E(x) = (z_\mu , z_\sigma), \qquad z \sim \mathcal{N}(z_\mu , z_\sigma), \qquad D(z) = x' \tag{5.1}$$

The training objective is defined as the data reconstruction loss plus a regularizer. The reconstruction loss encourages the decoder to learn to reconstruct the data. The regularizer is typically the Kullback-Leibler divergence (KL-divergence) of the approximate posterior (i.e. the encoder's distribution) from the prior (commonly chosen as a zero-centered Gaussian distribution) (Kingma & Welling, 2014). After a VAE has been trained, the decoder function can be isolated and used as a generative model that samples the latent space and generates data instances from the original distribution.  In this work, the decoder can be used to generate samples of boundary conditions and corresponding optimal geometries.

## 5.2. Data Formatting

The first step for using a ML model is to appropriately format the data, i.e. to convert the boundary conditions and corresponding geometries to a vectorized form. The plot shape is by nature two-dimensional, and the surrounding buildings can be easily represented by a two-

[6] In practice, training the VAE is possible using the re-parametrization trick that enables backpropagation of the gradients through the approximate posterior distribution sampling step (Kingma & Welling, 2014).

dimensional height map. The two elements are combined into a single two-dimensional representation with two channels and resolution 7 X 7. One channel contains binary values for the plot shape mask, the other contains continuous values for the surrounding buildings' heightmap.

On the other hand, the optimized geometries are three-dimensional. In ML, there are three primary ways to describe geometric data: single or multi-view images, voxels, and point clouds (Lun et al., 2017). More recently, signed Distance Functions (SDF) have also been used with 3d generative models (Kleineberg et al., 2020; Park et al., 2019). In this work, the optimized geometries in the dataset have been created as cuboid elements on a three-dimensional grid, resembling some aspects of a voxel representation. However, in contrast to voxels that are fixed elements and only have one channel containing binary values, each element in the Orientable Cuboids representation is transformable, with multiple degrees of freedom. Therefore, each degree of freedom is represented in a separate channel, taking continuous values between zero and one.

During optimization, seven parameters were used to fully define a cuboid: three parameters representing rotation (Euler angles), and four parameters representing anisotropic scaling (Figure 4.9). However, the distance between vectors representing Euler angles may not be representative of the actual angle difference, making the respective loss estimation during model training problematic. Therefore, the 3d rotations of the cuboids were instead represented using the cosine and sine of the Euler angles, which avoid angle representation discontinuities around 0, due to their periodic nature. After this transformation, the 3-dimensional representation used for the ML model has ten channels: six representing the rotation and four representing the anisotropic scaling parameters. The resolution of the 3d grid is 5 X 5 X 5, reflecting the resolution of the Orientable Cuboids used during the optimization process.

The two-dimensional boundary conditions and the three-dimensional geometries can be jointly used without further processing with a multi-input, multi-output VAE. Alternatively, they can be merged into a single, three-dimensional representation. In order to merge the two, the 5 X 5 X 5 X 10 (x, y, z, channels) tensors representing the geometries are first padded with zeroes to match the footprint of the boundary conditions, resulting in 7 X 7 X 5 X 10 tensors. Then, the 7 X 7 X 2 tensors representing the boundary conditions are similarly expanded along the Z-axis with zeroes, resulting in 7 X 7 X 5 X 2 tensors. The two tensors are concatenated into a single tensor with 12 channels in total.

## 5.3. Model Architecture

A multi-input, multi-output encoder can be created as shown in Figure 5.2, where the geometries and the boundary condition inputs are concatenated into a single, one-dimensional vector after they have passed through a series of 3d or 2d convolutions respectively. The concatenated vector is projected to the latent space through one or more fully connected layers. The decoder follows a mirrored architecture. This architecture can be particularly useful for problems where the boundary conditions and the solutions (geometries) represent quantities of different nature. However, in this work merging the boundary conditions and the solutions into a single representation is easily achievable. Furthermore, the boundary condition has a spatial structure that directly affects that of the solutions. Therefore, a single, three-dimensional representation was selected (Figure 5.3), since it enables information from both the boundary conditions and the solutions to be processed jointly throughout the whole neural network, allowing their relationships to be better identified. Early experimentation results were in favor of the single input-output model as well.

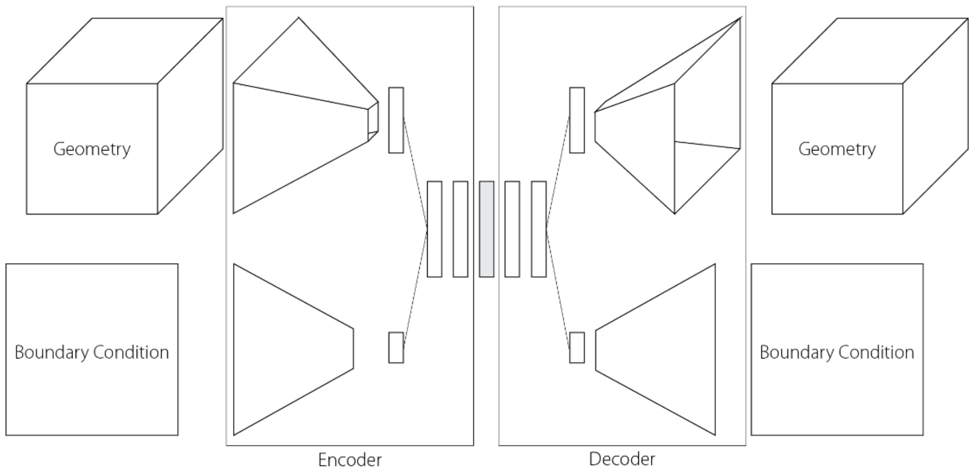


*Figure 5.2 Multiple input-output VAE model.*

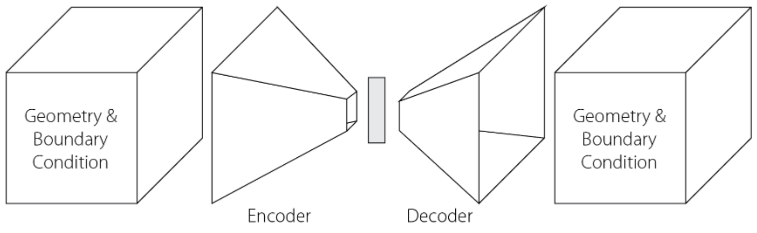


*Figure 5.3 Single input-output VAE model.*

The VAE that was used has an encoder consisting of three 3d convolutional layers followed by a flattened layer and a fully connected layer with size twice the dimensionality of the latent space. Each convolution has kernel size 3 with step size 1, and is followed by a ReLu activation and batch normalization (Ioffe & Szegedy, 2015). Each convolution has 16, 32, and 64 filters respectively. The decoder consists of a fully connected layer with 7,840 units that gets reshaped to a 7 X 7 X 5 X 32 tensor and is followed by three 3d transpose-convolutional layers and an output layer. Similar to the encoder, each transpose convolution has kernel size 3 with step 1, and is followed by a ReLu activation and batch normalization. The number of filters are 64, 32, and 16 respectively. The output layer is a transpose convolution layer with kernel size 1, step 1, and 12 channels. During inference, the output passes through a sigmoid function. Latent spaces with 64, 128, or 256 dimensions were used.

## 5.4. Loss Function

### 5.4.1. Per-data type loss

Each datapoint of the dataset contains information of three different kinds. First, the optimal geometries take continuous values between zero and one [0, 1]. Second, the surrounding buildings heightmap also takes continuous values between zero and one [0, 1]. Third, the plot shape mask takes binary values {0, 1}. Consequently, the reconstruction loss was split into three terms where each refers to the reconstruction loss of the particular data, as shown in Equation ( 5.2 ). In each case, the data type and likelihood distribution determine the loss functions that can be used. For the plot shape mask, the binary cross-entropy was used, which corresponds to the log-likelihood of Bernoulli distributed data. For the geometries, the selected loss function was the binary cross-entropy with a normalizing constant, which corresponds to the log-likelihood of a continuous Bernoulli distribution, introduced by Loaiza-Ganem and Cunningham (2019) for continuous [0, 1] data. Alternatively, the use of a structural error (SSID) (Wang et al., 2004) was explored. The use of SSID in generative networks has been suggested as a superior loss function before (Ghojogh et al., 2020; Kancharla & Channappayya, 2018; Parimala & Channappayya, 2019); however, it failed to produce good results here due to the low resolution of the dataset. Finally, the continuous Bernoulli log-likelihood and the simpler L1 distance worked equally well for the obstruction heightmap data, and the L1 distance was chosen for the final versions of the model.

$$Loss_{reconstruction} = Loss_{geometries} + Loss_{plotShape} + Loss_{surroundings} \tag{5.2}$$

For the plot shape and surrounding buildings heightmap channels, the loss is calculated on the single XY plane with z=0, to facilitate the training and avoid unnecessary gradients from the 'filler' data, which is anyway ignored in the interpretation of the reconstructions.

### 5.4.2. Masking

During optimization, the plot shape mask is used to cull out all cuboids that fall outside of the buildable site. The same technique is applied to the data generated by the VAE. The predicted plot shape is used as a mask for the predicted geometry, setting to zero all values that fall outside the building plot. By calculating the geometry loss after the mask has been applied, unnecessary gradients over invalid cuboids are avoided and training becomes more efficient. The same masking process is also applied to the surrounding buildings heightmap channel.

### 5.4.3. Performance Loss

The reconstruction loss of Equation ( *5.2* ) forces the VAE to learn how to encode and decode data to and from the latent space, preserving all the geometric information during the process. However, it is also important to ensure that the building performance of the geometries is preserved. In this work, building performance can be interpreted as a derived property of the data.

In general, similar data points are expected to have similar performances, and a data point that is reconstructed through the trained VAE should have the same performance as the original data.

In practice, however, it may be difficult to achieve satisfactory matching of performance when the VAE is trained solely using a generic metric of the geometric parameters' error. In order to focus the training on the specific aspects of the geometries that affect their performance, a set of additional, performance-based error metrics can be used. One constraint during the formulation of such metrics is that they need to be calculated through differentiable functions. This is due to the nature of the training process of neural networks, which relies on the backpropagation of the loss function's gradients.

In this work, three quantities of interest are identified: the solar gain performance ( 4.4 ), the volume, and the area coverage of the geometries. Raytracing-based solar radiation simulations are not differentiable and cannot be used in a loss function. One way to overcome this problem is to create a differentiable surrogate model for the solar gain calculation. The robustness of such a surrogate model is crucial to ensure that it can lead the generated geometries towards forms of desired performance. A neural network could be used as such a surrogate. However, training this model would require a synthetic dataset containing a variety of forms with a wider range of performances than those of the optimized data that comprise the current datasets. After further consideration and evaluation of the VAE reconstructions without such a solar radiation loss term, potential benefits of developing a solar radiation surrogate were considered minimal in the scope of this work, and this path was not explored further. The viability of a surrogate for the purpose of improving the VAE reconstructions performance remains an open question for future research.

On the other hand, the calculation of the building geometry volume using the geometric parameters contained in the dataset is easy to achieve in a straightforward way. Equation ( 5.3 ) shows the function to calculate the cuboid-based geometry volume, where w, l, h are the width, length, height parameters of a cuboid, and $w_{Ai}$, $w_{Bi}$, $l_{Ai}$, $l_{Bi}$ are the anisotropic width and length scaling parameters – detailed in Figure 4.9 – contained in channels 6–10 of the data. The normalizing factor $c_{vol}$ accounts for the data scaling and can be dropped during training.

$$V = c_{vol} * \sum_{i=0}^{cuboidsBuilt} wi * li * h_i = c_{vol} * \sum_{i=0}^{cuboidsBuilt} (w_{Ai} + w_{Bi}) * (l_{Ai} + l_{Bi}) * h_i, \quad h_i = 1.0 \tag{5.3}$$

In contrast to the volume, there is no mathematical formula to calculate the projected area of the orientable cuboids' geometries. However, if the rotation angles are disregarded, under the assumption that off-plane rotations from the XY plane are typically small in the dataset, a simple differentiable approximation of the projected area is possible. An observation of the geometric forms included in the dataset (Figure 4.23, Figure 4.24, Figure 4.25), confirms that this assumption is safe to make. Then, the projected area $A_{proj}$ can be approximated as the sum of the cuboids' base areas *AreaWL* (Equation ( 5.5 )) of the largest cuboids from each column (i, j) of the three-dimensional grid where i, j, k are the indices in X, Y, Z (Equation ( 5.4 )).

$$A_{proj} = \sum_{i=j=0}^{4} \max\left(AreaWL\big(cuboid_{ijk}\big), \qquad k = \{0, 1, 2, 3, 4\}\right) \tag{5.4}$$

$$AreaWL(cuboid_i) = wi * li = (w_{Ai} + w_{Bi}) * (l_{Ai} + l_{Bi}) \tag{5.5}$$

During optimization, cuboids that were scaled below a predetermined threshold *th* were removed from the resulting geometry. In a similar way, geometries generated by the VAE are filtered with the same threshold. The filter is not applied during the calculation of the geometry loss, to ensure smoother gradients. In contrast, the performance-related losses are calculated after the filter has been applied, to reflect the actual performance of the generated geometries. The loss function for the volume *Loss<sub>vol</sub>*

The loss function for the volume $Loss_{vol}$ and the loss function for the area projection $Loss_{area}$ are described in Equations ( 5.6 ) and ( 5.7 ), where y is the original datapoint and x the reconstruction, $mask_{predictedSite}$ is the predicted site mask detailed in Equation ( 5.8 ), and $mask_{threshold}$ is the threshold-derived mask where th=0.35 reflects the optimization process hyperparameter th, as detailed in Equation ( 5.9 ).

$$Loss_{vol}(x,y) = \Big(V(y)) - V\big(x * \ mask_{predictedSite}(x) * \ mask_{threshold}(x)\big)\Big)^2 \tag{5.6}$$

$$\begin{aligned} Loss_{area}&(x,y) \\ &= \Big(A_{proj}(y)) - A_{proj}\big(x * mask_{predictedSite}(x) \\ &\quad * \ mask_{threshold}(x)\big)\Big)^2 \end{aligned} \tag{5.7}$$

$$mask_{predictedSite}(x) = \begin{bmatrix} 0 & if \ x_{plotShape_i} < 0.5 \\ 1 & if \ x_{plotShape_i} \geq 0.5 \end{bmatrix} \tag{5.8}$$

$$mask_{threshold}(x) = \begin{bmatrix} 0 & if \ x_i < th \\ 1 & if \ x_i \geq th \end{bmatrix}, \quad th = 0.35 \tag{5.9}$$

### 5.4.4. Beta VAE

To achieve better generalization and latent space disentangling, a beta-VAE (Higgins et al., 2016) was used instead of the original VAE, with beta = 2.0. Therefore, the loss function takes the form of Equation ( 5.10 ).

$$Loss = 2.0 * Loss_{KL} + Loss_{geometry} + Loss_{plotShape} + Loss_{surroundings} + 1e - 4 * Loss_{vol} + 10.0 * Loss_{area} \quad (5.10)$$

## 5.5. Training

Each dataset was split into three parts: training, validation, and test set. Samples that share the same boundary condition were not separated, so that each boundary condition only appears in one of the three subsets. The exact division is shown in Table 5.1 below. The test set is used to report results in the next subsection (Training: Dataset Comparison) and the final section of this chapter (Evaluation). The validation set was used during the model architecture selection and hyperparameter tuning. All results of this section (Training) except for the Dataset Comparison are evaluated using the validation set.

*Table 5.1*

| | Training samples | Training conditions | Validation samples | Validation conditions | Test samples | Test conditions |
|---|---|---|---|---|---|---|
| v2.0 | 61,200 | 204 | 15,300 | 51 | 13,500 | 45 |
| v2.5 | 167,400 | 744 | 18,000 | 80 | 19,800 | 88 |
| v2.6 | 262,500 | 1,225 | 28,500 | 133 | 51,000 | 238 |

The VAE model was implemented using the Python library TensorFlow (Abadi et al., 2016) and trained using the Adam optimizer (Kingma & Ba, 2017) with batch size 128. The loss was calculated as the single sample Monte Carlo estimate of the expectation (*Convolutional Variational Autoencoder | TensorFlow Core*, n.d.), using the loss function of Equation ( 5.10 ). The model was trained using Google Cloud Services on a NVIDIA Tesla V100-SXM2-16GB GPU for 400 epochs with dataset v2.0, or 200 epochs with datasets v2.5 and v2.6, for time durations ranging from two to two and a half hours.

### 5.5.1. Dataset Comparison

This subsection explains the iterative process and the reasoning behind the creation of the three dataset versions. Each of the later versions was created in response to limitations of the previous one, which were revealed after training the VAE model. The main differences between the three dataset versions, introduced in Chapter 4: Dataset Optimization, are summarized in

Table 5.2 for clarity.

*Table 5.2*

| | Number of plot shapes | Obstructions sampling method | Pool of zone combinations | Obstructions heights values (normalized) |
|---|---|---|---|---|
| v2.0 | 300 | 1 uniform | 0 | [0, 1] |
| v2.5 | 228 | 1 uniform + 3 zone-based | 5 | {0, [0.4, 0.6], [0.8, 1.0]} |
| v2.6 | 228 | 1 uniform + 6 zone-based | 11 | [0, 1] |

A VAE trained using dataset v2.0 could reconstruct data samples in a convincing way, regarding their plot shapes and geometries. However, it failed to produce a good reconstruction of the data samples' obstructions, even after extensive experimentation with hyperparameters and network architecture. This failure is attributed to the difficulty of reproducing the heightmap values drawn from independent uniform distributions. Furthermore, the VAE was not able to reconstruct obstructions with more structured shapes, since no such examples were included in the dataset.

This led to the creation and use of dataset v2.5. A VAE trained on dataset v2.5 achieved a significant improvement in reconstructing obstructions while maintaining or improving the reconstruction quality for plot shapes and geometries. The addition of structured obstructions in dataset v2.5 did improve not only the reconstruction of similar well-defined shapes but also the reconstruction error on the uniformly sampled obstructions of dataset v2.0. However, the restricted sampling of heights and arrangements of obstructions for dataset v2.5 did not allow good generalization to off-domain cases. Obstructions with normalized heights different than 0, 0.5, or 1 and obstruction configurations not in the dataset could not be accurately reproduced.

Finally, dataset v2.6 was created and used to alleviate the previous limitations. A VAE trained on dataset v2.6 improved the reconstruction error not only for the v2.6 test set obstructions but also for the test set obstructions of v2.5.

Table 5.3 shows the mean absolute error (MAE) of obstructions, for VAE models trained on each of the three datasets and tested on all the datasets. Similarly, Table 5.4 shows the MAE of plot shape reconstructions. The VAE trained on dataset v2.6 achieves the best overall performance. The VAE trained on v2.0 is evaluated at 250 epochs and the models for v2.5 and 2.6 for 200 epochs, where their training seems to have converged (Figure 5.4).

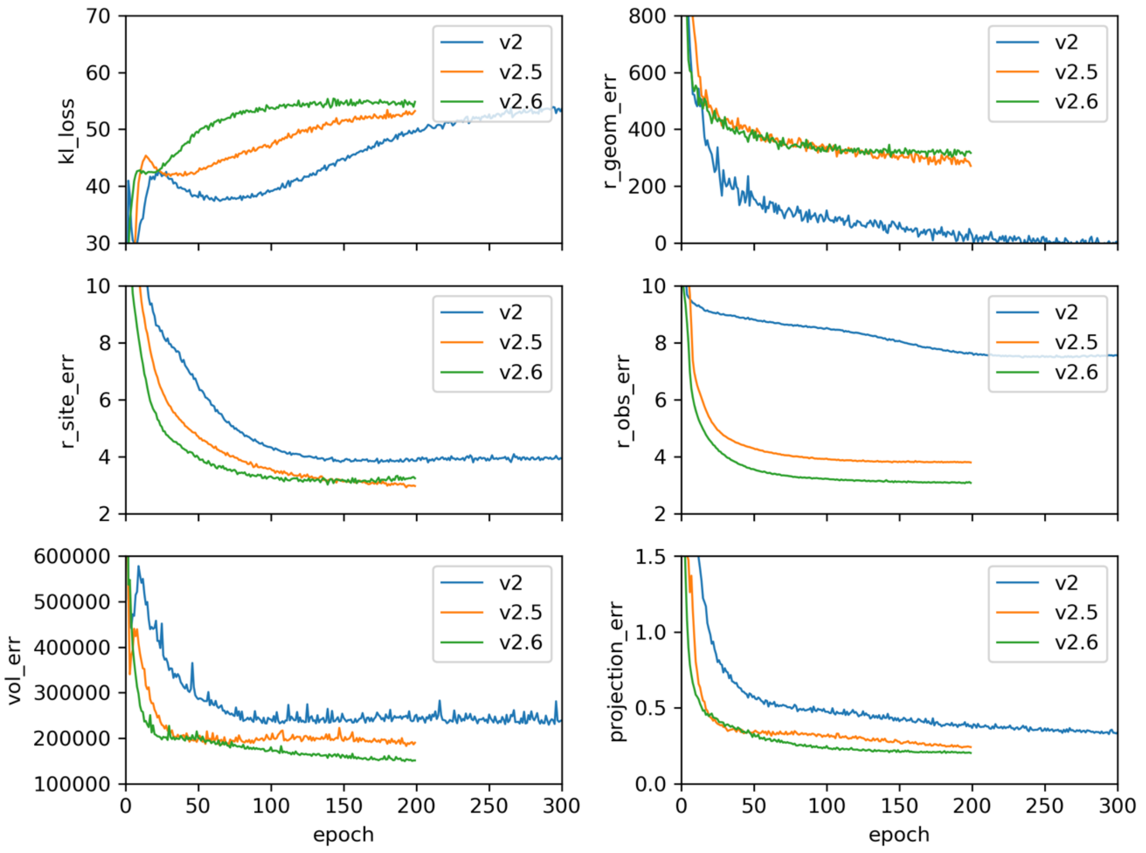


*Figure 5.4 Convergence graphs of validation sets for VAE models trained on dataset v2.0, v2.5, and v2.6. Individual error terms are plotted in each graph. These graphs only show convergence and are not directly comparable since the loss is evaluated on a different dataset for each model.*

*Table 5.3*

| Test set error for obstructions | Trained on v2.0 | Trained on v2.5 | Trained on v2.6 |
|---|---|---|---|
| MAE on v2.0 | 0.232 | **0.166** | 0.185 |
| MAE on v2.5 | 0.250 | 0.112 | **0.105** |
| MAE on v2.6 | 0.256 | 0.135 | **0.089** |

*Table 5.4*

| Test set error for plot shape | Trained on v2.0 | Trained on v2.5 | Trained on v2.6 |
|---|---|---|---|
| MAE on v2.0 | 0.036 | 0.014 | **0.013** |
| MAE on v2.5 | 0.046 | 0.030 | 0.030 |
| MAE on v2.6 | 0.051 | 0.031 | 0.031 |

Following the layout of Table 5.3, Figure 5.5 demonstrates examples of obstruction heightmap reconstructions using VAEs trained on dataset v2.0 (model-v2.0), dataset v2.5 (model-v2.5), and dataset v2.6 (model-v2.6). Each row contains a sample from datasets v2.0, 2.5, and 2.6, and the three reconstructions.

In the top row, the reconstruction using model-v2.0 stylistically resembles the original; however, it fails to accurately reproduce individual values. In the second and third rows, the model-v2.0 wrongfully generates heightmaps in empty areas of the original sample. In contrast, model-v2.5 did not preserve the checkered appearance of dataset v2.0 samples, but it created reconstructions where neighboring values of the obstructions have been averaged. Figure 5.5 Top-column 3 shows how brighter areas (top left and bottom right) and darker areas (bottom left and top right) have been overall preserved even though in a blurry form. Model-v2.5 accurately reconstructed the dataset v2.5 and improved the reconstructions of dataset v2.6. Finally, model-v2.6 had similar results for dataset v2.0 (Figure 5.5 Top Right) and dataset v2.5 (Figure 5.5 Center Right). In addition, it better reconstructed the more complex zone-based obstructions of dataset v2.6 (Figure 5.5 last row).

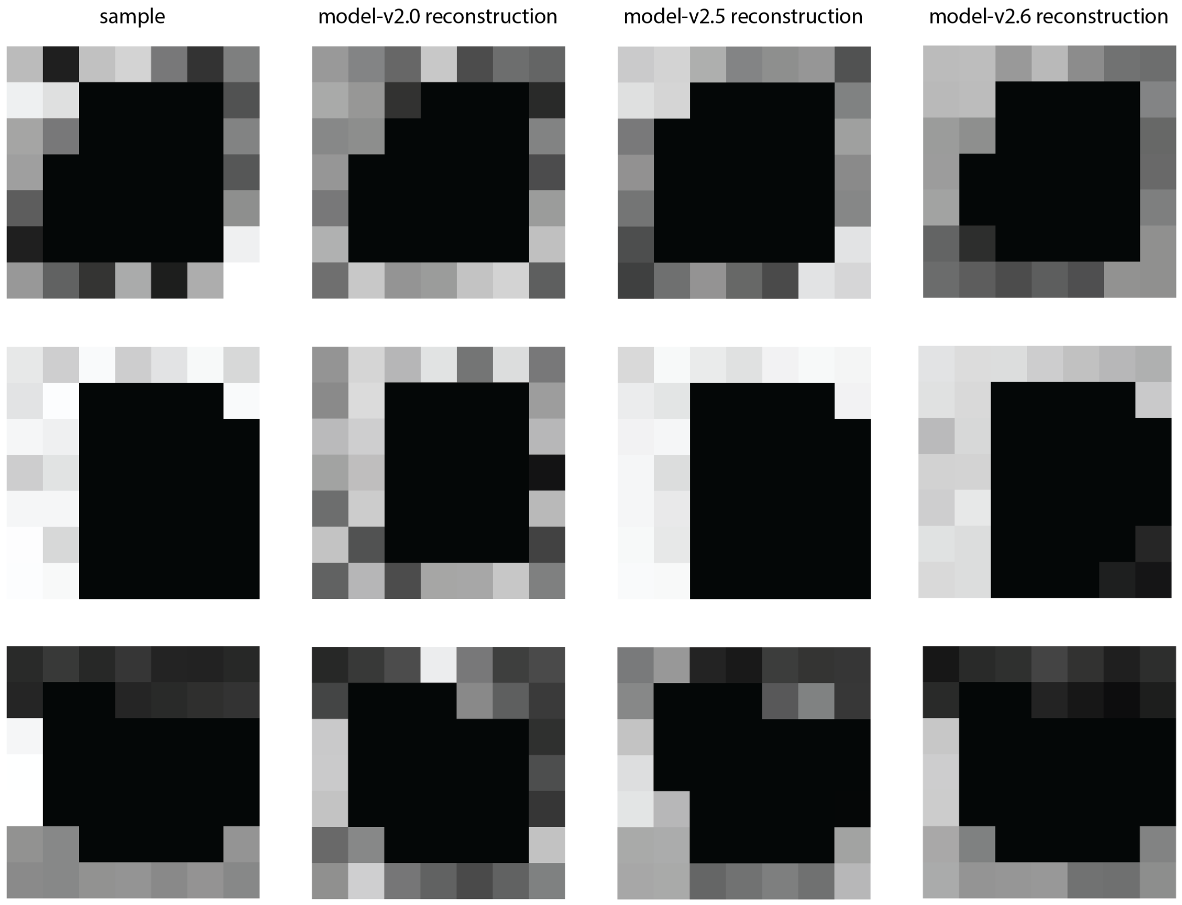


*Figure 5.5 Left column: obstruction heightmap samples drawn from the test sets of dataset v2.0 (Top), dataset v2.5 (Center), and dataset v2.6 (Bottom). Columns 2-4: reconstructions using models trained on datasets v2.0, v2.5, and v2.6, in order.*

Given the better reconstruction quality of the boundary conditions when training on dataset v2.6, this dataset was selected. Consequently, all results reported in the next sections have been evaluated using the dataset v2.6.

### 5.5.2. Hyperparameter Tuning

Several parameters regarding the model architecture and the training process were explored and the best ones were picked based on model evaluations using the validation set. This process is often called hyperparameter tuning, and may involve the number of convolutional filters and kernel sizes, the learning rate, the loss function tuning, etc... Next, more details are provided regarding the learning rate and the tuning of the loss function.

#### *5.5.2.1. Learning Rate*

During the hyperparameter tuning, the various terms of the validation loss were recorded separately to provide better insights into the process. A comparison of Figure 5.6 and Figure 5.7 demonstrates why this is important. The training graphs for two models with different learning rates are shown. Figure 5.6 shows the Evidence Lower Bound (ELBO), which equals to the negative total loss. Both the training and the validation ELBO show that training has converged at around 200 epochs without signs of overfitting for both models. The model with the higher learning rate may also have converged at a better point since it has a slightly higher ELBO. However, a closer inspection of the individual error terms in Figure 5.7 reveals different information. The boundary condition terms (plot shape and obstructions reconstruction errors) of the model with the higher learning rate start overfitting at around 20 epochs and their error keeps going up beyond that point (Figure 5.7 second row). However, the rest of the loss terms included in the ELBO obscure this fact. On the other hand, a lower learning rate marginally increases the loss for the geometries but prevents overfitting and leads to a lower loss for the boundary conditions. An accurate representation of the boundary conditions in the latent space is important to enable a meaningful navigation of the space and retrieval of the appropriate geometries. Therefore, the best choice is to use either the higher learning rate for 20 training epochs or the lower learning rate for 200 epochs. The second option was chosen, because it achieves a lower boundary condition error, with a slightly higher but more stable geometric error.

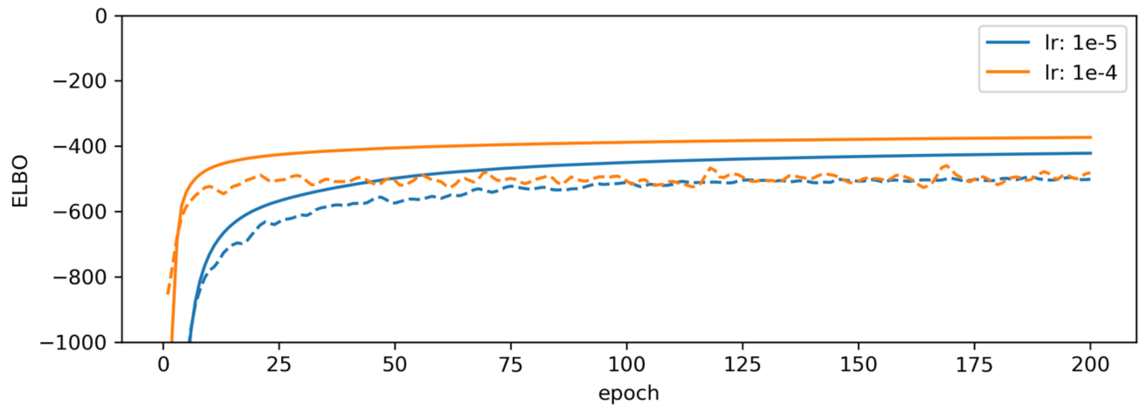


*Figure 5.6 Training (solid) and validation (dashed) negative loss (ELBO) during the training of the same VAE with different learning rates. Both models appear to be converging smoothly.*

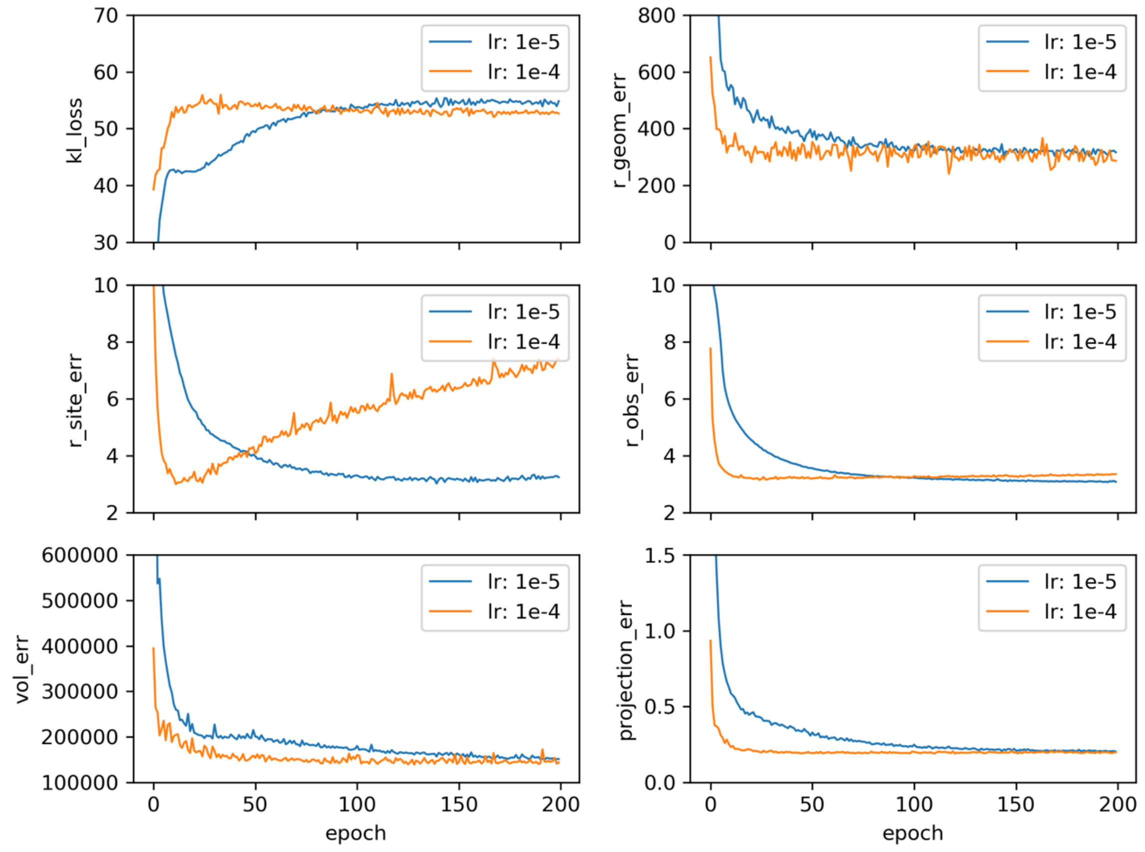


*Figure 5.7 Breakdown of individual loss terms on validation sets of the two VAEs from Figure 5.6. The VAE with the higher learning rate overfits to the plot shape channel (Center Left) and, to a lesser extent, to the obstructions channel (Center Right).*

### *5.5.2.2. Loss Term Weights*

In the loss function defined in Equation ( 5.10 ) the loss terms for the derived attributes of volume and area have been weighted to match the common numerical range of the geometry loss. However, the per-channel loss terms regarding the geometry, obstructions, and plot shape were not initially weighted. It was found that the boundary condition reconstruction error could be improved with appropriate weighting of the respective losses. Therefore, the loss takes the more generalized form of Equation ( 5.11 ), using the values from Equation ( 5.12 ). Figure 5.8 shows the effect of these weights on the plot shape and obstructions errors during training.

$$\begin{aligned} Loss = \ & beta * Loss_{KL} + f_{geometry} * Loss_{geometry} + f_{plotShape} * Loss_{plotShape} \\ & + f_{surroundings} * Loss_{surroundings} + f_{vol} * Loss_{vol} + f_{area} \\ & * \ Loss_{area} \end{aligned} \tag{5.11}$$

$$\left\{ \begin{array}{c} beta = 2.0 \\ f_{geometry} = 1.0 \\ f_{plotShape} = 3.0 \\ f_{surroundings} = 15.0 \\ f_{vol} = 1e-4 \\ f_{area} = 10.0 \end{array} \right\} \tag{5.12}$$

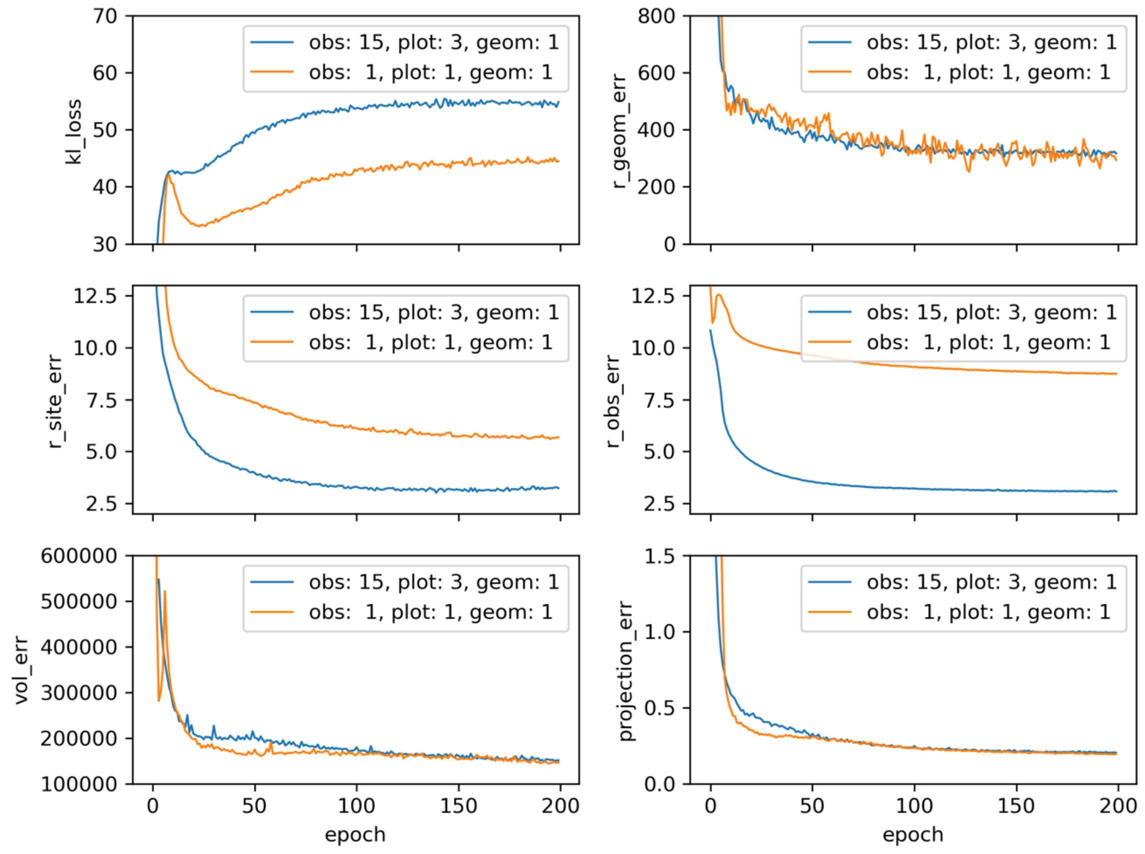


*Figure 5.8 Comparison of validation loss for two VAEs using different weights for the geometry (geom), plot shape (plot), and obstructions (obs) losses. A higher weight for plot shape and obstructions improves the respective losses without affecting the geometry reconstruction.*

*5.5.2.3. Performance Loss Terms*

To evaluate the effect of the volume (Equation ( 5.6 )) and area (Equation ( 5.7 )) loss terms to the reconstructed results' accuracy, a VAE was trained with and without these loss terms. The mean absolute percentage error (MAPE) of the reconstructed geometries from the ones in the dataset, as well as the boundary condition MAE, are summarized in Table 5.5. The addition of the area loss from Equation ( 5.7 ) slightly improved the reconstructed geometries' quality with respect to both the volume and the area coverage. The addition of the volume loss from Equation ( 5.6 ) drastically improved the volume error. The best area coverage reconstruction is achieved when both the volume and area losses are used. The best volume reconstruction is achieved without the area loss term and gets slightly worse when the area loss is added as well. Last, the volume and area loss terms do not significantly affect the boundary condition reconstruction. Therefore, it was concluded that both the volume and area loss terms contribute to better data reconstructions.

*Table 5.5*

| | Volume MAPE | Area MAPE | Plot shape MAE | Obstructions MAE |
|---|---|---|---|---|
| no performance loss | 16.61% | 19.84% | 1.37 | 3.03 |
| area loss | 16.33% | 19.07% | 1.45 | 2.91 |
| volume loss | **9.80%** | 18.63% | 1.37 | 3.03 |
| volume & area loss | 10.34% | **17.74%** | 1.44 | 2.95 |

## 5.6. Evaluation

The previous section demonstrated how the VAE was trained, minimizing the reconstruction error and avoiding overfitting. The model's ability to produce high-quality reconstructions is a prerequisite for generating new samples that follow the original data distribution, i.e. that correspond to optimally performing geometries for the respective boundary conditions. In this section the quality of the reconstructions is evaluated through visual inspection and performance comparison of the test set and the VAE-reconstructed geometries.

### 5.6.1. Performance

The performance of the reconstructed geometries is evaluated using three metrics: the solar gain objective, as defined in Equation ( 4.4 ), the total volume, and the area coverage. It is important to consider the solar gain in the context of the geometry volume and area coverage, for two reasons. First, the accurate volume and area reconstruction is a prerequisite for using them as constraints for the geometry generation during inference. Second, the solar gain performance is highly dependent on these two quantities; comparing the solar gain of two geometries with different volume, for example, would give no indication about the solar efficiency of their respective morphologies.

To calculate these metrics, the reconstructed building geometries were isolated from the reconstructed samples and combined with the original boundary conditions. Then, they were simulated using the same process that was employed during the optimization stage of the synthetic data generation. The results are summarized in Table 5.6. The mean absolute percentage error (MAPE) of the reconstructed geometries' radiation performance from the ones in the dataset is 7.41%, with a root mean square error (RMSE) of 22.138 kWh. The MAPE of the volume is 9.84% (RMSE: 421.1m$^3$) and the MAPE of the area coverage is 17.1% (RMSE: 0.088 or a deviation of 8.8% in the area coverage). The RMSE of the plot shape and obstructions reconstruction is 0.14 and 0.15, respectively.

These deviations are low enough to conclude that the reconstructed geometries remain optimal.

*Table 5.6*

| | Solar gain | Volume | Area | Plot shape | Obstructions |
|---|---|---|---|---|---|
| MAPE | 7.41% | 9.84% | 17.1% | - | - |
| RMSE | 25.437 | 414.1 | 0.088 | 0.14 | 0.15 |
| Dataset St. Dev. | 82.170 | 1802.9 | 0.146 | - | - |

### 5.6.2. Visual comparison

A number of samples (n=64) were randomly selected from the test set and reconstructed using the final VAE model. Figure 5.9 and Figure 5.10 include 3d renders in top view and 2d plots of the boundary conditions for subsets of representative examples and their reconstructions. In general, the boundary condition reconstructions are very close to the original ones. The geometry reconstructions present two trends, guiding the subdivision of samples into the two figures. Figure 5.9 demonstrates examples where the geometry reconstruction is in all aspects very close to the original geometry. Both monolithic and dispersed geometry aggregations are reconstructed accurately regarding the placement and orientation of their individual cuboid components. Figure 5.10 demonstrates examples where the reconstructed geometries' cuboids have obtained the correct placement, but with different orientations than the originals.

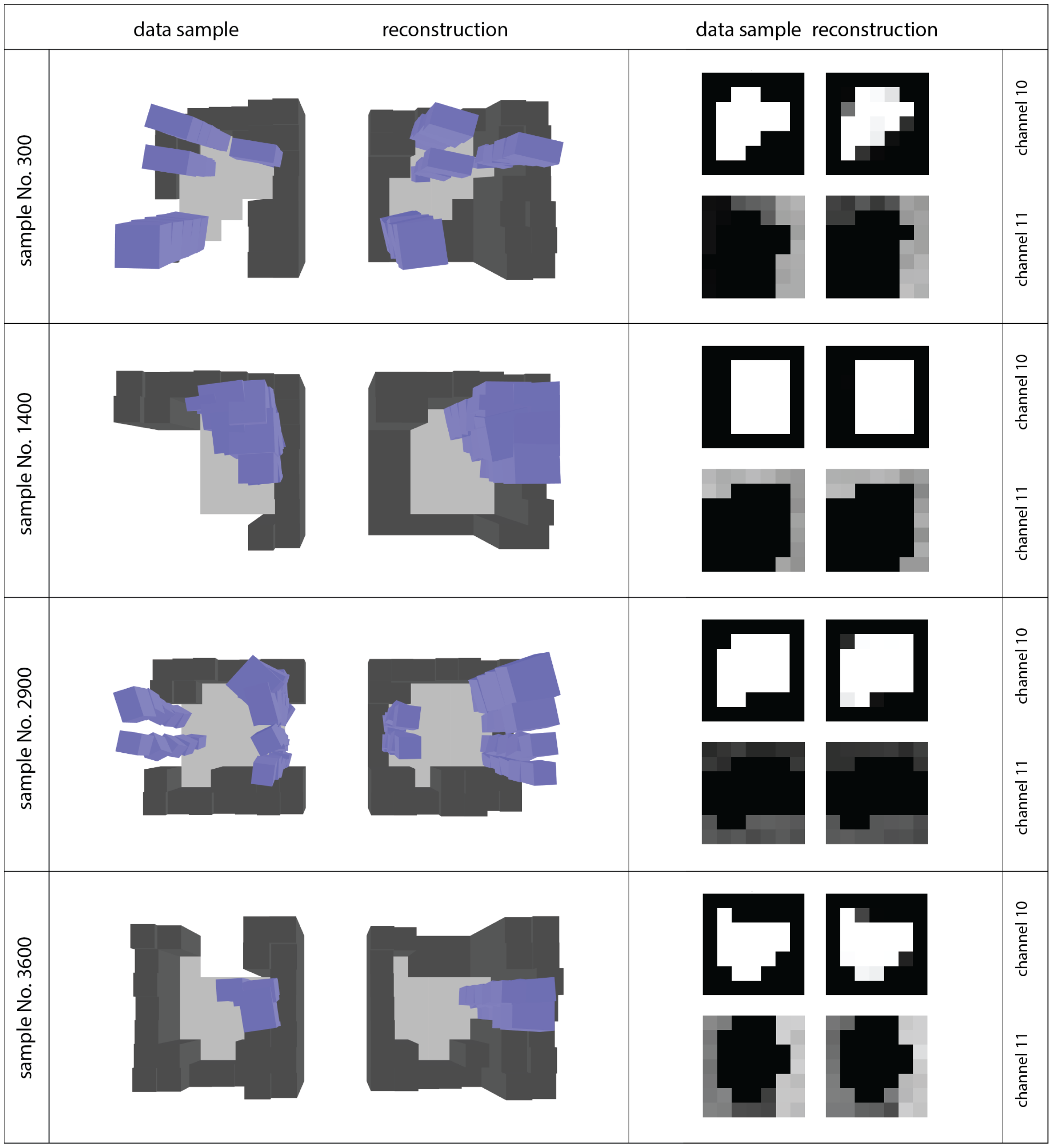


*Figure 5.9 Selected samples from the test set of dataset v2.6 and their reconstructions. Scaling and rotation cuboid parameters have been accurately reconstructed. Left: 3D renders in top view. Right: Plot shape (top of each row) and obstructions heightmap (bottom of each row).*

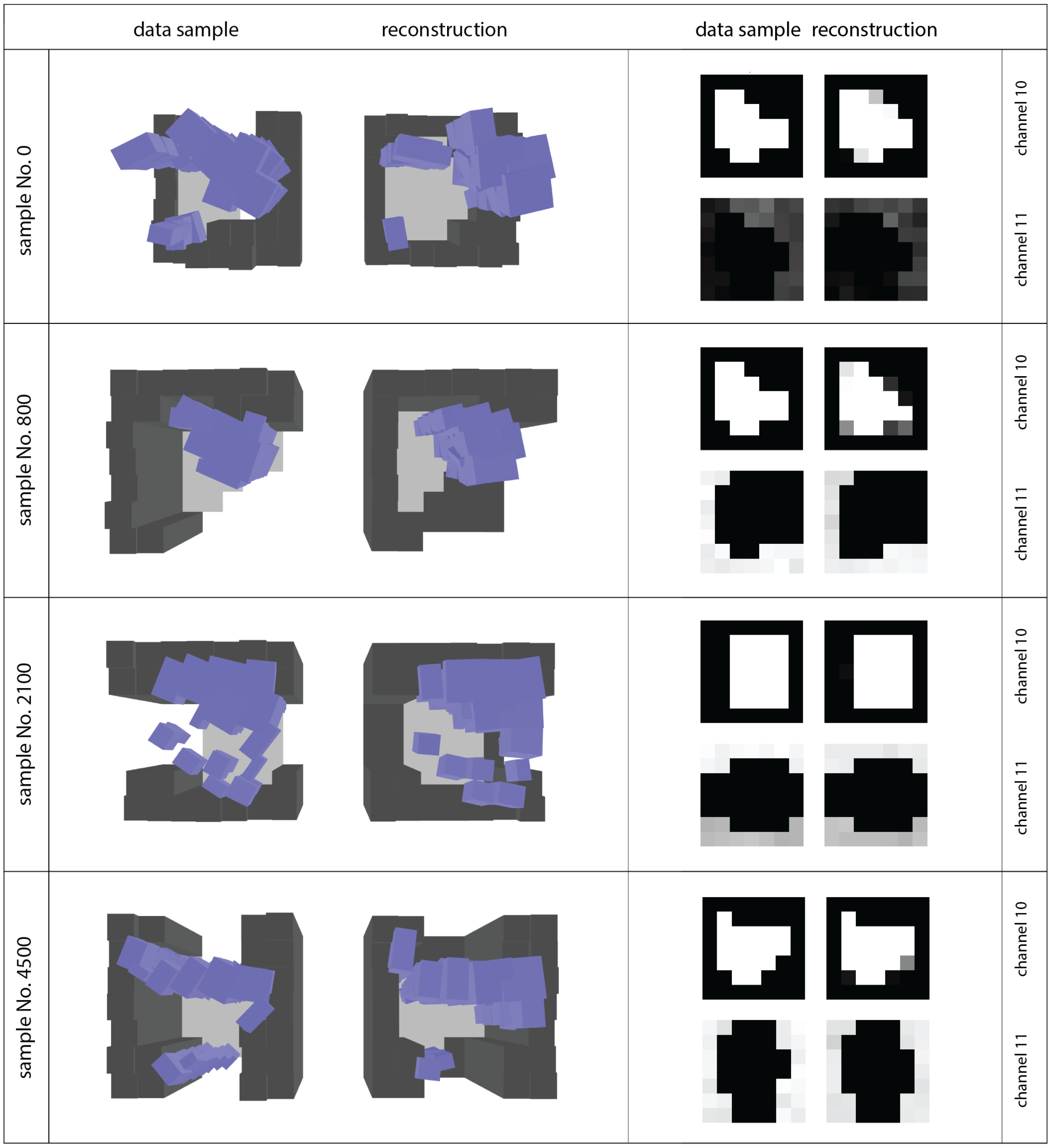


*Figure 5.10 Selected samples from the test set of dataset v2.6 and their reconstructions. Scaling parameters have been accurately reconstructed, but rotation parameters present deviations from the original ones. Left: 3D renders in top view. Right: Plot shape (top of each row) and obstructions heightmap (bottom of each row).*

The reconstructed geometries of Figure 5.9 and Figure 5.10 were simulated using the original boundary and the results plotted in Figure 5.11 An inspection of the solar gain objectives shows that the reconstructions with deviating angle parameters perform at least as well as the original data (Figure 5.11 Right).

Next, all test set samples that share the same boundary conditions as the ones in Figure 5.9 and Figure 5.10 were simulated in the same way, and their volume and solar gain performance are shown in the scatterplots of Figure 5.12. The performance of the reconstructions (orange) coincides with that of the original, optimized samples (blue). For reference, a set of baselines are included, whose geometry parameters were sampled following Gaussian distributions around evenly sampled means.

Figure 5.11 and Figure 5.12 do not reveal a significant difference regarding the accuracy of the reconstructions' solar gain performance between geometries that were reconstructed with accurate orientation and those that were not. On the other hand, a sensitivity analysis showed that the cuboids' orientation parameters have a lower effect on the solar gain objective than the scaling parameters. Therefore, a plausible conclusion is that for some optimization runs, the solutions – i.e. the geometries that made it into the dataset – deviated from the general pattern of cuboid orientations in the rest of the dataset. As a result, the VAE model did not accurately model these angle parameters; however, it managed to replicate their performance within reasonable limits.

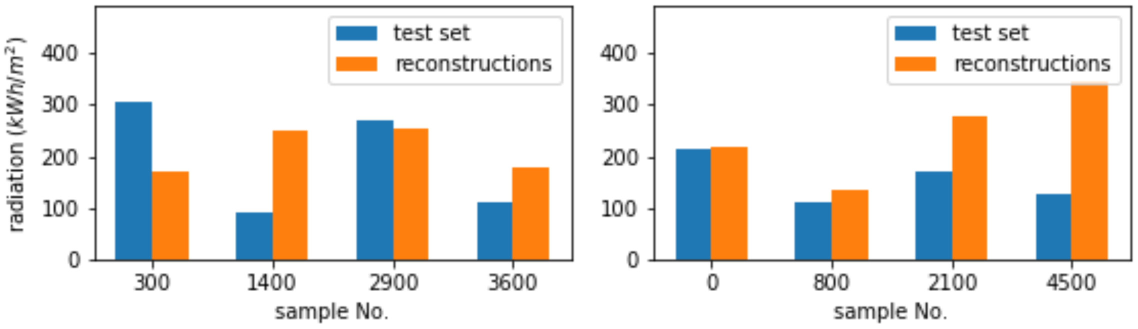


*Figure 5.11 Negative solar gain objective (higher is better) for selected samples from the test set and their reconstructions. Left: samples from Figure 5.9. Right: samples from Figure 5.10*

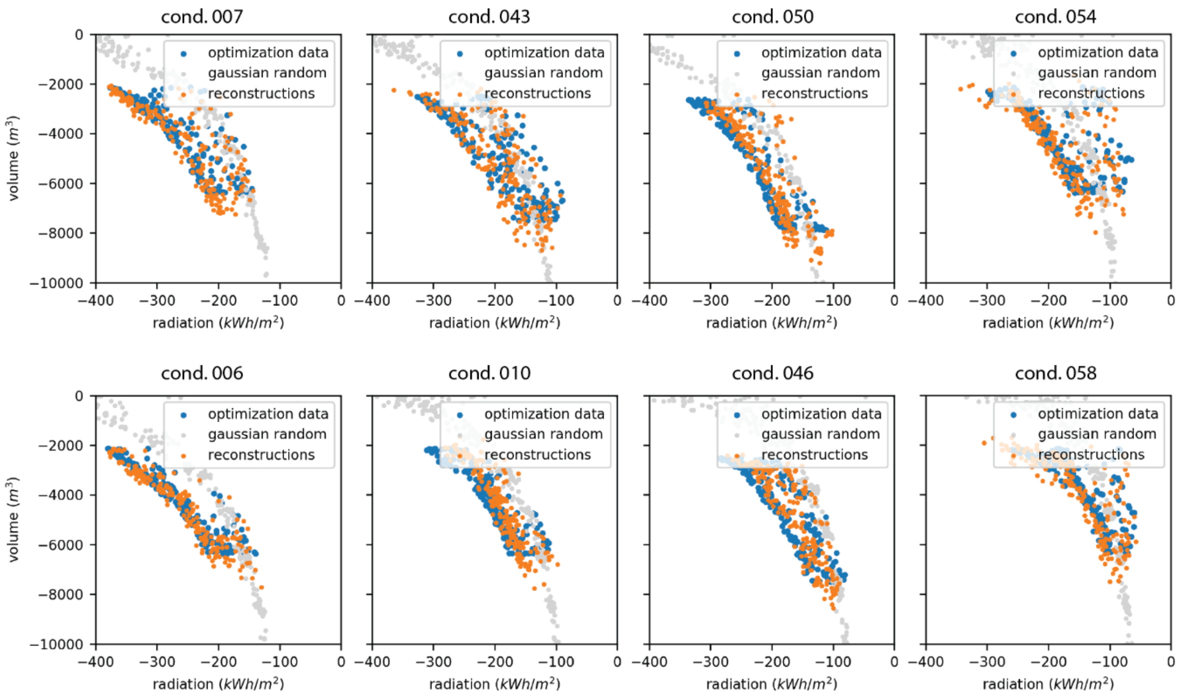


*Figure 5.12 Solar gain – volume objectives scatterplots for test set samples from 8 unique boundary conditions. Top: Boundary conditions for samples in Figure 5.11 Left. Bottom: Boundary conditions for samples in Figure 5.11 Right.*

# 6. Sampling the learned distribution

This chapter focuses on the use of the trained VAE as a generative model. The first section describes an initial evaluation of the model. The structure of the latent space is inspected by quantifying the effect of individual latent space dimensions on the generated samples. Then, the generative model is evaluated regarding the morphology and performance of the generated samples.

The second section introduces methods that enable the use of the generative model as a problem-solving tool. It first introduces a series of constrained sampling methods for generative models and specifically for VAE models, to enable the generation of geometries for specific boundary conditions. Then, constrained sampling is extended to enable the conditioning of solutions on partial geometries.

The third section discusses two types of methods for geometry exploration. Gradient-based methods and global latent-space attribute vectors are presented for goal-oriented geometry tuning.

## 6.1. Model Inspection

### 6.1.1. Latent Space Structure

The structure of the latent space is investigated by quantifying the effect of each dimension on the decoded geometry and boundary condition. This process can reveal what information is encoded into each latent space dimension. First, a zero vector $z_0 \in \mathbb{R}^{LD}$ is constructed where LD is the dimensionality of the latent space (LD = 128). Then, a set of LD vectors $z_i \in \mathbb{R}^{LD}$, i: {1, 2, 3, …, LD} is constructed; each of the vectors $z_i$ is a copy of $z_0$ where the *i*-th dimension of the *i*-th vector is set to 1. Using the trained VAE decoder, the corresponding data points {$x_j$, j: 0, 1, 2, 3, …, LD} are computed where x = D(z).

Figure 6.1 visualizes the zero latent vector decoding $x_0 = D(z_0)$. The vector $z_0$ coincides with the mean of the model's prior (the model's prior is a zero-centered Gaussian distribution introduced in section 5.1 VAE – Intro) and intuitively, it is expected to represent the most probable sample from the dataset distribution, which would be roughly the average of the training samples. Indeed, Figure 6.1 shows an obstruction heightmap with minimal heights on all sides of the building plot, and a plot shape occupying the maximum area. Last, the geometry encoded in the zero vector occupies the higher levels of the North-East corner. The preference for the higher levels of the North size can be explained given the probability of obstructions shading from the North, East, and West. The shift towards the East may be caused by the location-specific weather patterns, a bias in the boundary conditions sampling, or convergence to local minima during the dataset optimization.

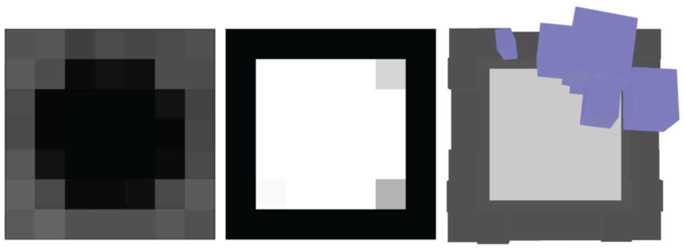


*Figure 6.1 Activation of zero latent vector $z_0$. Left: obstructions heightmap, where black is zero and white is the maximum height. Center: plot shape, where white indicates buildable plot. Right: complete rendered sample with obstructions in gray, plot shape in light gray, and geometries in blue.*

The difference of each data point $x_i$ from the zero-activation data point $x_0$ is calculated. Specifically, the L1 distances of the corresponding geometries, obstructions, and plot shapes are calculated using Equation ( 6.1 ).

$$Dist(a,b) = \sum_{i=1}^{nCuboids} \sum_{j=1}^{nChannels} |a_{ij} - b_{ij}| \tag{6.1}$$

The impact of each dimension *i* to the geometry, obstructions, or plot shape, is proportional to the corresponding distance Dist( $x_0^{geom}$ , $x_i^{geom}$ ) , Dist($x_0^{obs}$, $x_i^{obs}$) or Dist( $x_0^{plot}$ , $x_i^{plot}$) respectively. Figure 6.2 shows the effect of a unit step along each dimension of the latent space. The geometry, obstructions, and plot shape differences are plotted against the mean standard deviation of the training data for each dimension of the latent space, as it is output from the VAE encoder. As expected, on average, dimensions with higher uncertainty have a smaller effect on the generated sample.

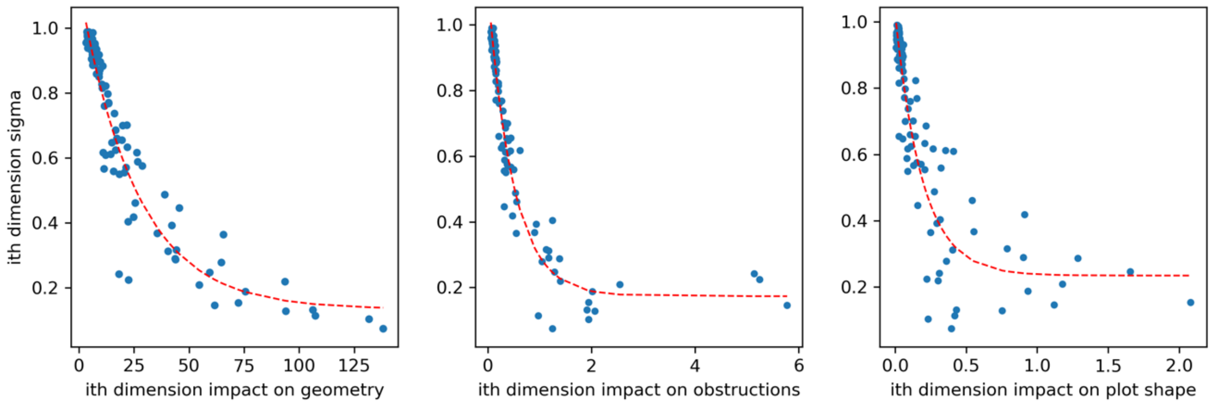


*Figure 6.2 Each point on the scatterplot represents one latent space dimension ($n_{LD}$=128). The Y-axis shows the mean standard deviation of the distribution predicted by the VAE encoder for the training data, along each latent space dimension. The X-axis shows the effect of taking one unit step along each latent space dimension. Left: effect on decoded geometries. Center: effect on decoded obstructions. Right: effect on decoded plot shapes. The red curve represents an exponential fit using non-linear least squares.*

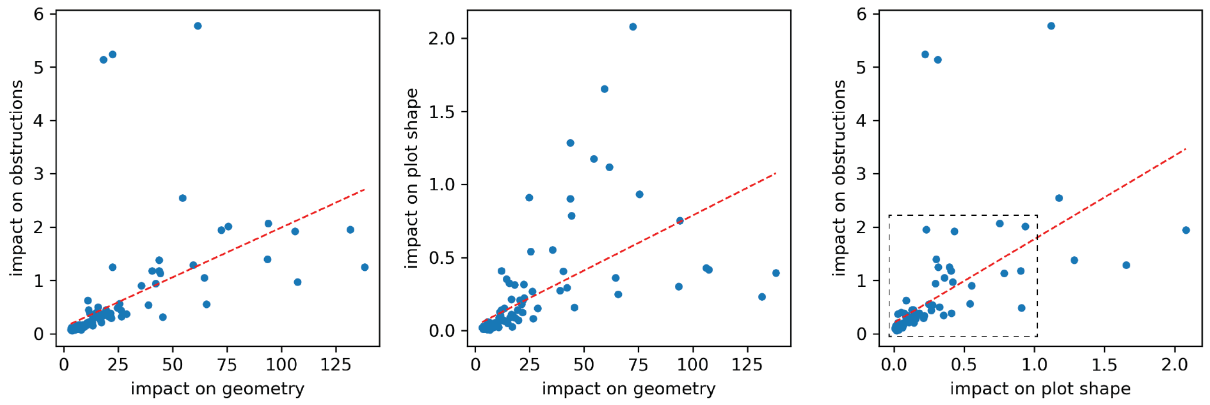


*Figure 6.3 Each point represents the effect of a unit step along one latent space dimension based on a zero vector. The red curves represent linear functions fit to the data using least squares.*

Figure 6.3 depicts the relationship between changes in the geometry, obstructions, and plot shape for a unit step taken along each dimension of the latent space. Figure 6.3 Left shows that, in general, latent dimensions associated with a strong effect on the generated obstructions have a strong effect on the generated geometries as well. This is an indication that the latent space has successfully encoded the relationship between boundary conditions and optimal geometric solutions. Figure 6.4 seventh column, for example, shows that a reduced southeast obstructions height is associated with the presence of a geometry in the same corner of the plot. When the southeast obstructions height increases, that geometry disappears because it would be shaded in a way causing suboptimal solar gain performance. In Figure 6.4 fifth column, a southeast increase of the obstruction heights causes the geometry to shift and rotate westward, increasing its solar access. Similarly, in Figure 6.4 eighth column, a decrease of the west obstructions' height allows the geometry to extend towards the west.

This relationship of proportional effect seems to break for the three dimensions with the strongest effect on obstructions (Figure 6.3 Left, upper left area of the scatterplot). A closer inspection reveals that the corresponding vectors activate obstructions that are west, north, and east of the geometry, as shown in the three leftmost columns of Figure 6.4. This can be explained since shading obstructions on these orientations have a weaker effect on the solar radiation objective, and therefore they are expected to have a smaller effect on the optimal geometric forms.

Figure 6.3 Center shows that some latent space dimensions encode information on geometry changes with little effect on the plot shape (horizontal spread) indicating the existence of various geometric solutions within a small range of plot shapes. This is in line with the existence of multiple obstructions and geometric solutions combined with each plot shape in the training set. Figure 6.5 illustrates how large changes in the geometry produced by unit steps along single latent space dimensions are only associated with small changes in the plot shape. In addition, Figure 6.3 Center implies that similar geometries can be found in different plot shapes as well (vertical spread).

The marked area in Figure 6.3 Right shows a relatively even spread of obstructions and plot shape changes for dimensions that have a small or medium effect on the boundary conditions. This hints toward the ability to easily formulate various combinations of obstructions and plot shapes. Dimensions that cause major changes to either obstructions or plot shapes can also be identified, indicated by points on the upper left and lower right areas of the scatterplot respectively.

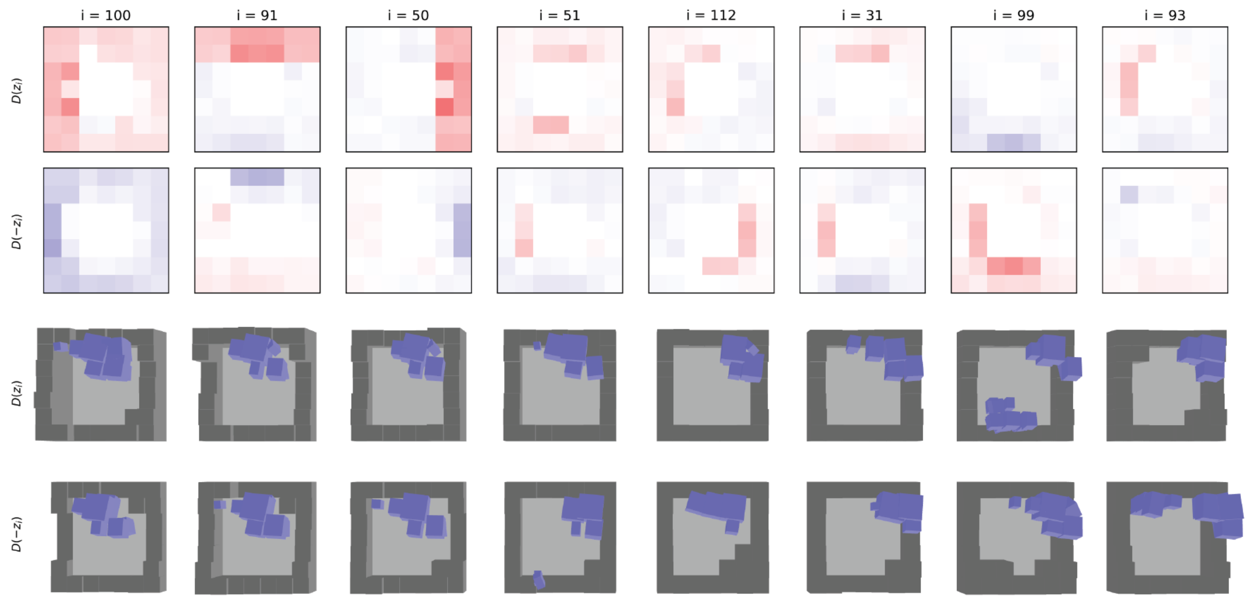


*Figure 6.4 First row: obstruction heightmap differences between activations of $z_0$ and the first 8 $z_i$ with the strongest effect on obstructions, sorted from left to right. White: $obs_0 == obs_i$, red: $obs_0 < obs_i$, blue: $obs_0 > obs_i$. Second row: the corresponding differences using activations of the negative $z_i$. Third and fourth rows: the corresponding rendered complete samples.*

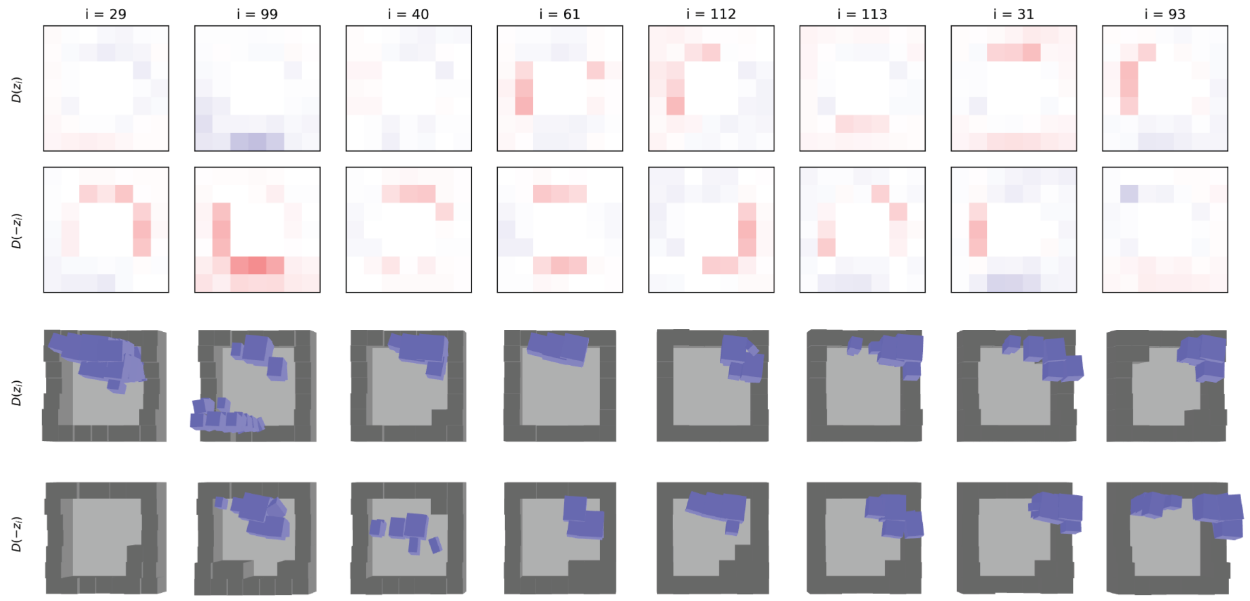


*Figure 6.5 Same as Figure 00 but sorted by the strongest effect on geometry.*

### 6.1.2. Generative Model Samples

A number of random samples (n=32) from the latent space were drawn following a normal distribution. The samples were mapped to the data space using the VAE decoder, resulting in 32 boundary conditions and geometries. Figure 6.6 illustrates the rendered samples, where morphological similarities to the optimized data can be easily identified.

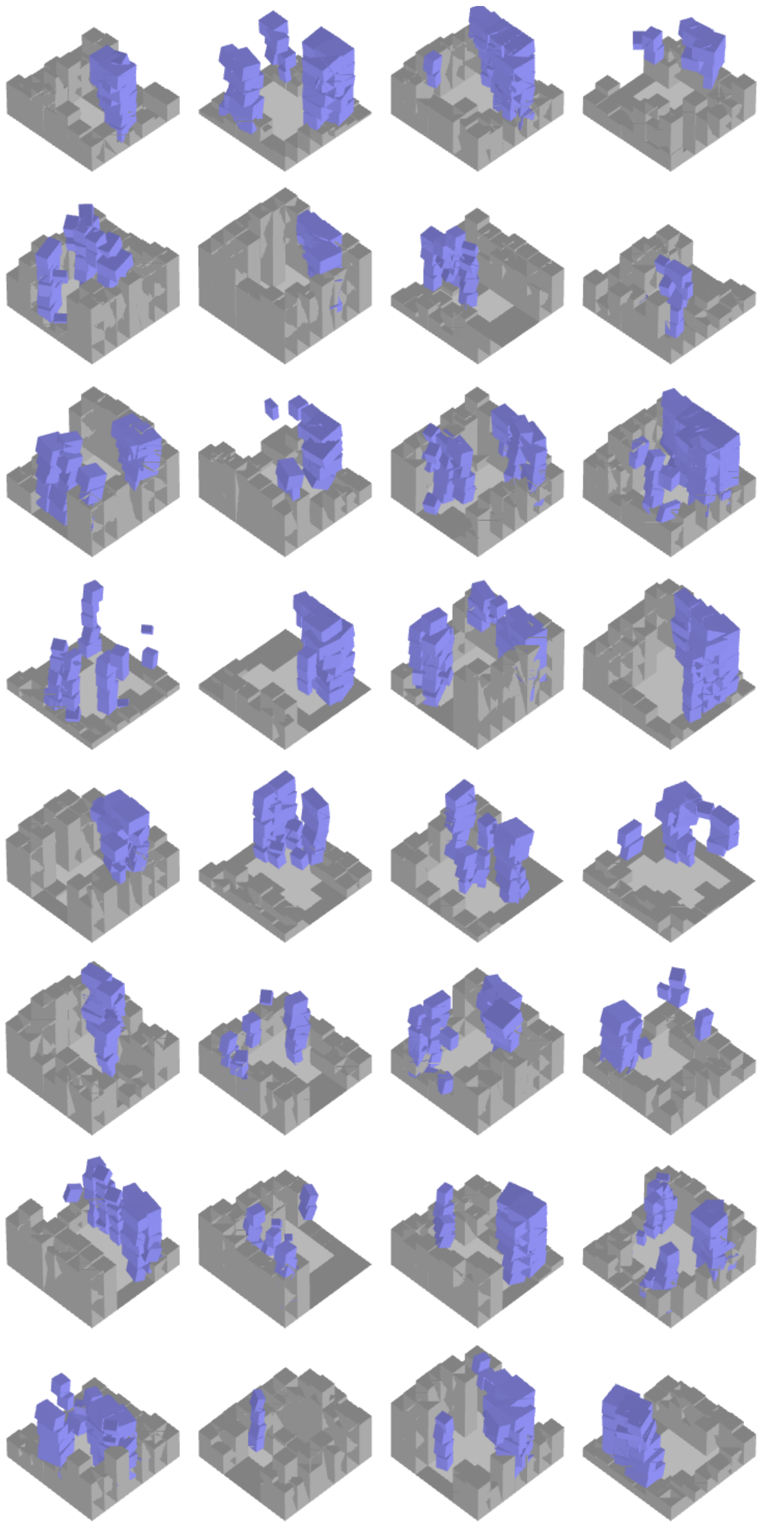


*Figure 6.6 Southeast isometrics of decoded latent space samples. Obstructions in gray, plot shape in light gray, and geometries in blue.*

To evaluate the generated samples' performance, the optimization objectives of solar gain, volume, and radiation were calculated. Next, their boundary conditions were isolated and formed 32 new optimization problems. An optimization process identical to the one used during the dataset generation provided a set of optimal geometries for each boundary condition. A comparison of the VAE-generated geometries with the corresponding optimized geometries showed that 63% (20 out of 32) of the VAE-generated geometries surpassed the Pareto fronts acquired at 200 generations and an additional 22% (7 out of 32) surpassed Pareto fronts acquired between 165 and 200 generations. The remaining 16% (5 out of 32) geometries were further evaluated by comparing their solar gain performance to that of optimized geometries with similar volume and area coverage. The results are detailed in Table 6.1, using a similarity threshold of 150m$^3$ for the volume and 15% for the area coverage difference. In sum, 6% (2 out of 32) of the samples did not have similarly sized geometries in the optimized data set, 6% of the samples had worse solar gain performance than the mean performance of the matched samples, and 3% (1 out of 32) had better performance.

In conclusion, the geometries generated by the VAE followed the dataset morphology and performed similarly to optimization results for their respective boundary conditions. A small percentage of samples did not have optimal performance.

*Table 6.1*

|  | Number of similar optimized samples | Optimized samples mean solar gain | Sample solar gain |
|---|---|---|---|
| sample 13 | 0 |  | -290.55 |
| sample 14 | 0 |  | -225.46 |
| sample 17 | 4 | **-427.06** | -326.89 |
| sample 24 | 19 | **-405.10** | -360.51 |
| sample 28 | 16 | -346.64 | **-356.56** |

## 6.2. Sampling Constrained by Site

The previous section showed that the VAE decoder can generate optimal data samples. However, to enable the use of the VAE as a design tool, more control over the generated samples is necessary. Instead of sampling a random boundary condition–building geometry pair, a building geometry needs to be generated for a given boundary condition. This section introduces a series of methods to achieve constrained sampling of the VAE's latent space. The pseudo-Gibbs and iterative Autoencoder (AE) dynamics methods are adopted from the deep learning data imputation literature. The latent space optimization and the interpolation-based methods don't have a similar theoretical background but are based on commonly used techniques for deep generative models.

### 6.2.1. Pseudo-Gibbs

The first method treats the constrained sampling problem as an imputation of data Missing Not At Random (MNAR). It involves a random initialization of the missing data and their iterative replacement with their VAE reconstructions. It was first introduced by Rezende et al. who formulated it as a Markov chain and showed that it generates samples from the correct distribution within an error $\varepsilon$ given a sufficiently accurate model (Rezende et al., 2014). McCoy et al. compare it with an analogous method of multiple imputations using principal component analysis (PCA) and find it superior (McCoy et al., 2018). Mattei and Frellsen give it the name pseudo-Gibbs sampling due to its similarity to Gibbs sampling (Geman & Geman, 1984) and suggest an extension, Metropolis-within-Gibbs sampling, which has improved convergence properties (Mattei & Frellsen, 2018).

In this work, the boundary condition is treated as the observed data $x_{obs}$ and the geometry as the missing data $x_{miss}$, with x = ($x_{obs}$, $x_{miss}$). The geometry data is initialized following a normal distribution, $x^0_{miss}$ ~ N(0, 1). Then the data is reconstructed through the VAE to obtain a new geometry $x^{t+1}_{miss}$, as shown in Equation ( 6.2 ), where $r_{VAE}$ reconstructs a data point through the VAE. The resulting geometry is combined with the original boundary condition (Equation ( 6.3 )) and a new reconstruction can be obtained. The process is repeated until convergence. In practice a maximum limit on the number of iterations can also be used.

$$x_{miss}^{t+1} = r_{VAE}(x^t)_{miss} \tag{6.2}$$

$$x^{t+1} = (x_{obs}, x_{miss}^{t+1}) \tag{6.3}$$

Figure 6.7 includes several graphs tracking the convergence of geometry imputations using pseudo-Gibbs sampling for the whole test set. Geometry convergence is traced by calculating the difference between the geometries generated in two consecutive iterations. Boundary condition errors are calculated with respect to the ground truth. Hypervolumes are calculated with respect to the solar gain, volume, and area objectives. Figure 6.7 Top Right shows that the boundary condition error was minimized within approximately 20 iterations for the test set. Figure 6.7 Top Left shows that only minimal changes in the predicted geometries occurred beyond the 50th iteration, and Figure 6.7 Bottom Right shows that the hypervolume of the solar gain, volume, and area objectives converged within 50 iterations as well. Figure 6.7 Bottom Left shows an example where the VAE-generated geometry hypervolume surpassed that of the optimized geometry after approximately 20 iterations.

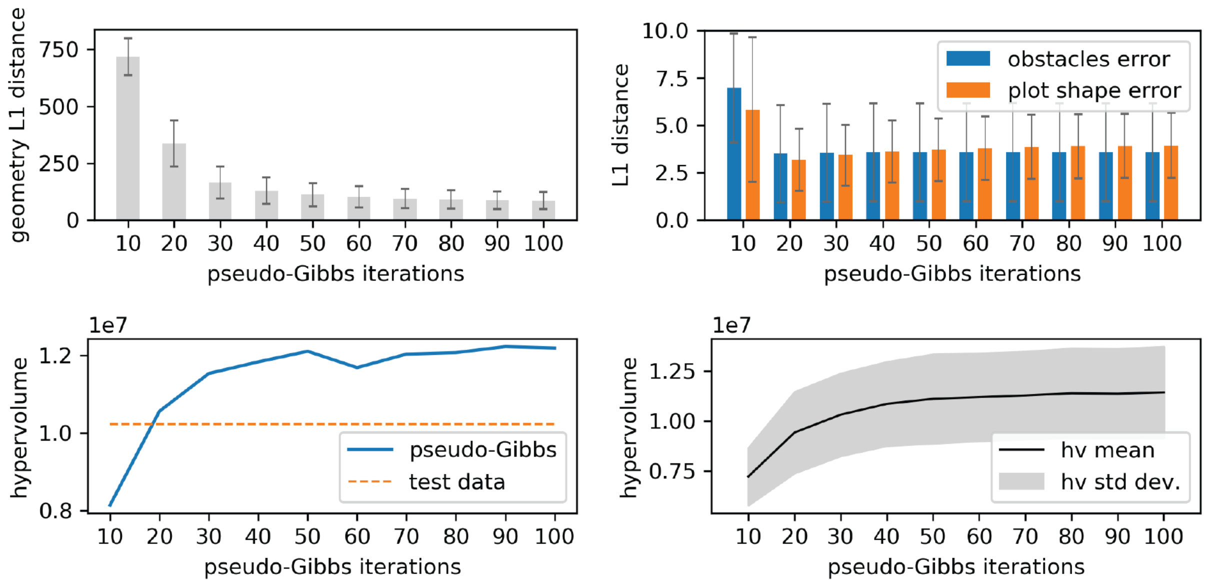


*Figure 6.7 Convergence of pseudo-Gibbs sampling using the test set. Top Left: Geometry change (test set mean and std dev.). Top Right: Boundary condition errors (test set mean and std dev.). Bottom Left: Representative example of optimized and VAE-generated geometries hypervolume. Bottom Right: Test set hypervolume.*

Figure 6.8 shows an example of how the performance of the generated solutions for a single boundary condition changed during 100 pseudo-Gibbs iterations. The performance of pseudo-Gibbs sampled geometries is plotted against the performance of the optimized data for the same boundary condition. The geometry initialization data are included as a baseline for reference, with geometry values sampled from Gaussian distributions with means between 0.18 and 0.5. Initially, the performance of the generated geometries has little overlap with that of the optimized ones. After 50 iterations, the two performance distributions appear very close to each other. Finally, further improvement is observed after 100 iterations, especially for the solar gain – area coverage scatterplot. In the final result, the distribution of the solar gain, volumes, and areas of the generated geometries appears very similar to that of the optimized data.

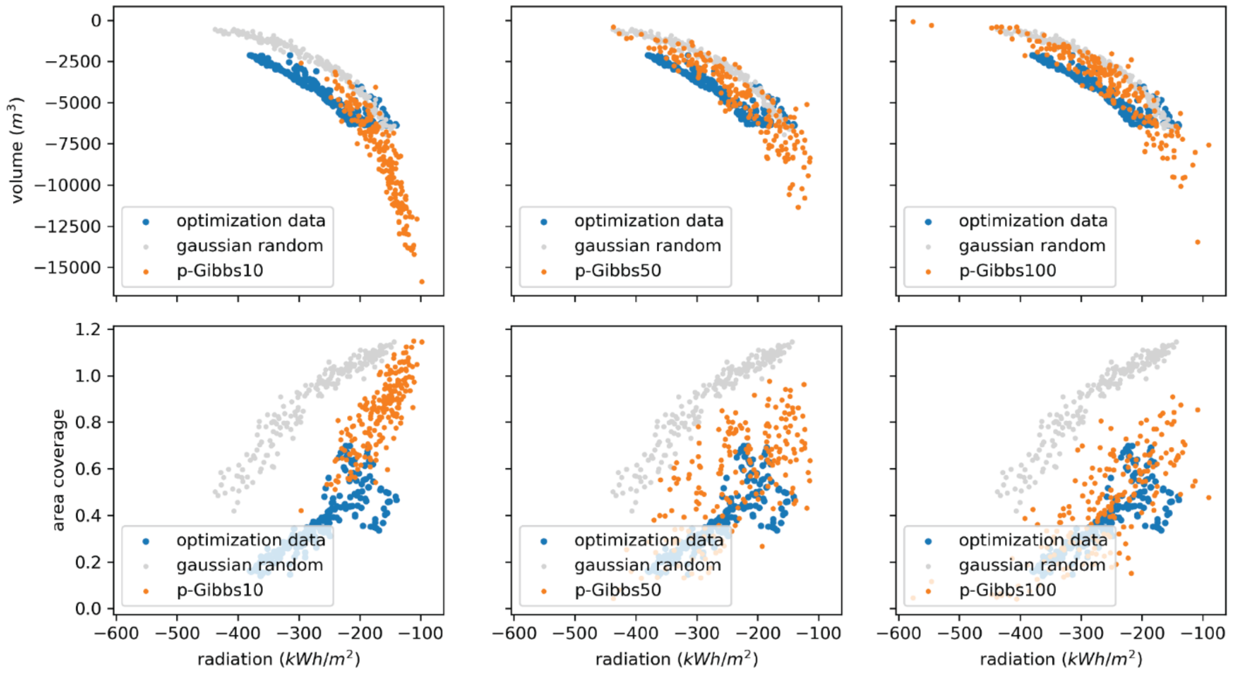


*Figure 6.8 Solar gain-volume (Top) and solar gain-area (Bottom) objectives scatterplots for a single boundary condition.*

Last, it was observed that the size of the generated geometries can be roughly controlled by changing the initialization of the geometry data. Figure 6.9 shows an example of how initializing the geometries by sampling normal distributions with different means affects the geometry generated after 100 pseudo-Gibbs iterations. All generated geometries resemble the morphologies of the optimized data, while the total size seems to follow that of the initialization.

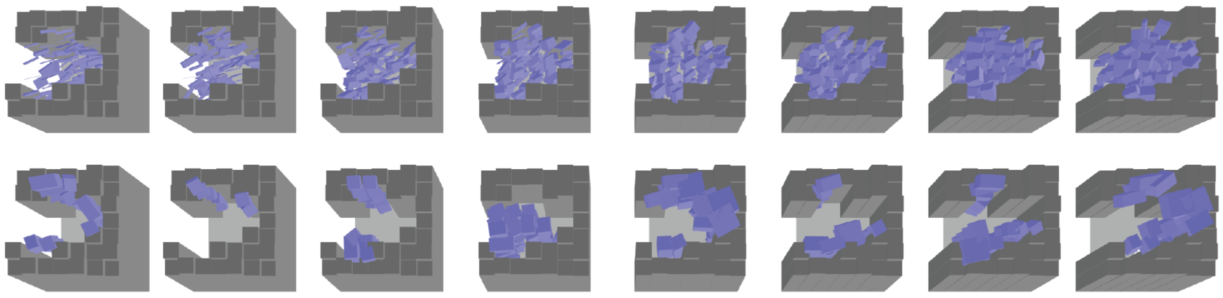


*Figure 6.9 Example of geometry initialization (Top), and corresponding geometries generated using pseudo-Gibbs sampling (Bottom). From left to right, the mean of the randomly initialized geometries increases.*

### 6.2.2. Autoencoder dynamics

Śmieja et al. proposed a modification to pseudo-Gibbs sampling that is equivalent to a projected gradient ascent, reporting improved results compared to imputation using pseudo-Gibbs sampling (Śmieja et al., 2020). In practice, each new reconstruction only affects the initialized datapoint to a small degree *h*, as shown in Equation ( 6.4 ).

$$x_{miss}^{t+1} = x_{miss}^{t} + h[r_{VAE}(x^t) - x^t]_{miss} \qquad (6.4)$$

Similar to pseudo-Gibbs sampling, for the purpose of optimal geometry generation, the boundary conditions are treated as the observed data and the geometries as the missing data. Figure 6.10 shows that, using the test set boundary conditions, the geometry generation converged at around 600 iterations, as indicated by the geometry difference (Top Left) and the hypervolume graphs (Bottom Right). The boundary condition error (Top Right) is similar to the one achieved using pseudo-Gibbs sampling and remains steady beyond 200 iterations. Figure 6.10 Bottom Left shows an example where the generated geometry achieved a higher hypervolume than the optimized data from the first iteration and continued improving its performance until the 1700th iteration. Figure 6.11 plots the performance of the generated geometries against that of the optimized ones, for a single boundary condition, showing how it changes as the iterations progress. The distribution of the generated geometries' solar gain, volume, and area objectives becomes very close to that of the optimized data for the same boundary condition, within 1000 iterations.

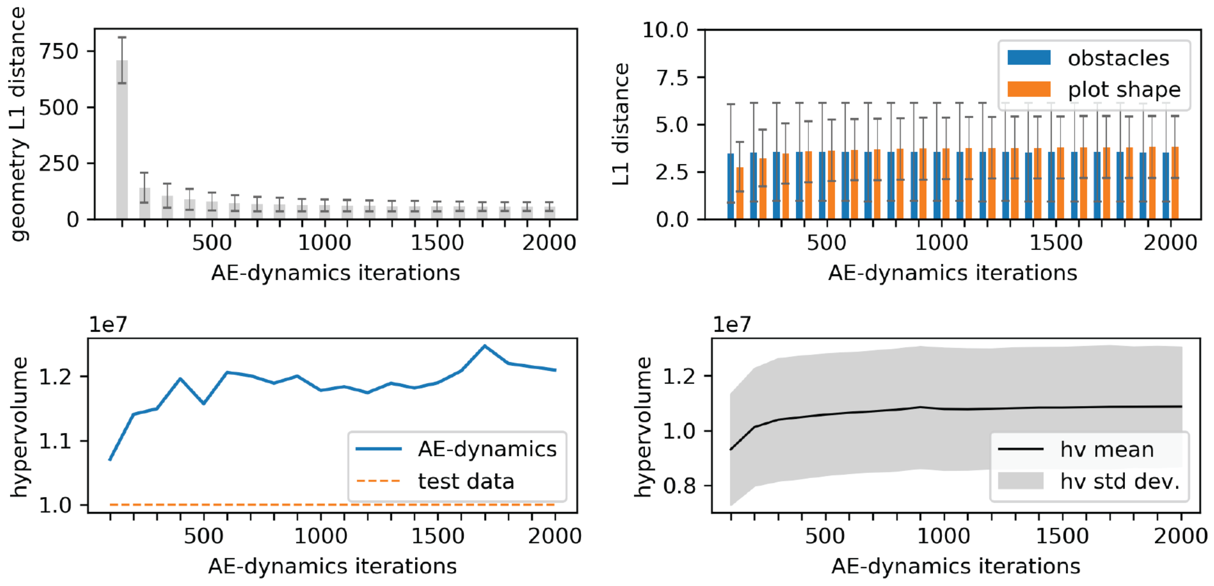


*Figure 6.10 Convergence of iterative Autoencoder dynamics sampling, using the test set. See Figure 6.7 for graph details.*

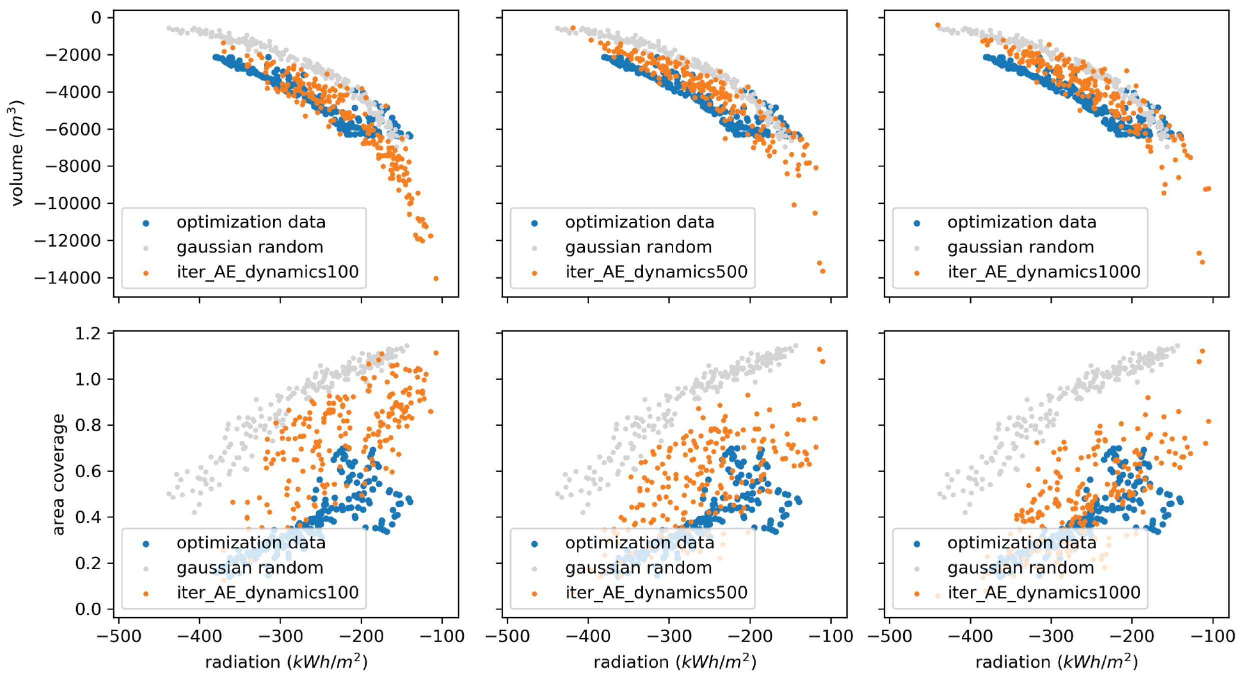


*Figure 6.11 Performance of iterative Autoencoder dynamics sampled geometries plotted against the performance of optimized data for a single boundary condition (cond006). Baseline same as described in Figure 6.8.*

An example of geometries generated using this method is shown in Figure 6.12 Bottom. Increasing noise intensities for the initialization step appears to roughly correspond to increasing volumes of the generated geometries. The resulting morphologies obtained for different geometry initializations appear more consistent than when using pseudo-Gibbs sampling, which can be potentially attributed to the smoothing factor *h*.

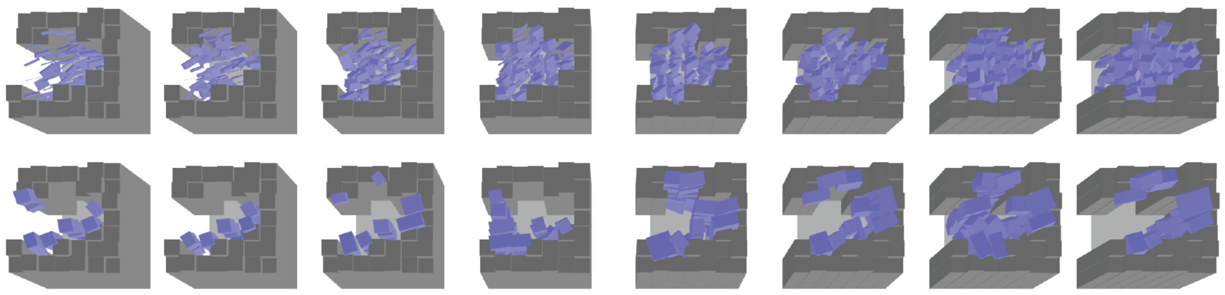


*Figure 6.12 Example of geometry initialization (Top), and corresponding geometries generated using iterative Autoencoder dynamics (Bottom). From left to right, the mean of the randomly initialized geometries increases.*

### 6.2.3. Latent space optimization

Another way to sample the generative model with constraints is to relax the constraints, formulate them as optimization objectives and perform a gradient-based search in the latent space. This method does not require an encoder network and bears similarities to the problem of GAN inversion, where a latent representation z* is retrieved given a target data sample x (Creswell & Bharath, 2019). The problem is solved by gradient descent, optimizing a random sample z as shown in Equation ( 6.6 ), where G is the generative network and L is a distance metric in the data space.

$$z^* = argmin\ L(G(z), x) \qquad (6.5)$$

However, in this work, different than in the inversion problem, only part of the target sample is known. In a similar case, Gadelha et al. used a custom "coverage metric" in the optimization objective to solve the problem of geometry completion using a VAE (Gadelha et al., 2020). Here, only the target boundary condition is known, and therefore the optimization objective takes the form of Equation ( 6.6 ) where the function $L_{bc}$ returns the distance of the boundary conditions of two samples, and D is the decoder network of the VAE. Finally, the boundary condition constrained sample is obtained using Equation ( 6.7 ), and the geometric solution is extracted $x^*$ = ($x_{bc}^*$, $x_{geom}^*$).

$$J(z) = L_{bc}(D(z), referenceSample) \tag{6.6}$$

$$z^* = argmin\, J(z), \quad x^* = D(z^*) \tag{6.7}$$

The solution can be further controlled in terms of any differentiable property or performance with the addition of a relevant term. Specifically, a geometry with the desired volume can be obtained as shown in Equations ( 6.8 ) and ( 6.9 ), where γ controls the compliance with the volume objective.

$$J(z) = L_{bc}(D(z), referenceSample) + \gamma L_{volume}(D(z), referenceSample) \tag{6.8}$$

$$L_{volume}(x, y) = |Volume(x) - Volume(y)| \tag{6.9}$$

Figure 6.13 shows the convergence during latent space optimization for 5 different initializations of 8 samples using a boundary condition from the test set, with and without a volume objective term. The addition of the volume objective adds some noise to the boundary condition error minimization (Figure 6.13 Center); however, in both cases the optimization seems to converge to a similar point within 100 iterations.

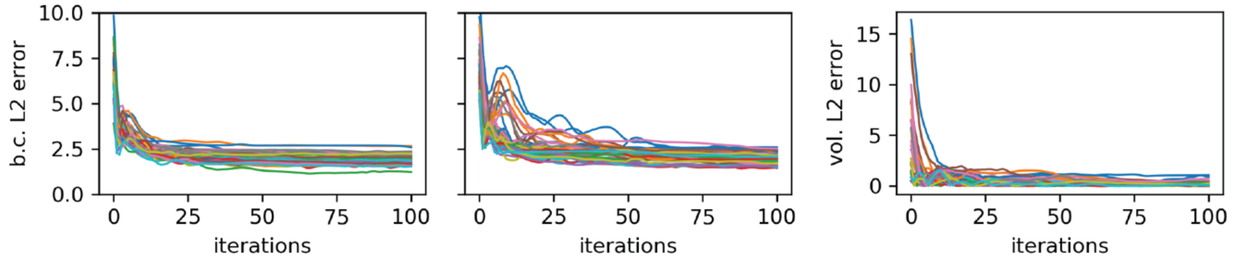


*Figure 6.13 Optimization objective convergence diagrams. Left: Latent space optimization using boundary condition error. Center and Right: Latent space optimization using boundary condition and volume errors.*

Figure 6.14 shows an example of 8 different reference samples $X_0$ (Top) initialized with geometries following a normal distribution with increasing mean, results obtained without the volume objective (Center), and results obtained with the volume objective (Bottom). The vector z was initialized by sampling the distribution $E(X_0)$. In both cases, the effect of the initialization data mean is reflected on the size of the resulting geometries. However, the range of the sizes is very different between the results that use the volume objective and those that do not.

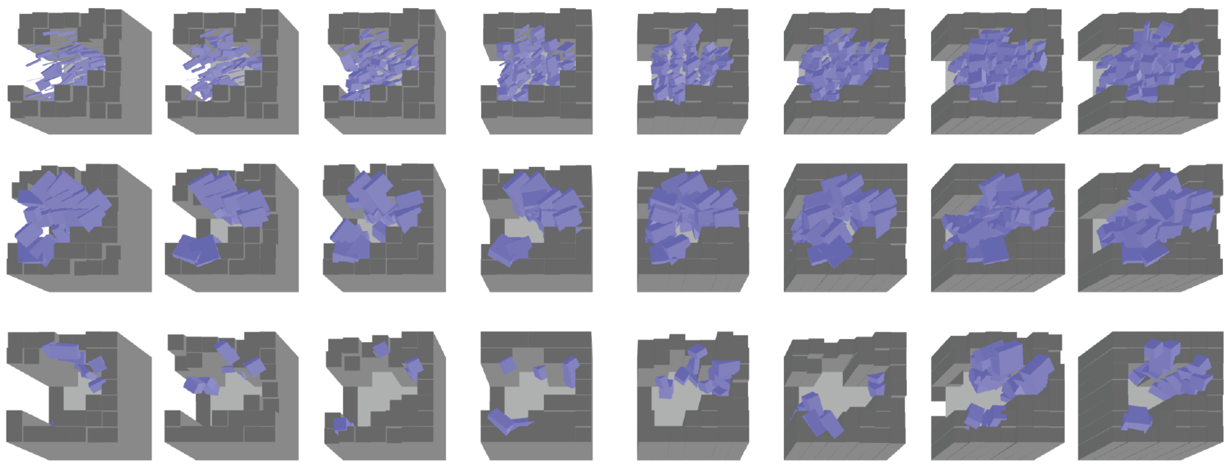


*Figure 6.14 Example of geometry initialization (Top), corresponding geometries generated using latent space optimization with boundary condition error (Center), and corresponding geometries using latent space optimization with boundary condition and volume error (Bottom).*

Figure 6.15 quantifies their respective solar gain, volume, and area coverage objectives evaluation. In Figure 6.15 Left, the solar gain – volume performance of the geometries obtained without the use of the volume objective seem to be on the extension of the Pareto front of the optimized data for the same condition; however, their volume range is limited to the higher range and no geometries of smaller size were obtained. In contrast, the addition of the volume objective offers better control of the generated geometries' size, resulting in geometries whose performance objectives resemble those of the optimized data.

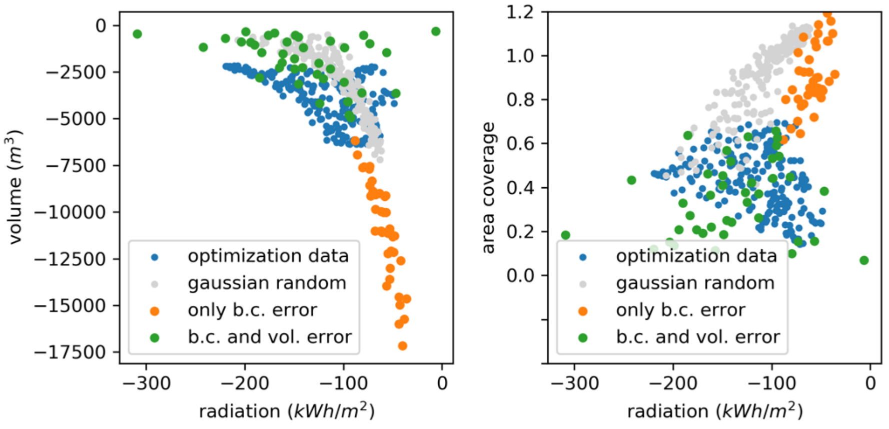


*Figure 6.15 Performance of geometries generated with latent space optimization, with and without volume error, for a single boundary condition (cond006). Baseline same as described in Figure 6.8.*

### 6.2.4. Max – min Interpolation

The last method for generating boundary-condition constrained samples with a controllable size leverages interpolation in the latent space. First, the latent representations of two extremal samples are required, one with the minimal and one with the maximal size. Equation ( 6.10 ) shows the construction of the minimum and maximum geometry samples, where $x_{bc}$ is the target boundary condition, and $x_{geomMin}$ and $x_{geomMax}$ are geometries where all the scaling parameters have been set to 0 or 1, respectively. Then, linear or spherical interpolation in the latent space produces vectors that correspond to geometric solutions with monotonically varying sizes. Finally, the geometries are obtained by passing the interpolations through the decoder network. Equation ( 6.11 ) shows how a sample $x_v$ can be obtained for an interpolation value v.

$$x_{min} = (x_{bc}, x_{geomMin}), \quad x_{max} = (x_{bc}, x_{geomMax}) \tag{6.10}$$

$$x_v = D\big(interpolate(v, E(x_{\min}), E(x_{max}))\big), \quad v \in [0,1] \tag{6.11}$$

Figure 6.16 shows examples of samples generated for a single boundary condition for multiple interpolation values. Previous research has suggested that spherical interpolation in the latent space creates samples that better match the data distribution (White, 2016). However, Figure 6.16 and Figure 6.17 do not provide a strong indication of superiority for either interpolation method. The solar gain, volume, and area objectives evaluations are very similar for the two, and their respective scatterplots suggest that they belong to the optimized data distribution.

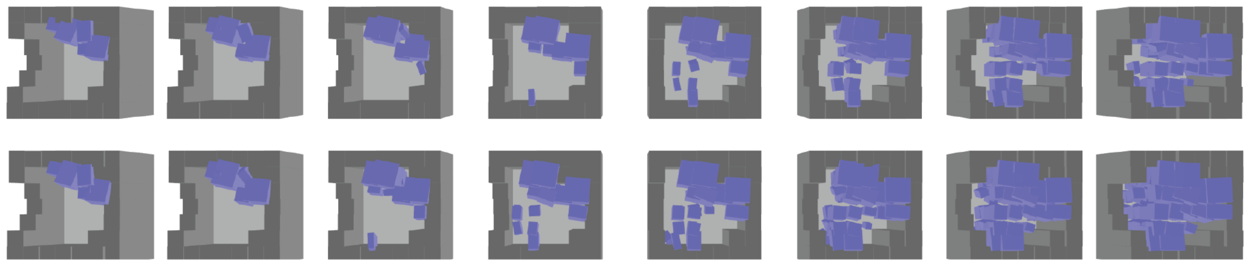


*Figure 6.16 Example of latent space interpolation between the encoding of empty and full geometry. Top: linear interpolation. Bottom: spherical interpolation (slerp).*

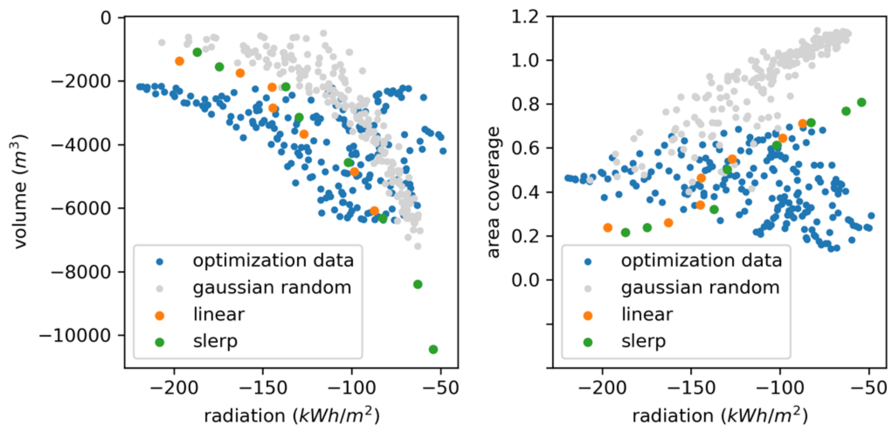


*Figure 6.17 Performance of geometries using linear and spherical interpolation of minimum and maximum geometry latent space encodings. Baseline same as described in Figure 6.8.*

### 6.2.5. Comparison

Next, the five methods for boundary condition constrained sampling (pseudo-Gibbs, AE-dynamics, latent space optimization without and with volume objective, max-min interpolation) are compared.

#### *6.2.5.1. Constraint Compliance*

Figure 6.18 shows the boundary condition error of the generated samples using the five methods, i.e., the compliance of the generated samples to the boundary condition constraint, for the whole test set. The mean boundary condition errors for pseudo-Gibbs, iterative AE-dynamics, and latent space optimization are within an acceptable range, with the best results achieved for the latent space optimization methods. The boundary condition error for the max-min interpolation is higher, and the generated obstructions and plot shapes can be visibly different from the target. For example, in Figure 6.16 the east is consistently occupied by a medium-height obstruction, contrary to the target as seen in Figure 6.14 Top row, which has an unobstructed east.

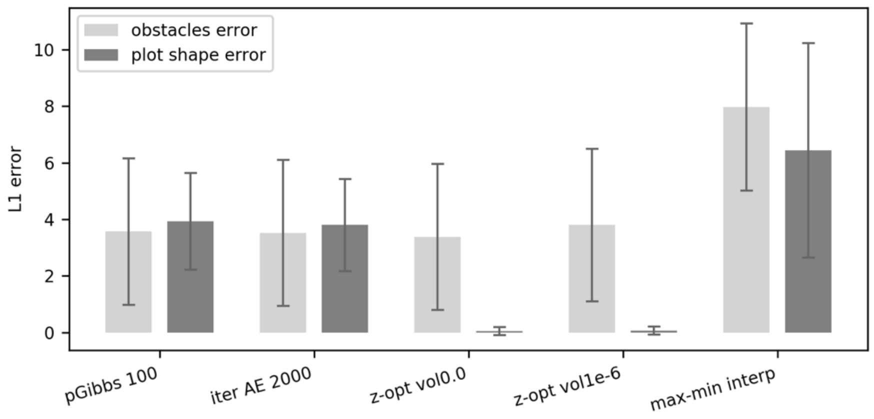


*Figure 6.18 Boundary condition errors of retrieved solutions for boundary condition-constrained sampling using the test set.*

The boundary condition error for the max-min interpolations directly depends on the latent vectors $z_{min}$ and $z_{max}$ retrieved for the geometries $x_{geomMin}$ and $x_{geomMax}$. The process for obtaining these latent vectors described in Equations ( 6.10 ) and ( 6.11 ) is the equivalent of a single pseudo-Gibbs iteration without the final decoding and replacement of the observed data. Therefore, Equation ( 6.11 ) can be reformulated as:

$$x_v = D\left(interpolate\left(v, {z^t}_{\min}\ , {z^t}_{max}\right)\right), \quad v \in [0,1] \tag{6.12}$$

where $z^t_{min}$, $z^t_{max}$ are the latent vectors obtained after *t* pseudo-Gibbs iterations. A small number of iterations can significantly improve the boundary condition error while still encoding the minimal and maximal geometries. Figure 6.19 Bottom shows how the boundary condition error is improved for four pseudo-Gibbs iterations (t=4). Furthermore, the updated method obtains geometries with a slightly improved Pareto front for the example condition of Figure 6.19 Top.

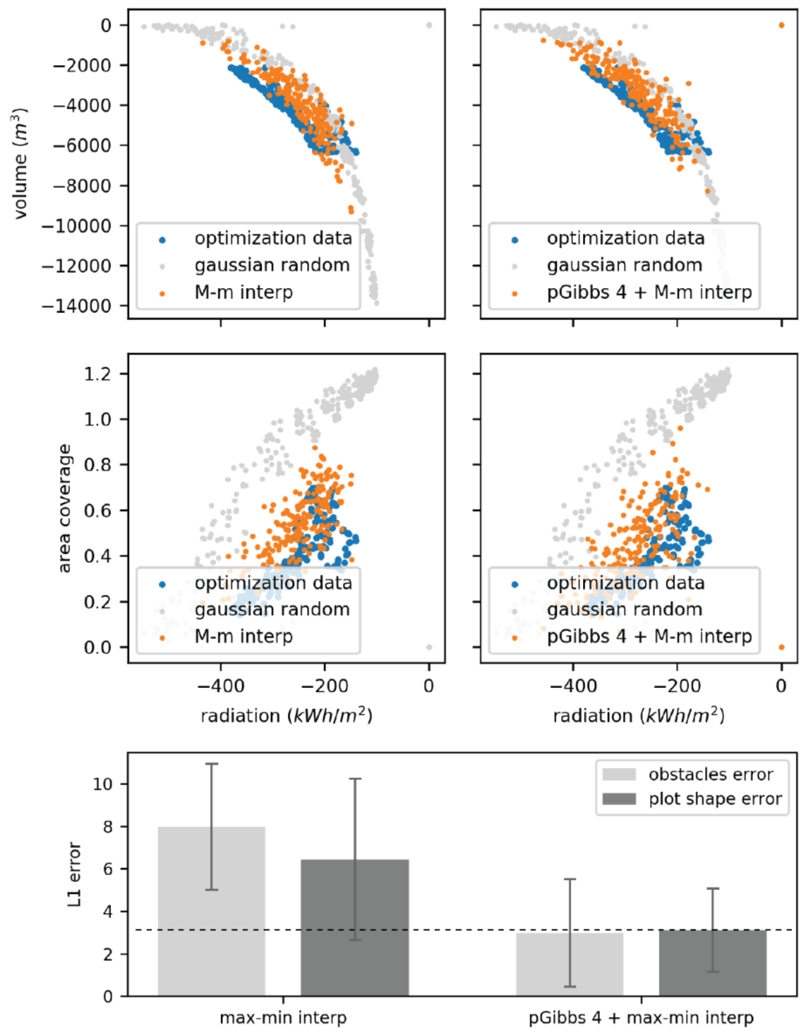


*Figure 6.19 Performance objectives (Top, Center) and boundary condition errors (Bottom) for simple max-min interpolation and max-min interpolation with pseudo-Gibbs initialization using 4 iterations (t=4).*

### *6.2.5.2. Performance*

Next, the five methods are compared regarding their ability to generate optimal geometries. Each row of Figure 6.20 refers to one of the discussed methods. The left column visualizes the distribution of solar gain and volume objectives of the VAE-generated geometries against that of the optimized geometries. The right column shows the distribution of solar gain and area coverage objectives.

Samples generated from all methods have a similar solar gain to that of the optimized data, for geometries of equivalent volume and area. The only exception is the geometries obtained using latent space optimization without a volume objective, which seem to extend the Pareto front of the optimized data for geometries of much larger size. In addition, the pseudo-Gibbs and the iterative AE dynamics methods achieve a better alignment of the area coverage objectives to the optimized data, for the boundary condition of this example.

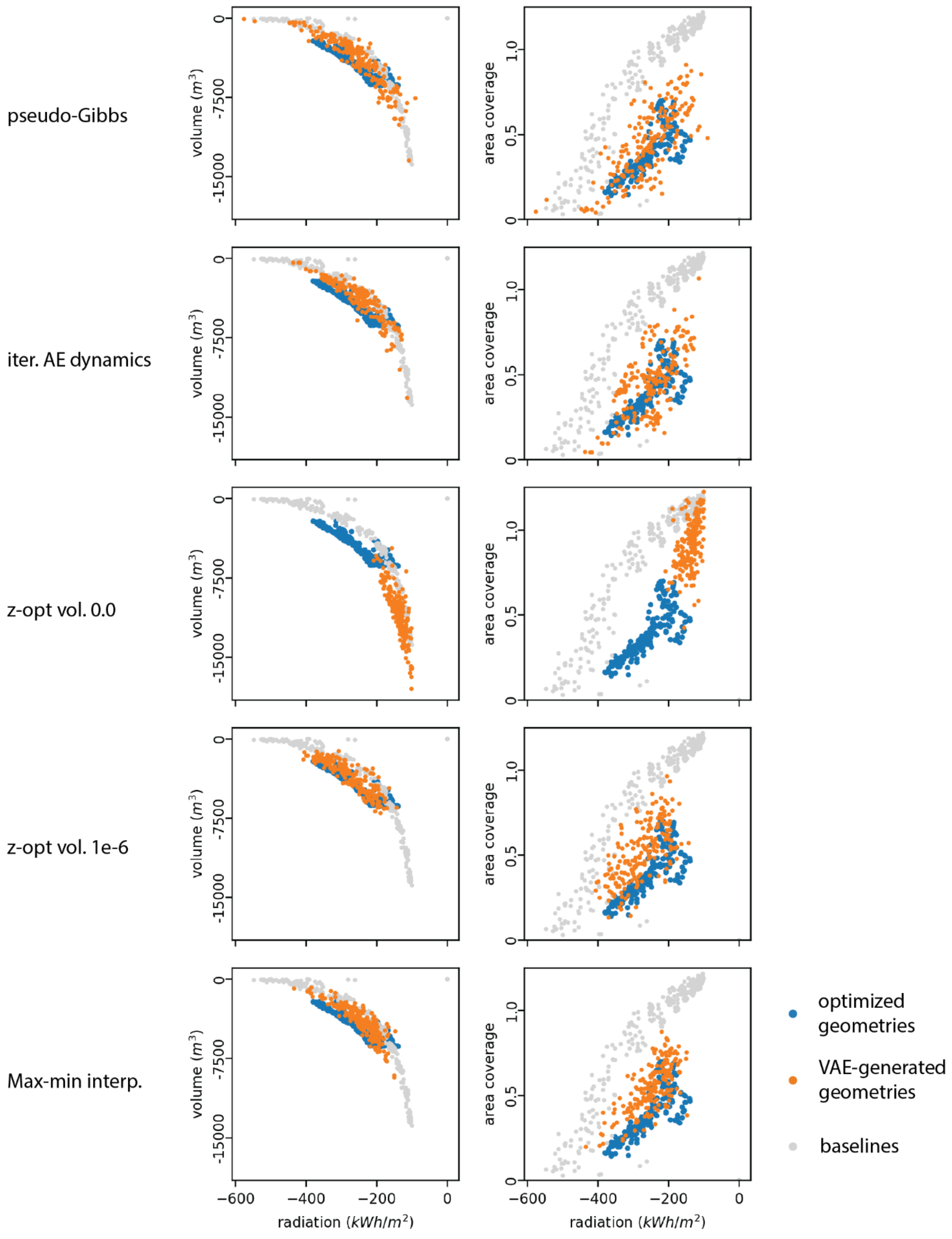


*Figure 6.20 Performance objectives for boundary condition constrained sampling using the test set.*

#### *6.2.5.3. Morphology*

Next, Figure 6.21 visualizes examples of solutions generated with each of the discussed methods. One common boundary condition was used with 6 different geometry initializations. Each random initialization was sampled in the data space from a normal distribution with mean between 0.05 and 0.45 (Figure 6.21 Top). The increasing means are expected to result in increasing size of the generated geometries. When using the max-min interpolation (last two rows), only the first and the last initializations were directly used, due to the different nature of the method.

The pseudo-Gibbs and the iterative AE dynamics methods generate geometries that are visually consistent with the optimized geometries of the dataset. However, the assumption about the correlation between the initialization and the resulting geometry size does not seem to always hold true.

The latent space optimization and the max-min interpolation methods generate forms that are visually consistent with the dataset as well. The effort to explicitly control the volume of the generated geometry resulted in the introduction of some noise, as can be seen in the last three rows of Figure 6.21. The noise is limited and can be easily removed in a post-processing step; however, it may indicate the potential for further improvement of the latent space structure. Finally, it was observed that the max-min interpolation generated smoother results when a higher number of pseudo-Gibbs iterations was used in Equation ( 6.12 ), i.e., when the extremal geometries are sampled from a probability distribution closer to the learned data distribution. Similar to Figure 6.16, no significant difference is observed between linear and spherical interpolation.

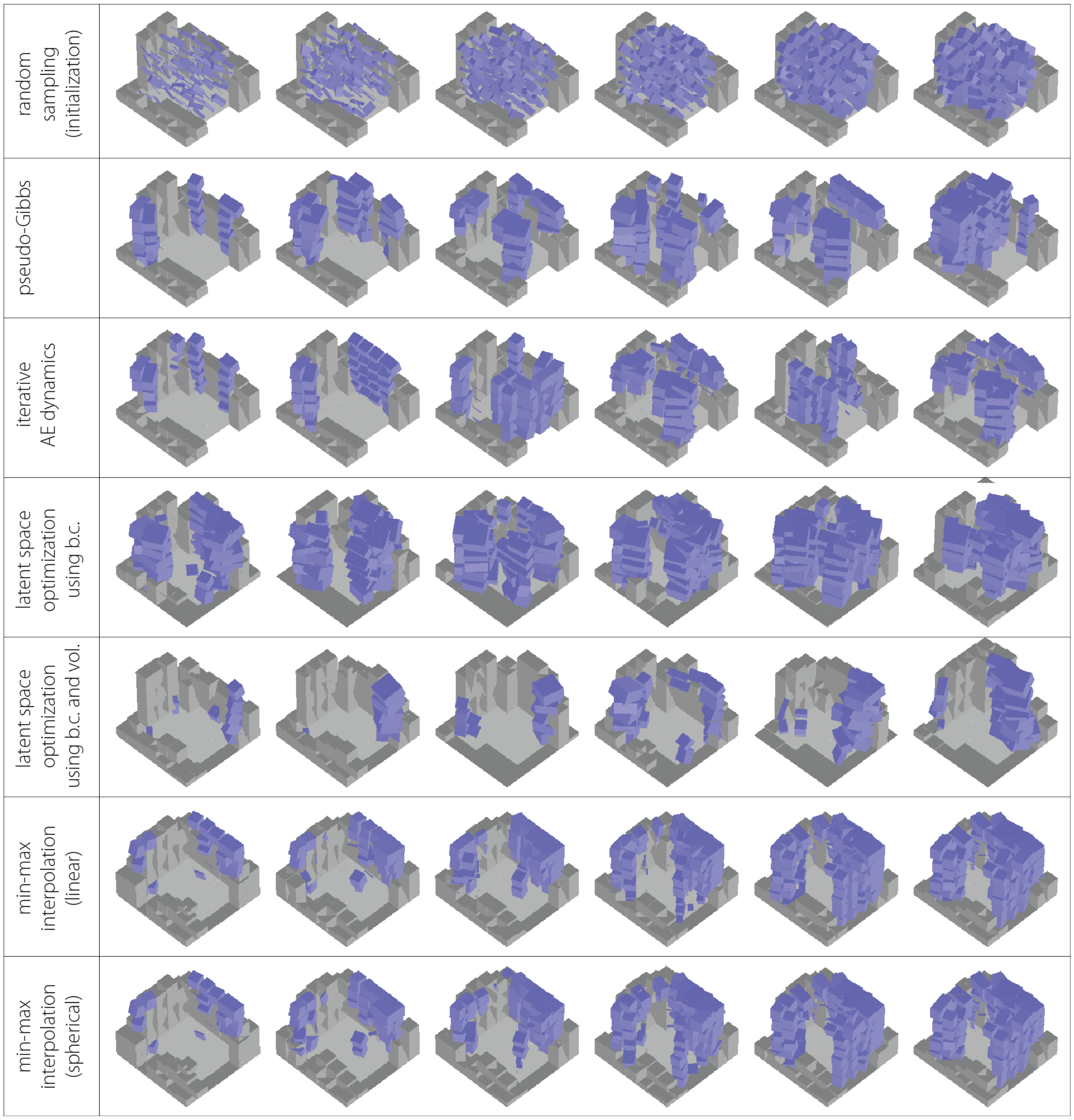

random sampling (initialization)
pseudo-Gibbs
iterative AE dynamics
latent space optimization using b.c.
latent space optimization using b.c. and vol.
min-max interpolation (linear)
min-max interpolation (spherical)

*Figure 6.21 Geometries of various sizes generated through constrained sampling of the trained VAE.*

### *6.2.5.4. Size Constraints*

The latent space optimization and the max-min interpolation methods explicitly control the size of the generated geometries through the volume factor $\gamma$ in Equation ( 6.8 ) and the interpolation value v in Equation ( 6.12 ), respectively. The size of the geometries generated using the pseudo-Gibbs and iterative AE dynamics methods has been assumed to depend on the initialization.

To better understand the relationship between the geometry initialization and the resulting volume for pseudo-Gibbs and iterative AE dynamics sampling, the initialized and generated volume were compared at various points of the respective processes, until convergence. A total of 10 different boundary conditions were used, for 60 pseudo-Gibbs and 600 AE dynamics iterations. Figure 6.22 and Figure 6.23 show two examples of how the relationship between initialized and generated volume evolved during the iterative process. Each line shows the initialized volume against the generated volume during a single iteration for 6 different geometry initializations. In some cases, such as the one depicted in Figure 6.22, a correspondence between the two volumes was roughly maintained from the beginning until the end of the process. In other cases, such as the one of Figure 6.23, a correspondence existed in earlier iterations, but was gradually lost as the 6 processes converged towards similar – or similarly-sized – geometries.

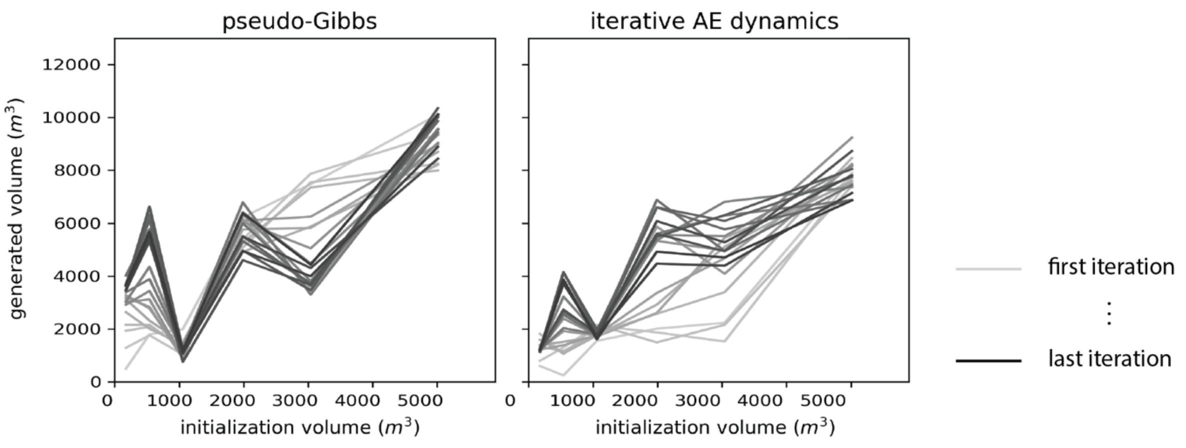


*Figure 6.22 Initialized vs generated volume for 6 geometries during different iterations. The initial volume is reflected in the final solutions.*

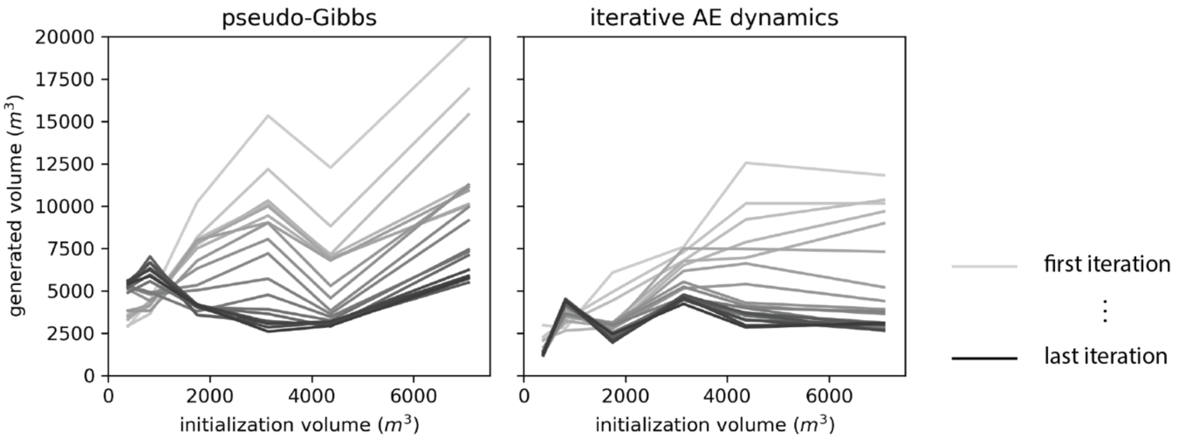


*Figure 6.23 Initialized vs generated volume for 6 geometries during different iterations. The initial volume is not reflected in the final solutions.*

Based on these observations, the geometry initialization cannot be reliably used to control the generated volume using pseudo-Gibbs or iterative AE dynamics. In absence of an alternative, it may be used as a first, rough estimate, followed by a second, size tuning step.

#### *6.2.5.5. Time*

Regarding the execution time, all methods achieve a time improvement of many orders of magnitude compared to the optimization process. Table 6.2 details the execution times of the VAE sampling methods calculated using 128 samples from the test set. Based on previous discussion about convergence, each process was run for the following number of iterations: pseudo-Gibbs: 50, iterative AE dynamics: 500, max-min interpolation: 5 pseudo-Gibbs steps, latent space optimization (with volume objective): 75 iterations. The timing for the sampling methods were produced on a Windows machine with an Intel® Core™ i5-6500 CPU @ 3.20GHZ. In addition, Table 6.2 includes the equivalent total optimization time for 200 generations, estimated based on the v2.6 dataset optimization timestamps, which was run on a faster Windows machine with an Intel Xeon CPU E5-2687W v3.

*Table 6.2*

| | 128 samples to convergence | 1 sample to convergence | 1 sample for 1 iteration |
|---|---|---|---|
| pseudo-Gibbs | 12.17 | 0.095 | 0.002 |
| iterative AE dynamics | 119.51 | 0.934 | 0.002 |
| max-min interpolation | 2.54 | 0.02 | 0.004 |
| latent space optimization | 31.27 | 0.244 | 0.003 |
| NSGA-II optimization | 582.58 | - | - |

In sum, all methods are able to generate geometries that look and perform similarly to the optimization data, in a fraction of the optimization time. The pseudo-Gibbs sampling and the iterative AE dynamics achieve the best performance alignment to the optimization performances (Figure 6.20). The max-min interpolation and the latent space optimization enable the most accurate control of the generated geometry's size. The max-min interpolation has the fastest execution time (Table 6.2); however, this is dependent on the number of pseudo-Gibbs steps. While 5 steps were enough to encode the zero and the full geometry samples, better interpolation results were achieved when the two extremal geometries were in a more confined range (min: 0.05, max: 0.45), which in turn required more pseudo-Gibbs steps (t >= 25). As a result, in the current implementation the success of the interpolation method is highly dependent on the success and the limitations of the pseudo-Gibbs sampling.

## 6.3. Sampling constrained by site and partial geometry

Many of the methods described in the previous section can be extended to take into account a partial geometry constraint. In practice this enables the generation of geometry continuations in an interactive way with the designer, resembling an ‘autocomplete’ function that preserves any user-generated geometries.

In the case of pseudo-Gibbs sampling and sampling using AE dynamics the only adaption necessary is to include the user-defined geometry in the observation data $x_{obs}$ and remove the respective cuboids from the missing data $x_{miss}$ in Equations ( 6.2 ), ( 6.3 ), and ( 6.4 ). In the case of latent space optimization, an additional, geometry-related term is added to the objective, which takes the form of Equation ( 6.13 ). The volume weight γ can be set to zero or a very small number, to allow the generation of geometries that are larger than the input.

$$\begin{aligned} J(z) = {} & L_{bc}(D(z), referenceSample) + \gamma L_{volume}(D(z), referenceSample) \\ & + aL_{geom}(D(z), referenceSample) \end{aligned} \tag{6.13}$$

$$\begin{aligned} & L_{geom}(prediction, reference) \\ & \quad = \sum_{i=1}^{nCuboids} \left(prediction_{cuboid_i} - reference_{cuboid_i}\right)^2 \\ & \quad * CuboidVolumes\left(reference_{cuboid_i}\right) \end{aligned} \tag{6.14}$$

Equation ( 6.14 ) provides the geometry loss as the squared difference of the cuboids’ parameters weighted by the given cuboids’ volumes. Intuitively, larger elements of the user-defined geometry will be better preserved in the retrieved solution. Empty areas have cuboids with volume zero and do not contribute to the geometry loss calculation. Therefore, in the generated solution, they may or may not be filled based on the latent space structure, the boundary condition loss, and the target volume.

Figure 6.24 shows examples of five geometry-constrained solutions using each of the three methods: pseudo-Gibbs sampling, iterative AE dynamics, and latent space optimization. All generated geometries comply to the constraints by reproducing the boundary condition and by including the input geometry in the solution, albeit with small deviations in the scaling and orientation parameters.

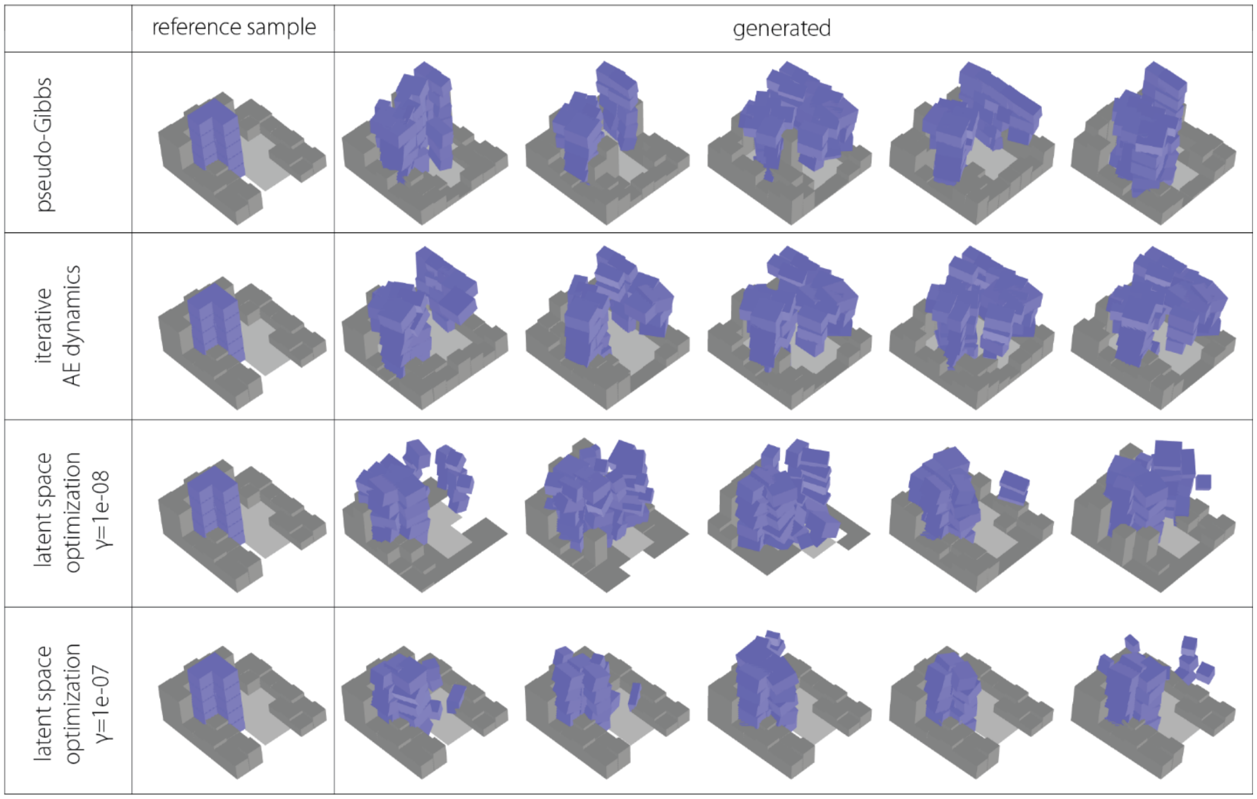

reference sample
generated
pseudo-Gibbs
iterative AE dynamics
latent space optimization γ=1e-08
latent space optimization γ=1e-07

*Figure 6.24 The first column shows an example of a user-defined geometry in a given boundary condition. The second to sixth columns show VAE-generated geometries constrained by the geometry and boundary condition of the first column. Top: pseudo-Gibbs sampling. Center: iterative AE dynamics. Bottom: latent space optimization with γ=1e-08.*

Figure 6.25 shows the boundary condition error and compliance with the input geometry for the three methods, as the iteration progresses. Contrary to the latent space optimization, which benefits from an increasing number of iterations, the pseudo-Gibbs and AE dynamics methods achieve the best results around 4 and 10 iterations respectively.

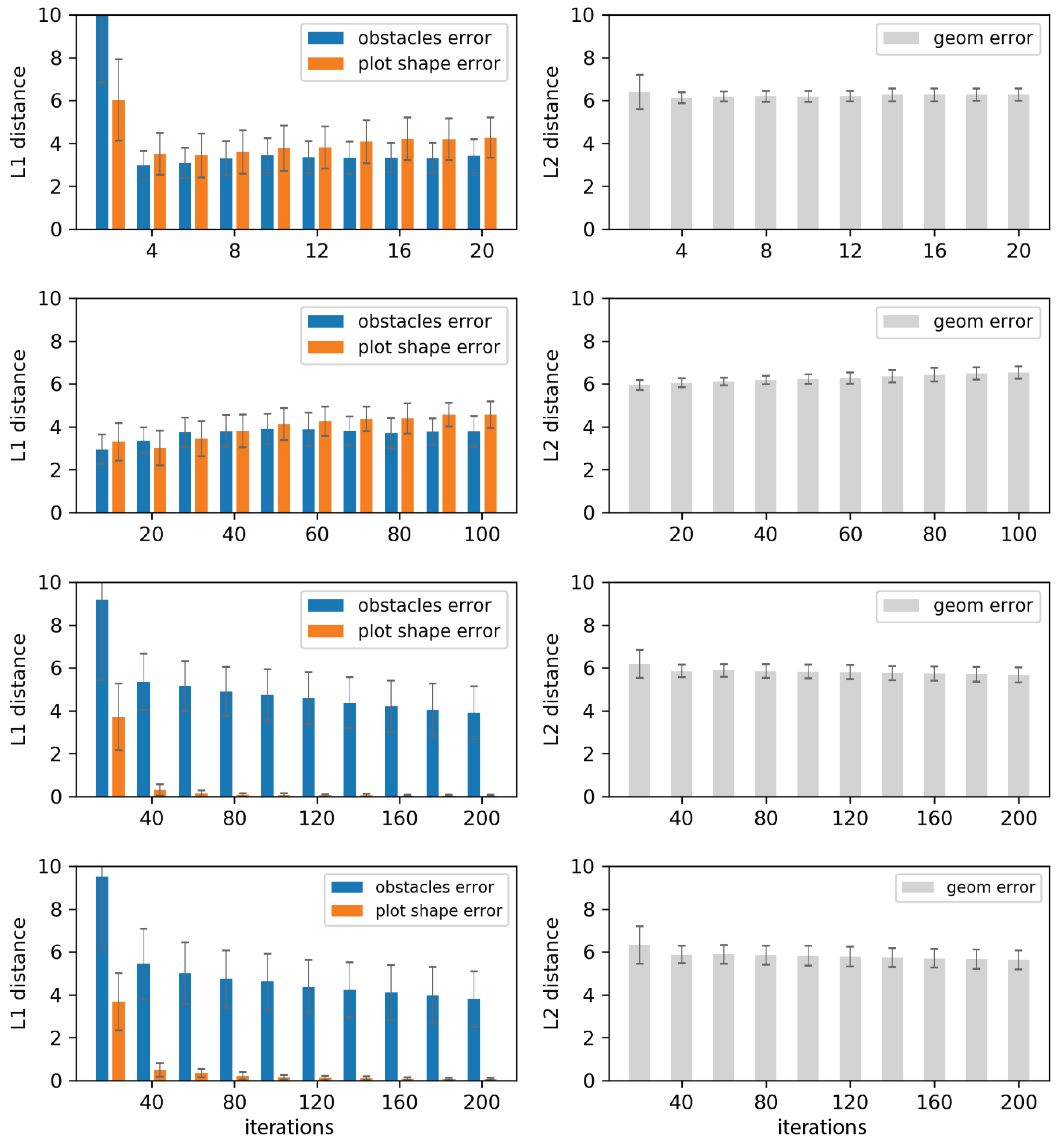


*Figure 6.25 Rows: pseudo-Gibbs sampling, iterative AE dynamics, latent space optimization with γ=1e-8, latent space optimization with γ=1e-7.*

Figure 6.26 shows the evaluation of the performance objectives (solar gain, volume, area coverage) of 100 solutions for the boundary condition and geometry constraints of Figure 6.24, using the three methods. These results are compared against the optimized geometries for the same boundary condition, coming from the test set. All three methods produce geometries with performances close to the ones of the optimized data. The geometry constraint imposes a lower bound on the size of the generated geometries, which is reflected in the volume and area coverage evaluations. The narrower spread of the iterative AE dynamics compared to the pseudo-Gibbs results hints that the former method more consistently converges to similar solutions, while the latter produces a wider variety of geometries.

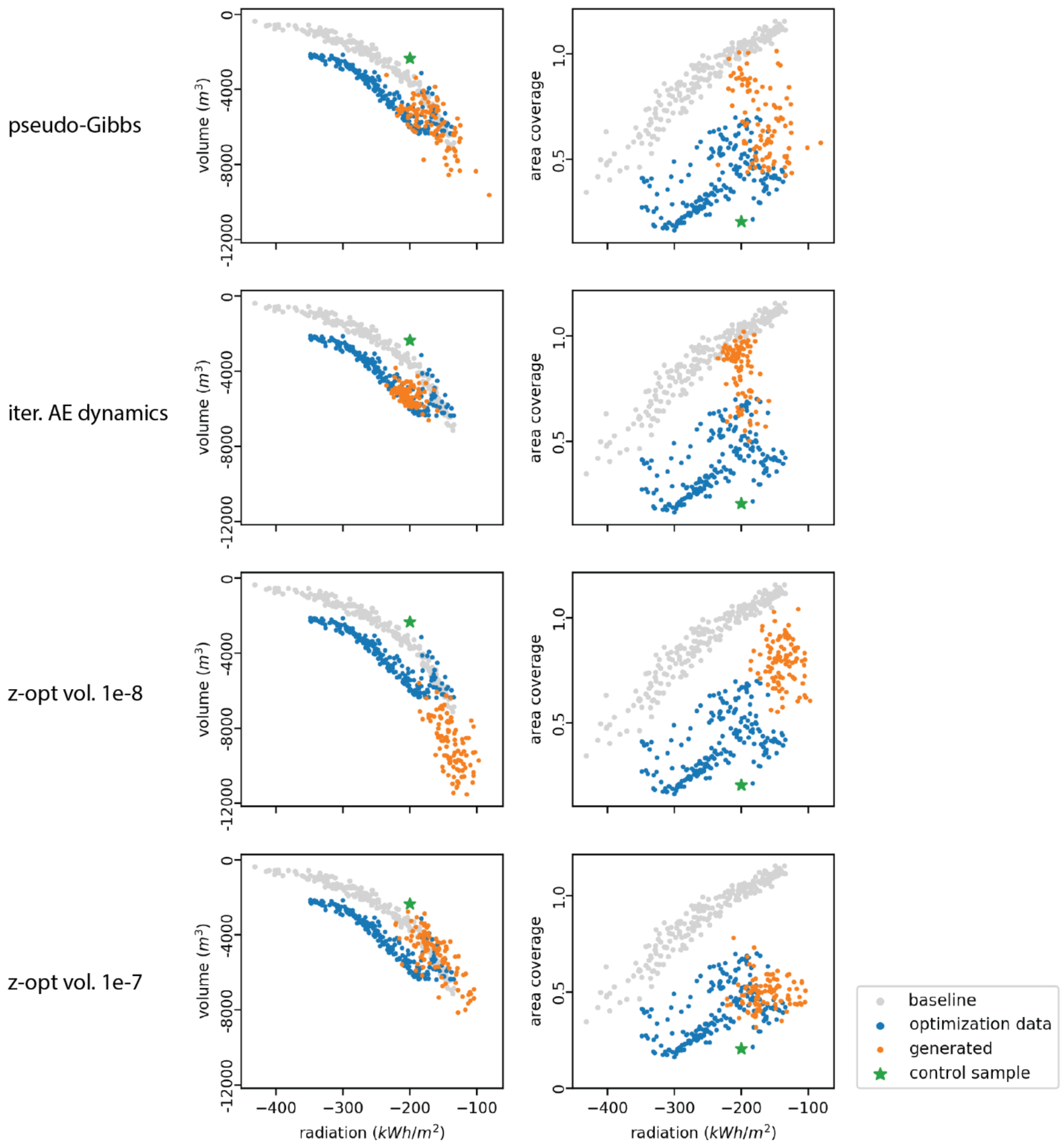


*Figure 6.26 Performance objectives of reference sample, optimized data, and geometries produced using the different constrained sampling methods..*

In the case of latent space optimization, the size of the generated geometries can be controlled by changing the value of γ in Equation ( 6.13 ). A larger γ produces geometries with a size closer to the input, as shown in Figure 6.24 and Figure 6.26.

## 6.4. Latent-space based manipulation

The structure of the learned latent space can be leveraged to manipulate samples based on several types of constraints or attributes. The previous sections already discussed how boundary condition compliance can be achieved using several methods. This section introduces methods for geometry exploration and goal-oriented tuning with respect to the generated volume, area, and solar gain performance.

### 6.4.1. Attribute vectors from subset means

Previous research on generative models has demonstrated the calculation of attribute vectors for images using labeled data (Larsen et al., 2016; White, 2016). Vectors for binary attributes, such as 'eyeglasses' or 'smile', are calculated by subtracting the mean encoding of the samples that have the attribute from the mean encoding of the samples that do not. Data samples can then be manipulated by applying this vector to their latent representations. The resulting images preserve the general appearance of the originals while being updated only with respect to the targeted feature.

In this work, the attributes of interest (geometry volume, area, solar gain objective) are continuous. Therefore, to compute the corresponding latent vectors, subsets of the data containing the samples with the highest and lowest values for each attribute can be selected in place of the binary-classified data. Specifically, for each attribute, a latent vector can be created by subtracting the mean encodings of the data from the top and bottom 10% of the training set sorted by the selected attribute. Equation ( 6.15 ) shows how sample $x_i$ is generated as a modification of an existing sample $x_0$ by applying the latent space attribute vector $z_{attribute}$. The factor α is controlling the strength of the modifier. The term f<sub>attribute</sub> is a normalizing factor, empirically determined for each attribute.

$$x_i = D(z_0 + a * f_{attribute} * z_{attribute}), a\colon [-1, 1], z_0 \sim E(x_0) \tag{6.15}$$

This method for constructing attribute vectors is known to suffer from correlations that may exist in the data (Larsen et al., 2016; White, 2016). One such example is the inherent correlation between geometry volume and area coverage. An initial effort to decouple the two attributes was made by constructing a balanced dataset through data replication, following White (White, 2016). Table 6.3 shows how the area coverage means of the two subsets – containing geometries with low or high volume – were brought to similar levels following this process.

*Table 6.3*

| | Initial volume mean | Initial area coverage mean | Balanced volume mean | Balanced area coverage mean |
|---|---|---|---|---|
| Low volume set | 1037 | 0.37 | 1046 | 0.49 |
| High volume set | 7511 | 0.49 | 7486 | 0.48 |

Other correlations exist in the dataset as well. Notably, the boundary conditions affect the size of the optimized geometries. More specifically, the size of the building plot directly imposes area and volume constraints on the geometries during the dataset generation process. In addition, the number and height of the obstructions may bias optimization results toward larger or smaller geometries as well. As a result, sampling geometries of maximum and minimum volume from the whole training set creates subsets that are biased with respect to the boundary conditions. To remove these biases from the resulting attribute vectors, the sampling process for the high and low volume subsets was updated to include an equal number of samples from each boundary condition. Finally, the updated subsets were balanced for area coverage using data replication.

As a baseline, data space attribute vectors were also computed by following an equivalent process directly in the data space, without encoding the data. Equation ( 6.16 ) shows how sample $x_i$ is generated by directly applying data space attribute vector $x_{attribute}$ to the original sample $x_0$.

$$x_i = x_0 + a * f_{attribute} * \ x_{attribute}, a: [-1, 1] \tag{6.16}$$

## 6.4.2. Gradient-based methods

Another way to manipulate data samples in a goal-driven way is using gradient-based methods. No labels are required; however, the attributes of interest need to be able to be calculated in a differentiable way. From the three attributes of interest in this work (volume, area, radiation objective) only the volume (Equation ( 5.3 )) and an approximation of the area (Equation ( 5.4 )) can be derived using differentiable functions.

Starting from a given data point $x_t$ (geometry$_t$, obstructions$_t$, plot$_t$) with a latent representation $z_t$~E($x_t$), a new sample $z_{t+1}$ can be obtained by taking one step in the direction of the attribute gradient in the latent space as shown in Equations ( 6.17 ) and ( 6.18 ) for volume and area modifications respectively. The sample $x_{t+1}$ (geometry$_{t+1}$, obstructions$_{t+1}$, plot$_{t+1}$) is obtained by decoding $z_{t+1}$. The terms $f_{gradVol}$, $f_{gradArea}$ are normalizing factors. Larger modifications are achieved by taking multiple steps in the latent space in an iterative way.

$$z_{t+1} = z_t + f_{gradVol} * \nabla Vol\big(D(z_t)\big) \tag{6.17}$$

$$z_{t+1} = z_t + f_{gradVol} * \nabla Area_{proj}\big(D(z_t)\big) \tag{6.18}$$

During gradient-based geometry modification, a boundary condition constraint can be applied to ensure a latent space trajectory that corresponds to samples within the solution space of the original boundary condition. In addition, an area or volume constraint can be added for the volume-driven or area-driven modifications respectively, to encourage the decoupling of the two attributes. For example, Equation ( 6.17 ) takes the generalized form of Equation ( 6.19 ), where $f_{vol}$, $f_{bcConst}$, $f_{areaConst}$ are the weights for the volume, boundary condition difference, and area difference, and $f_{grad}$ is a normalizing factor.

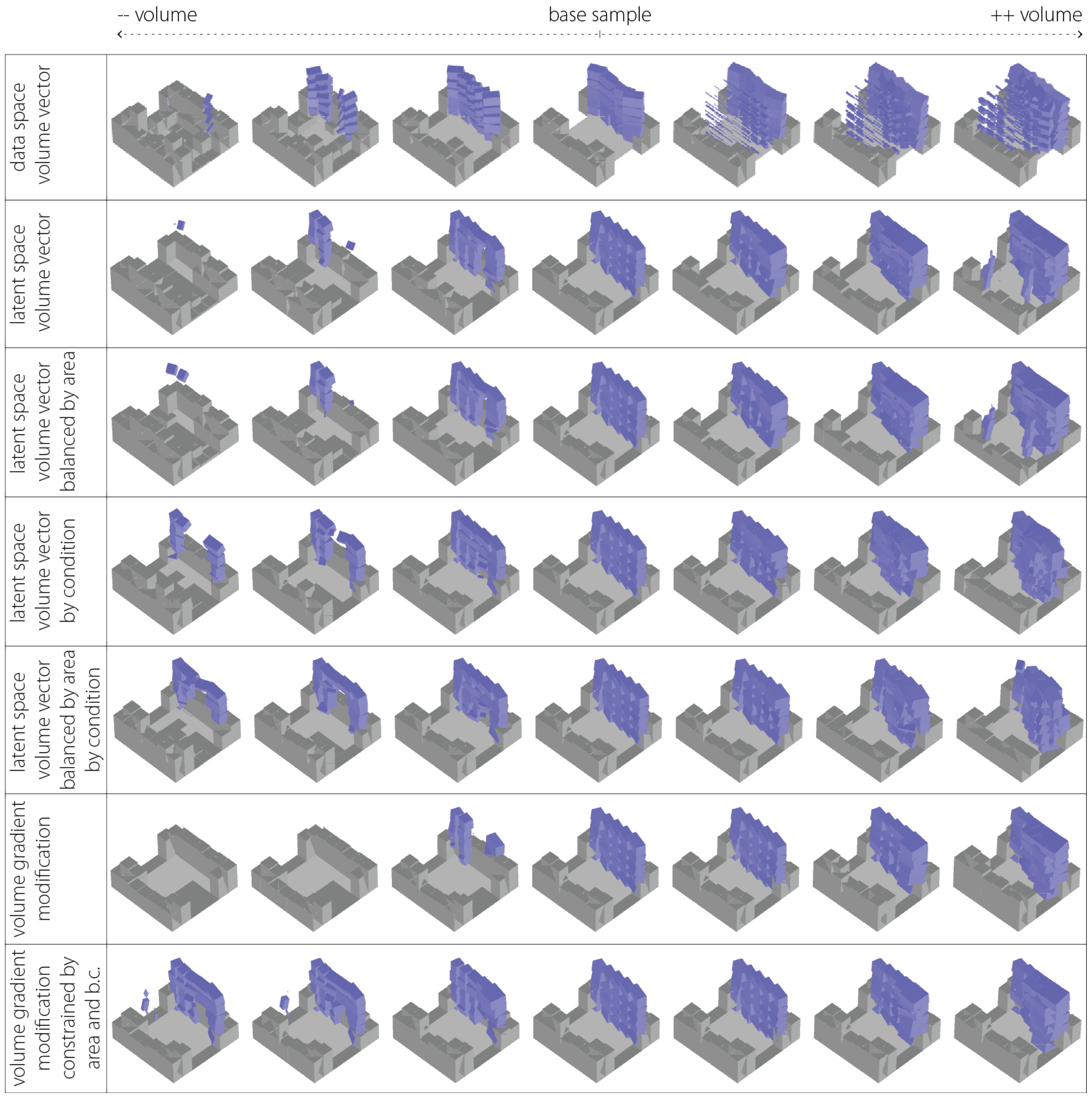
$$\begin{aligned} z_{t+1} = z_t + f_{grad} \\ * \nabla\Big(f_{vol} * Vol\big(D(z_t)\big) + f_{bcConst} * L_{bc}(D(z_t), x_0) + f_{areaConst} \\ * \big(Area_{proj}(D(z_t) - Area_{proj}(x_0)\big)^2\Big) \end{aligned} \tag{6.19}$$

## 6.4.3. Evaluation

This subsection presents examples and compares results of geometries generated using the goal-driven geometry exploration methods presented above. Eight samples with unique boundary conditions were selected from the test set and their geometries were modified by applying all the above methods, regarding their volume, area, and solar gain performance. Figure 6.27, Figure 6.29, and Figure 6.31 depict seven snapshots of their volume-, area-, and solar gain-driven modifications respectively. Figure 6.28, Figure 6.30, and Figure 6.32 visualize their respective boundary condition errors and geometry attribute evaluations. All performance evaluations were performed using the original boundary conditions, regardless of potential boundary condition deviations in the modified samples.

Figure 6.27 shows volume-driven modifications for a single sample from the test set. The first row is using the volume attribute vector calculated in the data space, as a baseline. While the total volume of the geometries does seem to monotonically increase from left to right, the generated geometries do not seem to follow the distribution of the dataset, with respect to their morphological characteristics. Geometries of increased volume are noisy, with tiny boxes added into the previously empty space. Geometries of decreased volume appear noisy and fragmented. In addition, the boundary conditions of the smaller geometries are significantly deviating from the original one.

In contrast, all methods that operate in the latent space produce geometries that morphologically resemble the ones in the dataset, while also achieving a monotonically increasing volume from left to right. Rows 4 and 5, which are generated from latent space attribute vectors using evenly sampled boundary conditions, replicate the plot shape and obstructions of the original sample better than rows 2 and 3, which are generated from latent space vectors using the global minima and maxima of the dataset. In addition, row 5, which is generated using a volume vector balanced for area coverage, prioritizes modifications that change the volume without affecting the area. This is better illustrated in the first three columns, where boxes from the top level are preserved, achieving an almost constant area while changing the total volume of the geometry. Last, rows 6 and 7 are generated using gradient-based methods. Row 7 is using boundary condition and area constraints. The boundary condition constraint is visibly improving the replication of the plot shape for smaller geometries, compared to all other methods.



-- volume
base sample
++ volume
data space volume vector
latent space volume vector
latent space volume vector balanced by area
latent space volume vector by condition
latent space volume vector balanced by area by condition
volume gradient modification
volume gradient modification constrained by area and b.c.

*Figure 6.27 Volume-driven modifications.*

Figure 6.28 quantifies how the volume, area, plot shape, and obstructions heightmap is affected when a volume-driven modification is applied to a base sample, using the several methods discussed above. For each method, the respective normalizing factors and number of iterations have been tuned so that all methods produce geometries with approximately matching ranges of volume modifications. This is important to make comparisons possible.

In general, the observations that were made for Figure 6.27 are supported by the quantitative results. More specifically, the attribute vectors that have been evenly sampled from all boundary conditions reduce the plot shape and obstruction replication errors compared to the attribute vectors that have been sampled globally. The overall boundary condition error is minimized using the gradient-based method with the relevant constraint, which is the only method to explicitly apply such a correction.

When comparing the effect of volume modification to resulting area coverage, only small differences have been achieved. The best results come from the gradient-based method with area constraint for volume increase, and from the volume latent attribute vector, sampled by boundary condition and balanced by area, for volume reduction. The small effect size is expected due to the inherent interdependence of the two characteristics, as well as limitations related to the dataset.

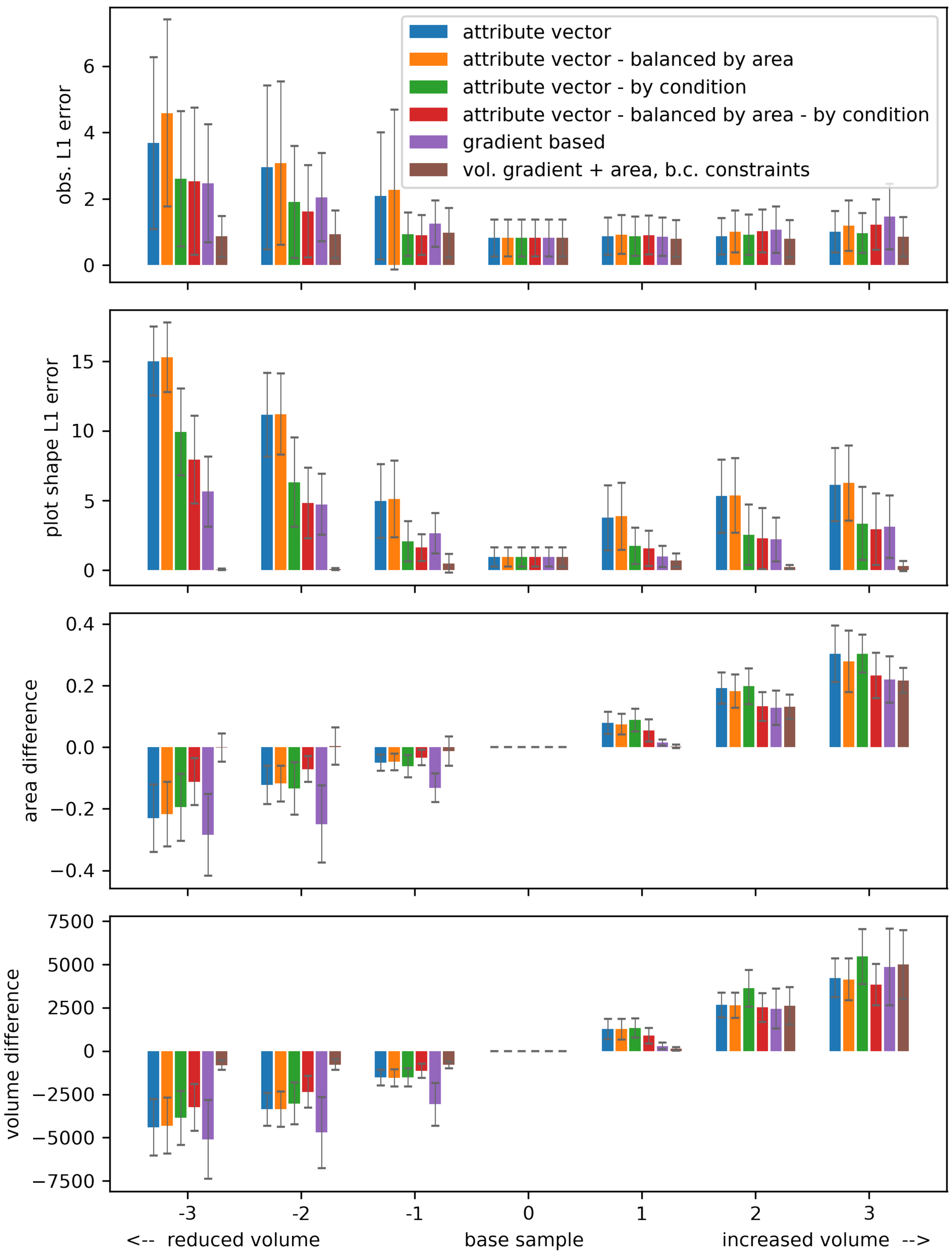


*Figure 6.28 Generated obstructions and plot shape error from the original samples $X_0$. Area and volume difference of the generated samples from the base reconstruction $D(Z_0)$.*

Figure 6.29 depicts 7 snapshots of area-driven modifications for a single test set sample, following the structure of Figure 6.27. Similar to the volume-driven modifications, the attribute vector computed in the data space fails to produce meaningful results. All methods that operate in the latent space produce geometries with reasonable morphologies and area coverage that changes as expected.

Efforts to decouple area and volume attributes do not have a visible effect for geometries with reduced area from the base (rows 3, 5, 7). This is because the base geometries already occupy the full available height, which does not allow for a relative volume increase when the area is reduced. On the other hand, generated geometries for area coverage higher than the base tend to show a reduction of mass at their lower level in rows 3, 5, 7. This is the result of an effort to maintain the total volume constant, while the area increases.

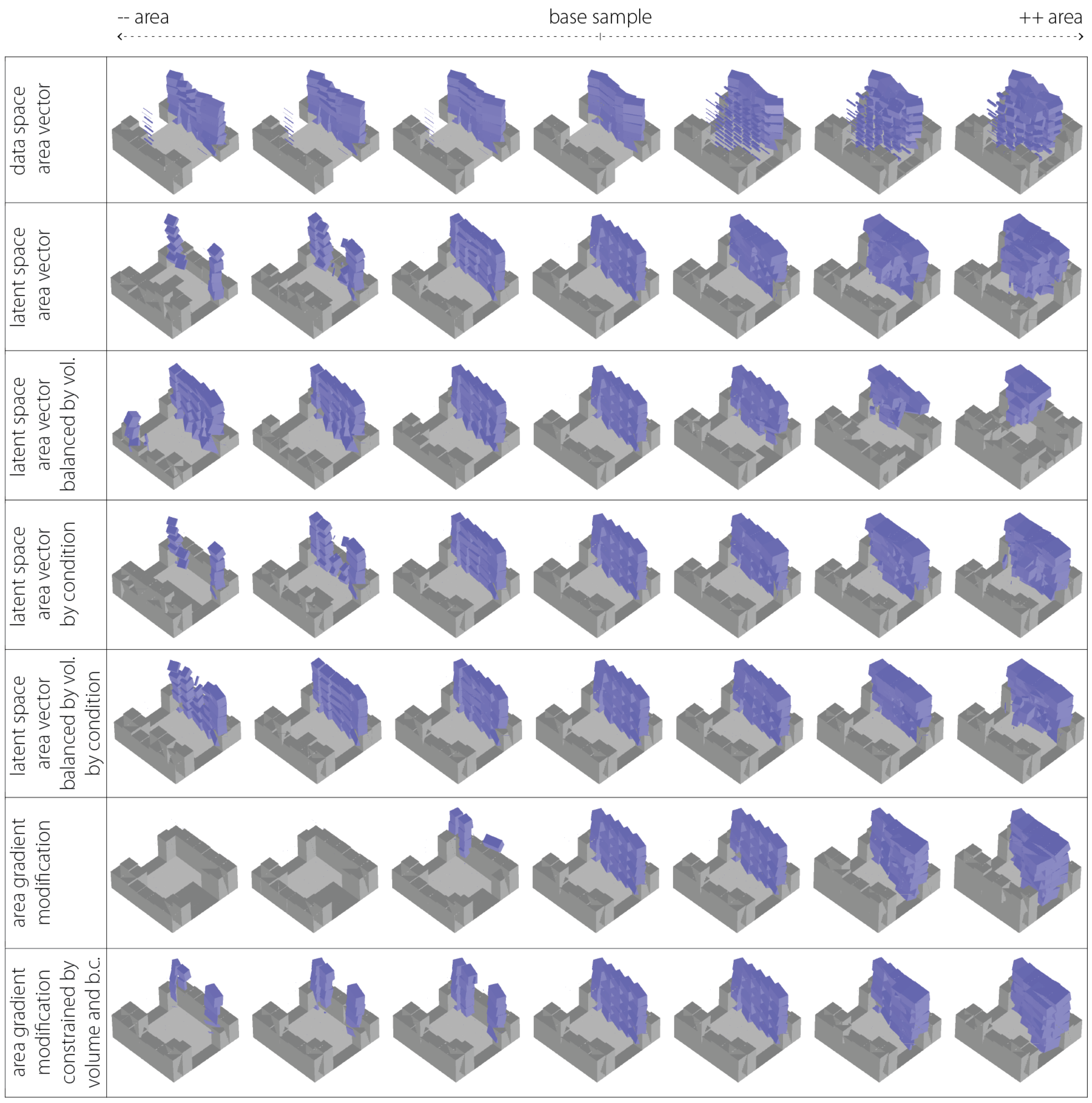

-- area
base sample
++ area
data space area vector
latent space area vector
latent space area vector balanced by vol.
latent space area vector by condition
latent space area vector balanced by vol. by condition
area gradient modification
area gradient modification constrained by volume and b.c.

*Figure 6.29 Area-driven modifications.*

Figure 6.30 quantifies the effect of the area-driven modifications. It shows that the area latent space attribute vector achieves an almost constant volume for a small increase in the area coverage, which is also evident in the last columns of row 3 in Figure 6.29. However, this happens at the expense of boundary condition replication accuracy, as observed in the top two charts of Figure 6.30. The best overall performance is achieved by the latent space attribute vector evenly sampled by boundary condition and balanced by volume, which has a low mean error for obstructions and plot shape reconstruction and increases the area coverage with a proportionally small change in the total volume. The gradient-based method with constraints has the best boundary condition reconstruction but is not as successful at maintaining a constant volume while the area coverage changes.

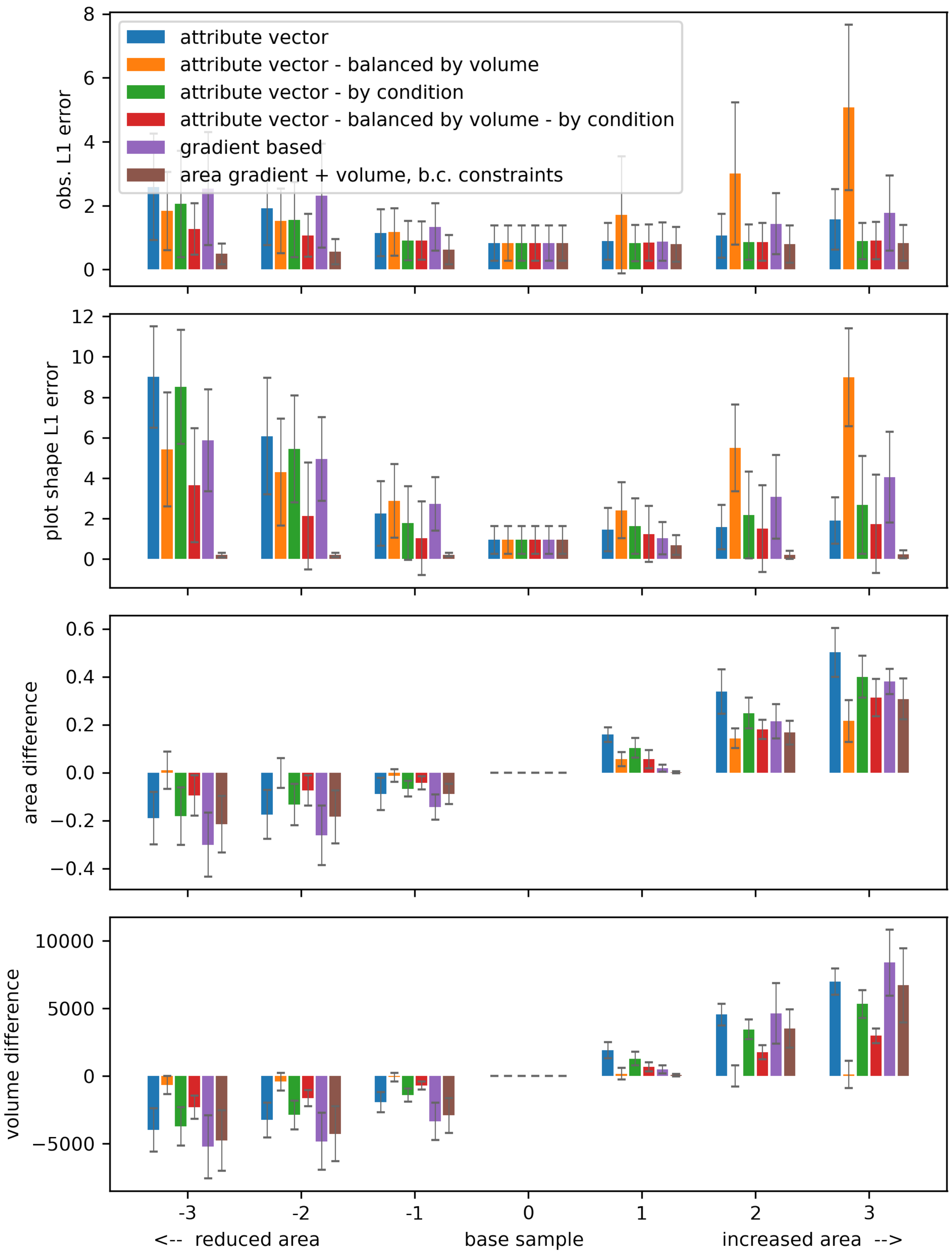
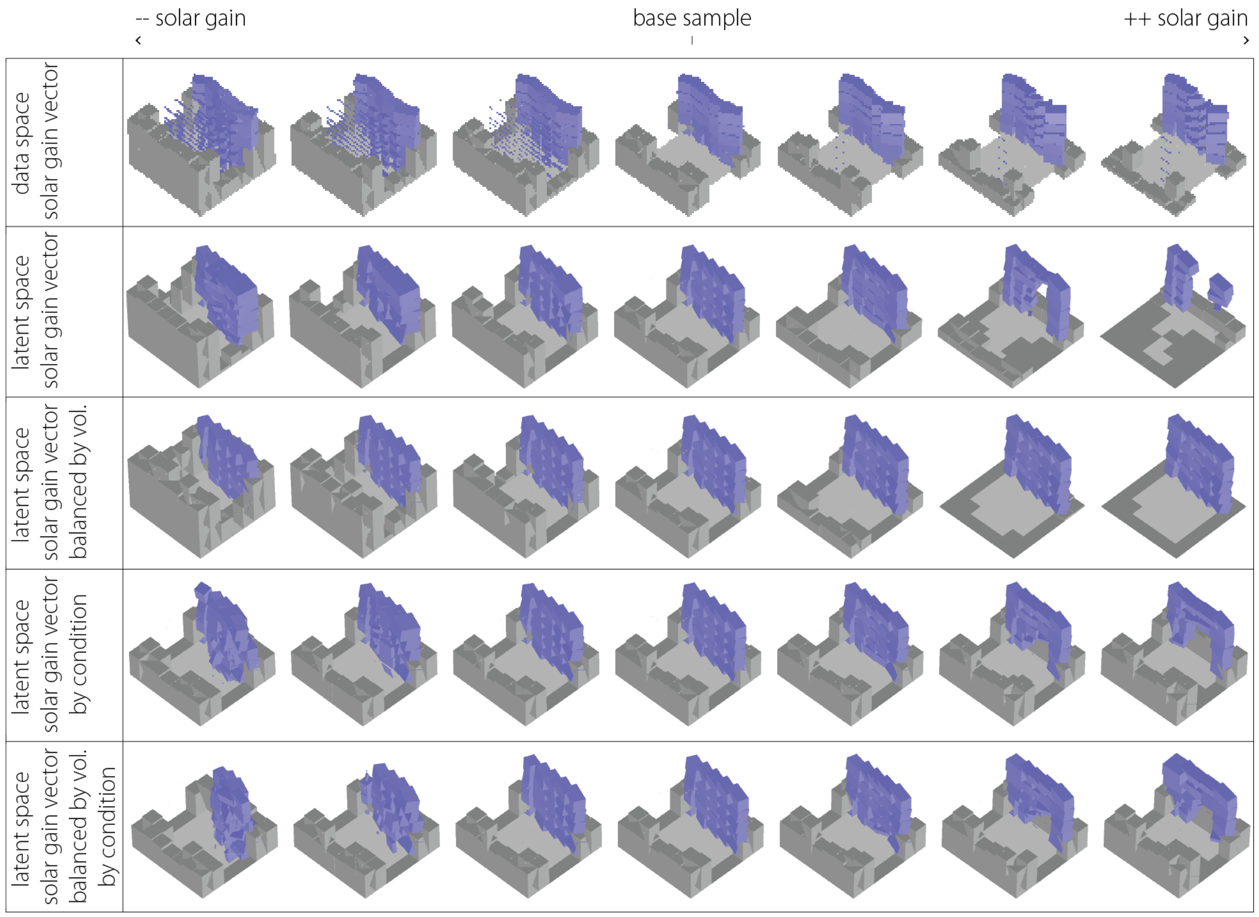


*Figure 6.30 Generated obstructions and plot shape error from the original samples $X_0$. Area and volume difference of the generated samples from the base reconstruction $D(Z_0)$.*

Last, regarding the solar gain-driven geometry modifications, gradient-based methods could not be applied due to the lack of an appropriate differentiable function. Figure 6.31 shows that similar to the previous examples, the data space attribute vector is only adding noise without significant changes to the original geometry. Row 2 shows a correlation between volume and solar gain performance, while rows 2 and 3 show a strong correlation between boundary conditions (south obstructions) and solar gain. The latent space vectors evenly sampled across boundary conditions alleviate these problems, as shown in rows 4 and 5. The solar gain performance change for these geometries seems to target morphological characteristics, such as the overall proportions and the orientations of the individual boxes.



-- solar gain
base sample
++ solar gain
data space solar gain vector
latent space solar gain vector
latent space solar gain vector balanced by vol.
latent space solar gain vector by condition
latent space solar gain vector balanced by vol. by condition

*Figure 6.31 Solar gain-driven modifications.*

Figure 6.32 confirms that the solar gain latent space attribute vectors have the expected effect on the solar gain performance of the resulting geometries. The only exception is the vector that has been globally sampled and balanced by volume, which does not have a significant effect on the solar gain. This provides evidence that the solar gain effect of the unbalanced globally sampled vector is driven solely by the size of the geometry. In addition, the disproportionately large effect of both globally sampled vectors on the boundary condition shows the limitations of this sampling method regarding correlated data attributes.

On the other hand, both vectors that have been evenly sampled across boundary conditions minimize the obstructions and plot shape errors. They also achieve higher solar gain improvements, while keeping the volume changes at lower levels.

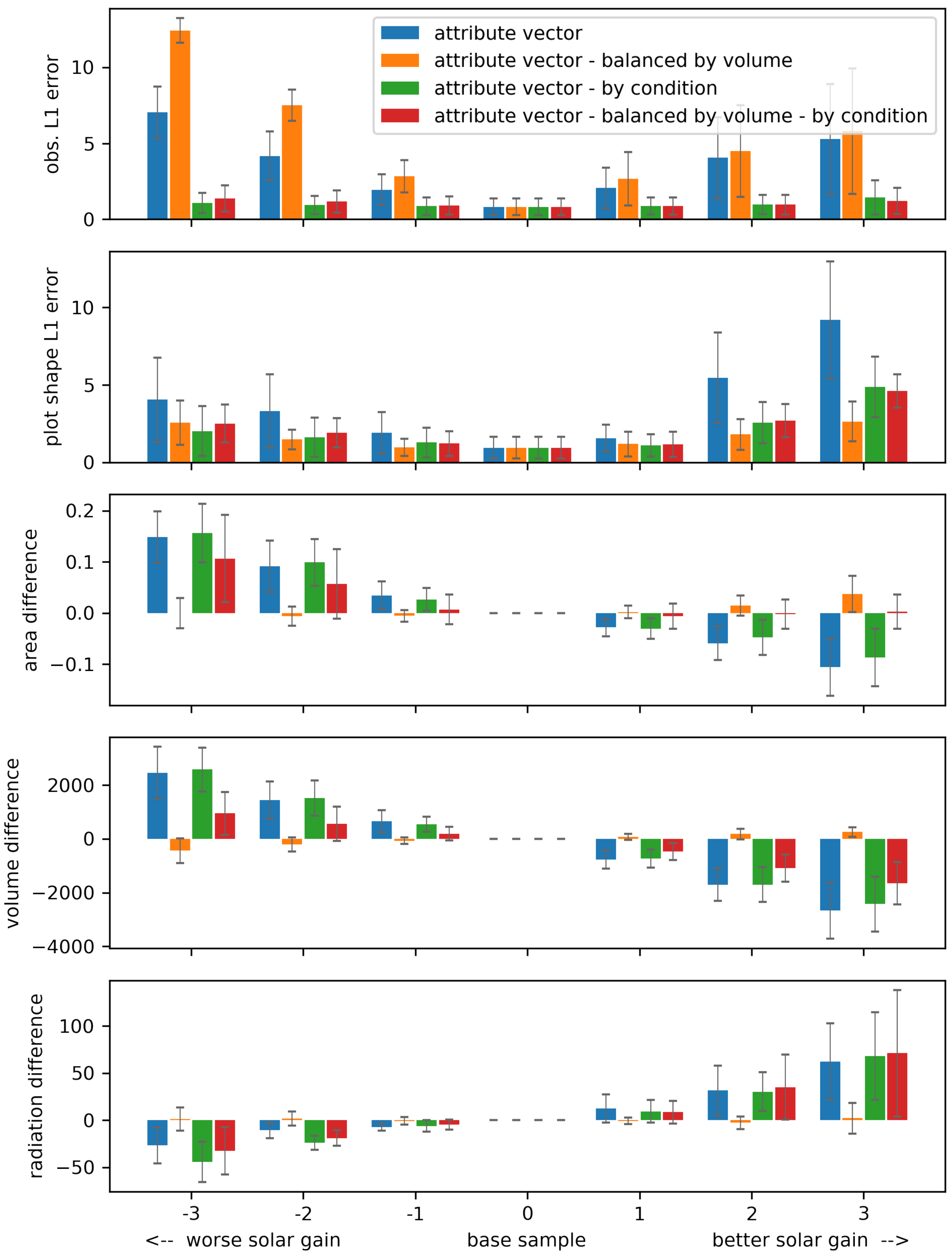


*Figure 6.32 Generated obstructions and plot shape error from the original samples $X_0$. Area, volume, and solar gain difference of the generated samples from the base reconstruction $D(Z_0)$.*

# 7. Prototype

This chapter describes a prototype developed to demonstrate how the introduced framework can be leveraged to assist the early form-finding process.

The prototype takes the form of a design tool with its own 3d modeling environment. The tool suggests optimal geometries in response to the current site and any already designed forms. Building size constraints can be met by controlling the volume and area of the generated geometry. Last, the generated geometries can be exported for use with any 3d modeling software.

The prototype is implemented using a client-server architecture. On the client-side, a web-based front end presents the designer with all the UI controls and implements a WebGL 3d-modeling environment. On the server-side, a python-based backend implements a Flask server and uses the trained VAE model with the introduced sampling methods to generate predictions based on the user inputs.

## 7.1. Outline

The interaction of the designer with the system is structured into three levels: generate, tune, and explore, outlined in Figure 7.1. During the generation phase, the designer first sets up the building site. Design preferences or constraints such as built and unbuilt areas may also be defined. Last, the designer selects the approximate size of the building. The system generates a geometry in response. During the tuning phase, the designer can control the compliance to the geometric and size constraints. Last, during the exploration phase, the designer can explore what the generated geometry would look like if it had a different volume, area, solar gain performance, or morphology. The goal of this step is twofold. First, it can provide inspiration, challenge the defined constraints, and possibly help redefine some of the problem's objectives. Second, it allows a glimpse into the structure of the underlying solution space landscape, helping the designer understand the tradeoffs between objectives as they have been encoded into the system.

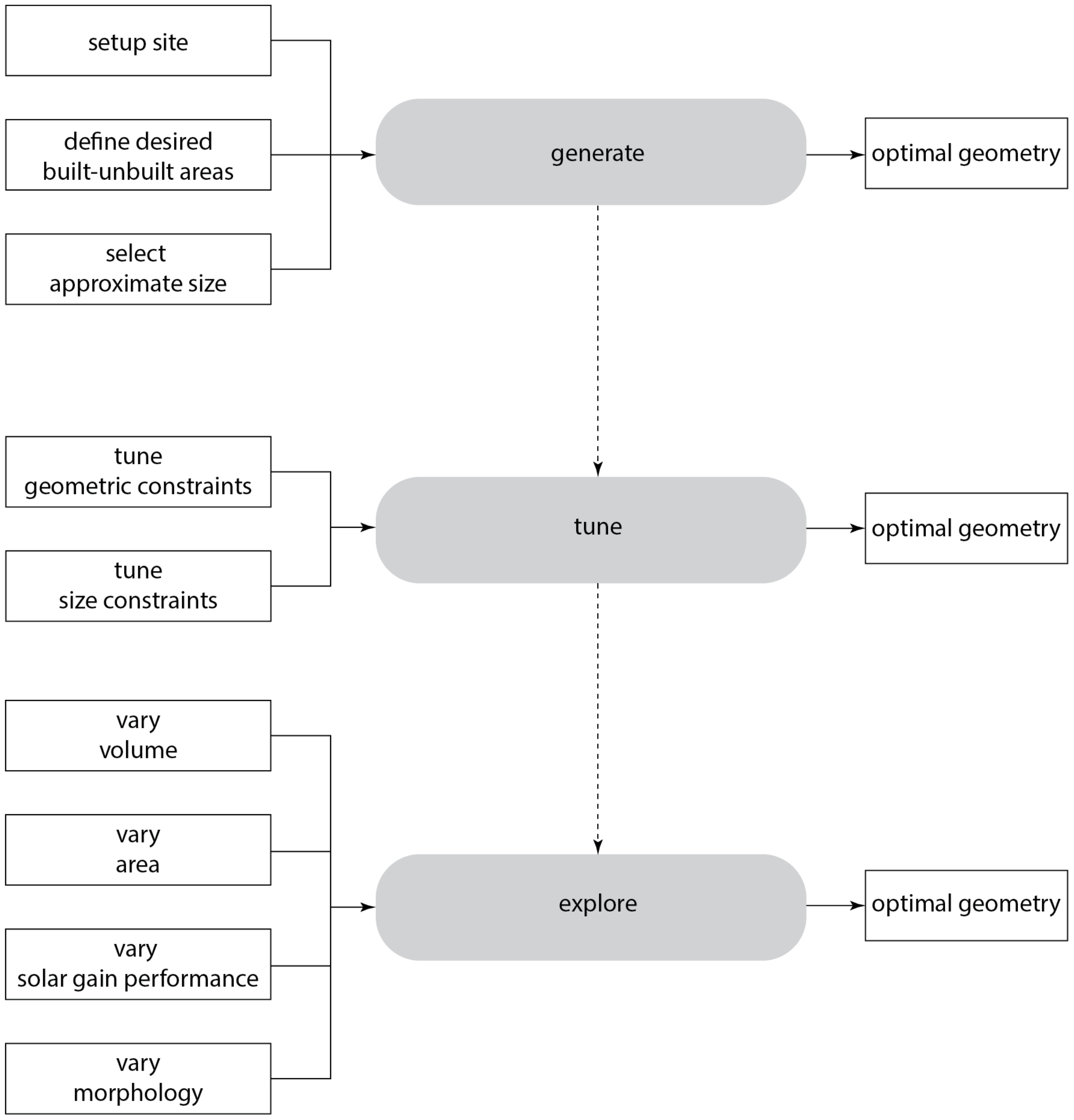


*Figure 7.1 Outline of the prototype interaction in three steps: generate, tune, explore.*

## 7.2. Processes

This section gives a more detailed account of the prototype's internal structure and the methods implemented to enable the behavior described above. The detailed outline is shown in Figure 7.2.

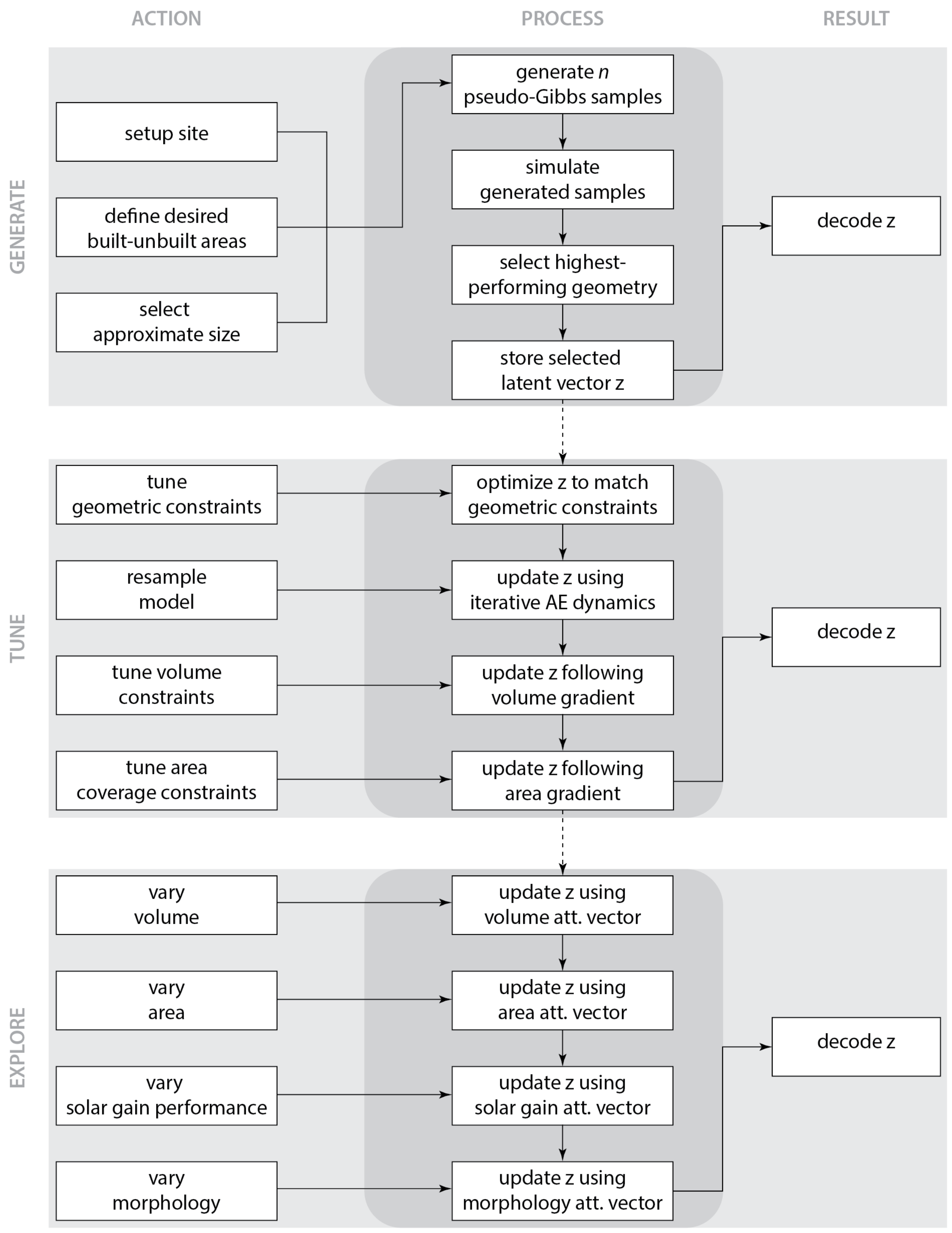

ACTION
PROCESS
RESULT
GENERATE
setup site
define desired built-unbuilt areas
select approximate size
generate *n* pseudo-Gibbs samples
simulate generated samples
select highest-performing geometry
store selected latent vector z
decode z
TUNE
tune geometric constraints
resample model
tune volume constraints
tune area coverage constraints
optimize z to match geometric constraints
update z using iterative AE dynamics
update z following volume gradient
update z following area gradient
decode z
EXPLORE
vary volume
vary area
vary solar gain performance
vary morphology
update z using volume att. vector
update z using area att. vector
update z using solar gain att. vector
update z using morphology att. vector
decode z

*Figure 7.2 Detailed outline of the prototype and the implemented methods.*

The generate phase uses the pseudo-Gibbs sampling method to generate *n* samples; the current implementation uses n=8. The approximate size, controlled by a slider, defines the noise intensity for the random initialization of the pseudo-Gibbs sampling. In order to take into account the desired built and unbuilt areas, the corresponding grid cells are considered as observed data, together with the plot shape and obstructions, in Equation ( 6.2 ). Each of the *n* samples is initialized by sampling a normal distribution with a mean which depends on the selected approximate size, and pseudo-Gibbs sampling is performed for 5 iterations. Next, all generated geometries are simulated in the original boundary conditions using multi-processing, and the solution with the best solar gain performance is selected. The latent vector $z_{gen}$ is stored on the server and the decoded sample is rendered on the front end.

The tuning phase uses gradient-based methods to meet the constraints using two loops with an optional resampling step in between. First, if further enforcement of the geometric constraints is necessary, Equation ( 7.1 ) is used. The error referring to the compliance of the generated geometry to the user-defined built area $L_{geom}$ is defined in Equation ( 6.14 ), and the error for the user-defined unbuilt $L_{void}$ is defined in Equation ( 7.2 ). The number of iterations $t$, the step size $\alpha$, weight factors for built and unbuilt areas $f_{geom}$ and $f_{void}$, as well as the weight factor for volume compliance of the generated geometry to the user-defined built, are all controlled by the user through sliders. In addition, to ensure that the generated solutions remain within the limits of the defined site, the boundary condition error $L_{bc}$ is used, weighted by the factor $f_{bcConst}$. The starting point $z_0$ is defined as the latent representation $z_{gen}$ of the solution acquired during the generate phase.

$$z_{t+1} = z_t + \alpha * \nabla\Big(f_{geom} * L_{geom}(D(z), x_0) + f_{void} * L_{void}(D(z), x0) + f_{vol} * \big(Vol(D(z_t)) - Vol(x_0)\big)^2 + f_{bcConst} * L_{bc}(D(z_t), x_0)\Big) \tag{7.1}$$

$$L_{void}(prediction, reference) = \sum_{i=1}^{nCuboidsVoid} \left(prediction_{cuboid_i} - reference_{cuboid_i}\right)^2 \tag{7.2}$$

When a high number of iterations is necessary to meet the geometric constraints, the generated solutions may appear noisy. Previous research has suggested that the latent space of generative models may contain zones that do not correspond to the data distribution of the training data (White, 2016). Therefore, if the resulting $z_t$ drifts away from the initial $z_{gen}$, it may land in such a zone. To overcome this problem, an optional resampling step has been provided after the tuning of the geometric constraints, implemented as a VAE sampling using the iterative Autoencoder dynamics method. Previously generated geometries are considered as observed data, along with the plot shape and obstructions, same as with pseudo-Gibbs sampling. Both the smoothing factor *h* and the number of sampling steps *t* of Equation ( 6.4 ) can be controlled using sliders.

Next, the volume and area coverage constraints can be satisfied by modifying the current geometry using another gradient descend loop, detailed in Equation ( 7.3 ).

$$z_{t+1} = z_t + \alpha * \nabla\Big(f_{vol} * Vol\big(D(z_t)\big) + f_{area} * AreaProj\big(D(z_t)\big)\Big) \tag{7.3}$$

Last, during the explore phase, the resulting geometry is modified using latent space attribute vectors computed on the training set. Attribute vectors were balanced by boundary conditions and other attributes as described in the previous chapter. More specifically, the volume vector was balanced by area, and the area, solar gain, and morphology vectors were balanced by volume.

## 7.3. Environment

Following the same design vocabulary used for the dataset generation, modeling is realized in a grid-based environment. The grid size and resolution are identical to the ones defined for the dataset to allow a one-to-one correspondence when interfacing with the predictive model. Therefore, on the ground level, a 7X7 grid defines a site of 35mX35m. The maximum building plot occupies the central 5X5 area of the grid, corresponding to a rectangle with a 25m side, making it suitable for typical small to medium buildings with residential, office, or commercial use.

Figure 7.3 shows a screenshot of the prototype environment with the site grid in the center. Two panels frame the modeling area. On the left, the designer selects a system state, which enables modeling inputs of different types. On the right, a series of settings controlling the inference process are grouped in three stages: generate, tune, and explore.

In the top right corner of the modeling environment, a panel displays information about the generated geometry. In detail, it displays the total volume of the cuboids, the area percentage of the building plot that is covered by the geometry, and the mean incident solar radiation intensity during the winter and summer periods.

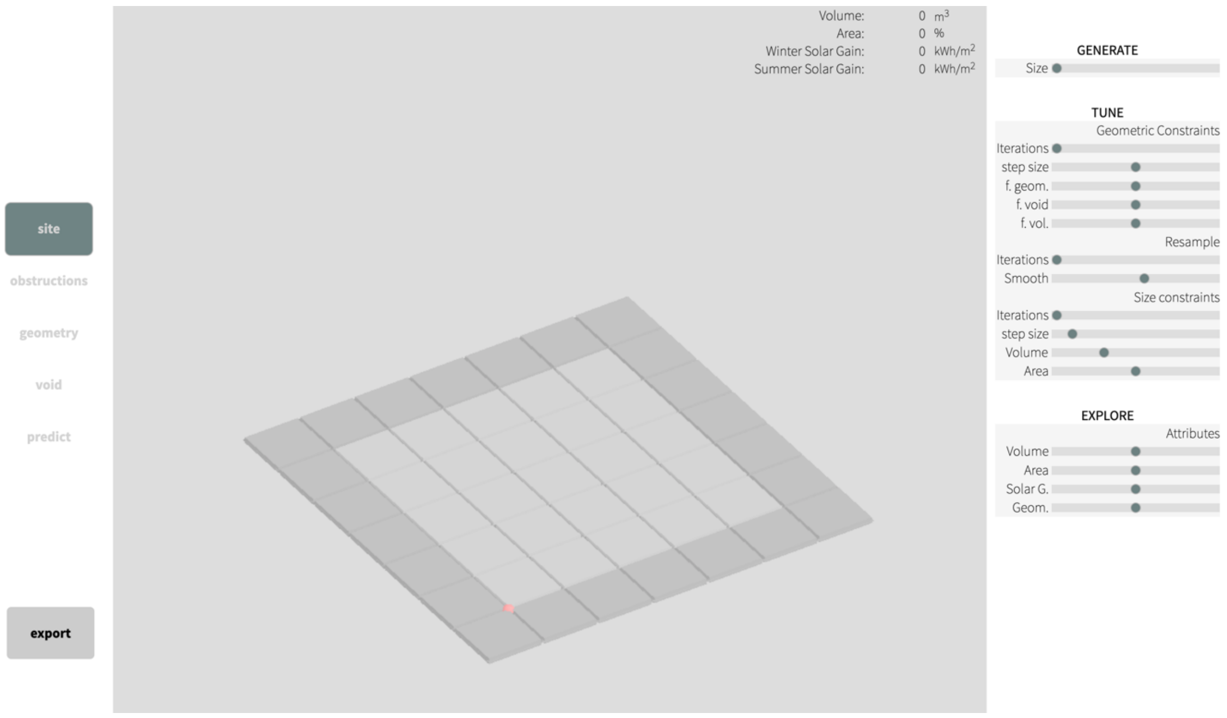
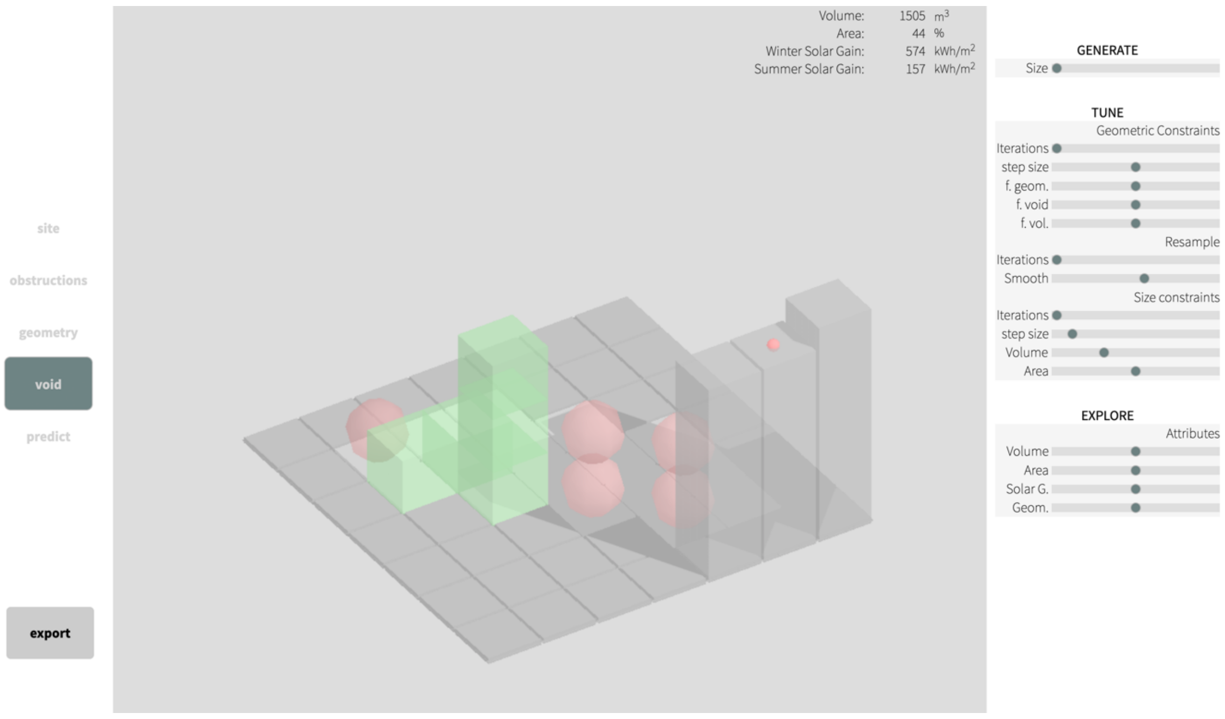

Volume: 0 m³
Area: 0 %
Winter Solar Gain: 0 kWh/m²
Summer Solar Gain: 0 kWh/m²

*Figure 7.3 Screenshot of the prototype.*

## 7.4. Workflow

### 7.4.1. Generate

The modeling state "site" allows the building plot to be created by adding or removing cells from the ground-level grid. The state "obstructions" allows the design of shading obstructions, representing the surroundings. Any cell from the ground-level grid not assigned as "site" can be extruded up to the maximum height of 25m to become a shading obstruction. Next, areas of desired built or unbuilt space can be defined by selecting the states "geometry" or "void". In these states, geometric primitives can be inserted into any cell of the 3-dimensional grid that falls within the building plot. Built cells are represented with cuboids, while unbuilt cells are represented with spheres. Figure 7.4 shows an example of a building site extending along the North-West–South-East axis with surrounding buildings abstracted as shading obstructions on the South. Desired built and unbuilt areas have been defined as green and red, respectively.



Volume: 1505 $m^3$
Area: 44 %
Winter Solar Gain: 574 $kWh/m^2$
Summer Solar Gain: 157 $kWh/m^2$

*Figure 7.4 User input example. The site has been specified. Desired built (green) and unbuilt (red) areas have been defined as well.*

Any change in the desired built-unbuilt geometries triggers a new geometry generation from the system, which is added to the modeling environment and rendered in white color. Various predictions can be acquired by repeatedly generating geometries in the “predict” state. Figure 7.5 shows four different geometries that were generated using the setting of Figure 7.4. In the left column, the generated geometries comply with all the geometric constraints, as they occupy all the desired built areas (green) and do not occupy any of the desired unbuilt areas (red). On the top right, some of the built constraints have not been fulfilled and on the bottom right, some of the unbuilt constraints have been violated.

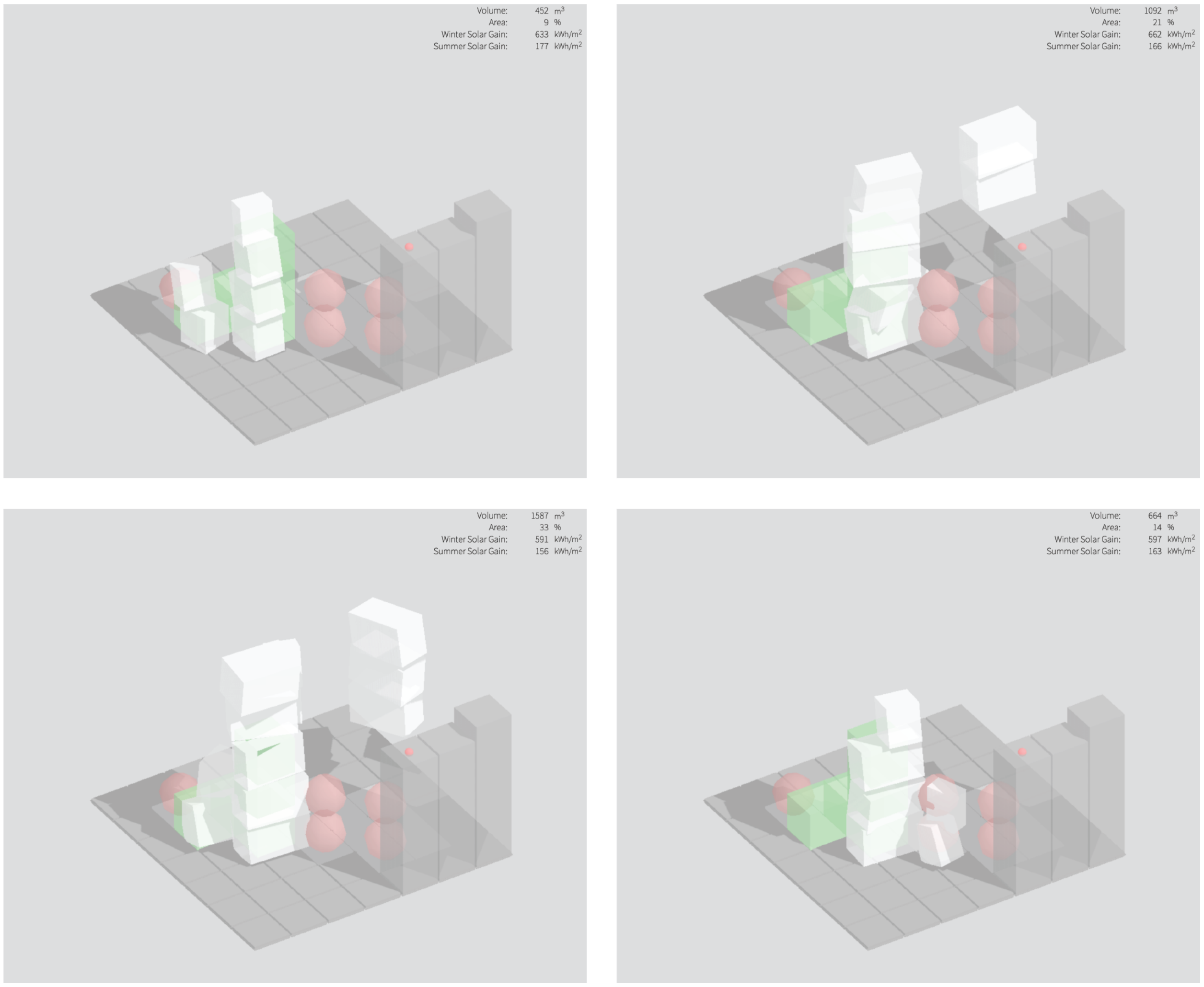
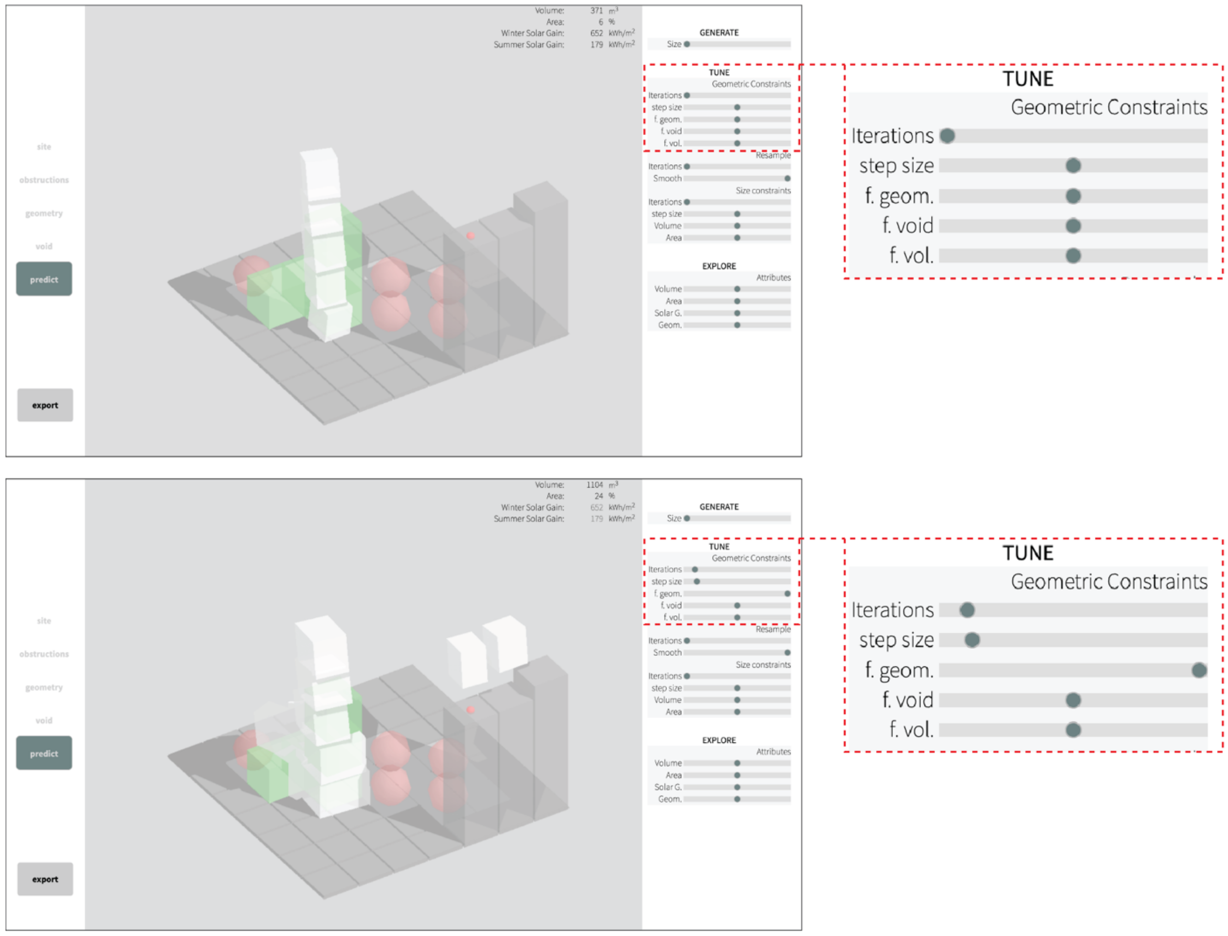

Volume: 452 m$^3$
Area: 9 %
Winter Solar Gain: 633 kWh/m$^2$
Summer Solar Gain: 177 kWh/m$^2$
Volume: 1092 m$^3$
Area: 21 %
Winter Solar Gain: 662 kWh/m$^2$
Summer Solar Gain: 166 kWh/m$^2$
Volume: 1587 m$^3$
Area: 33 %
Winter Solar Gain: 591 kWh/m$^2$
Summer Solar Gain: 156 kWh/m$^2$
Volume: 664 m$^3$
Area: 14 %
Winter Solar Gain: 597 kWh/m$^2$
Summer Solar Gain: 163 kWh/m$^2$

*Figure 7.5 Examples of geometry generation based on the specified site, built, and unbuilt constraints.*

### 7.4.2. Tune

The compliance of the generated geometry with the geometric constraints can be further enforced in the tuning phase. Figure 7.6 Top shows an example where the initially generated geometry only partially complies with the built constraints. In Figure 7.6 Bottom, the geometry is tuned to cover the whole area that the user has defined as built. The change is happening by navigating the latent space of the trained VAE. As a result, the updated geometry not only changes in response to the constraints, but additional mass has been placed on the highest level of the southeast corner. This can be interpreted as an effort to balance the adverse effect of the newly satisfied constraints on the solar gain performance. In specific, the cuboids in the northwest corner that were added because of the user-defined geometric constraints are shaded by the already existing cuboids (tower shape). In the winter-dominated climate of Boston, this reduces the total building's solar gain performance. In contrast, the cuboids that were generated in the southeast corner simultaneously with the constraint satisfaction are fully exposed from the south, which increases the building's solar gain performance.


Volume: 371 $m^3$
Area: 6 %
Winter Solar Gain: 652 $kWh/m^2$
Summer Solar Gain: 179 $kWh/m^2$

Volume: 1104 $m^3$
Area: 24 %
Winter Solar Gain: 652 $kWh/m^2$
Summer Solar Gain: 179 $kWh/m^2$

*Figure 7.6 Tuning the 'built' geometric constraints.*

Likewise, Figure 7.7 shows how a generated geometry was tuned to comply with the unbuilt constraints that were initially violated. In this example, the mass that was removed from the area marked as unbuilt was moved on top of the existing tower formation, maintaining the total building size approximately the same.

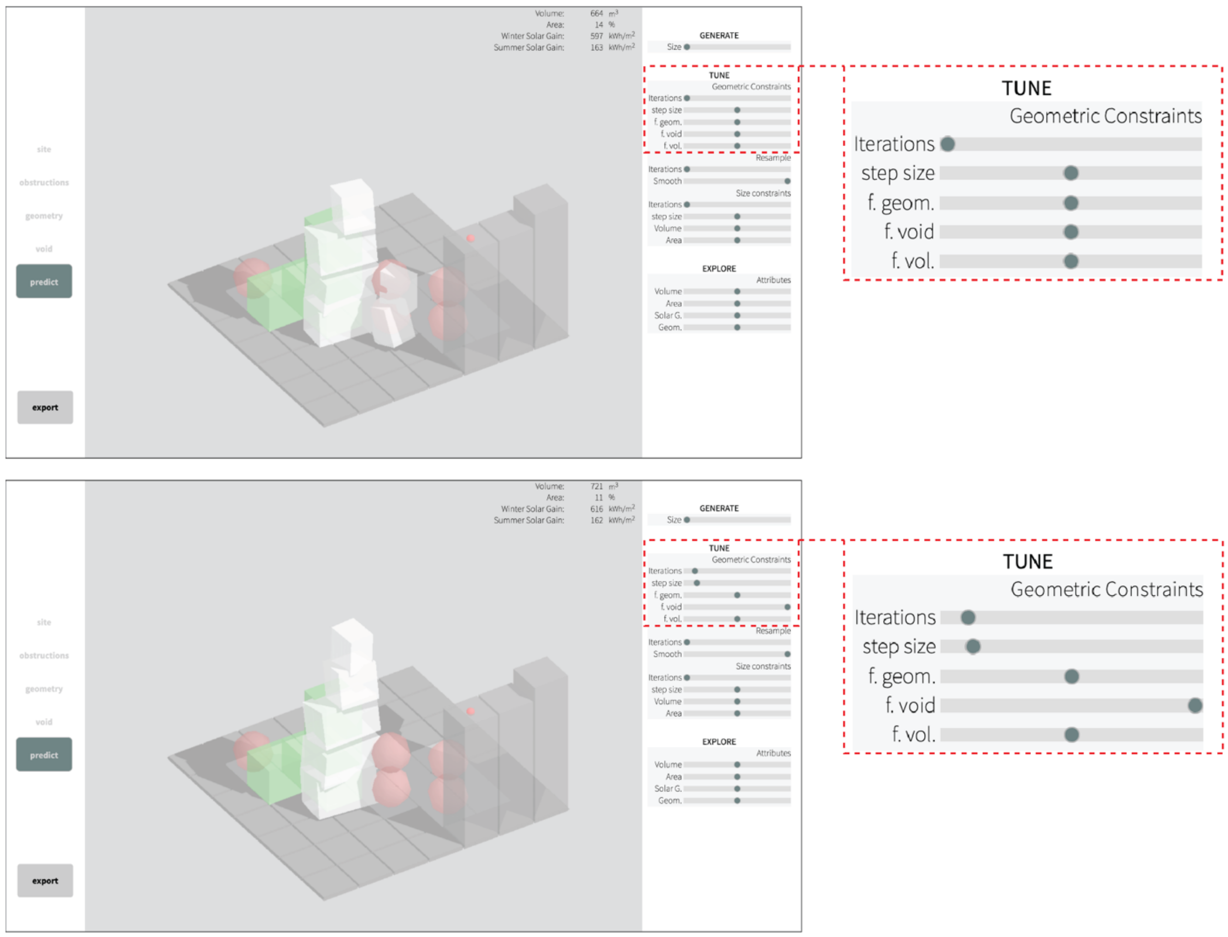
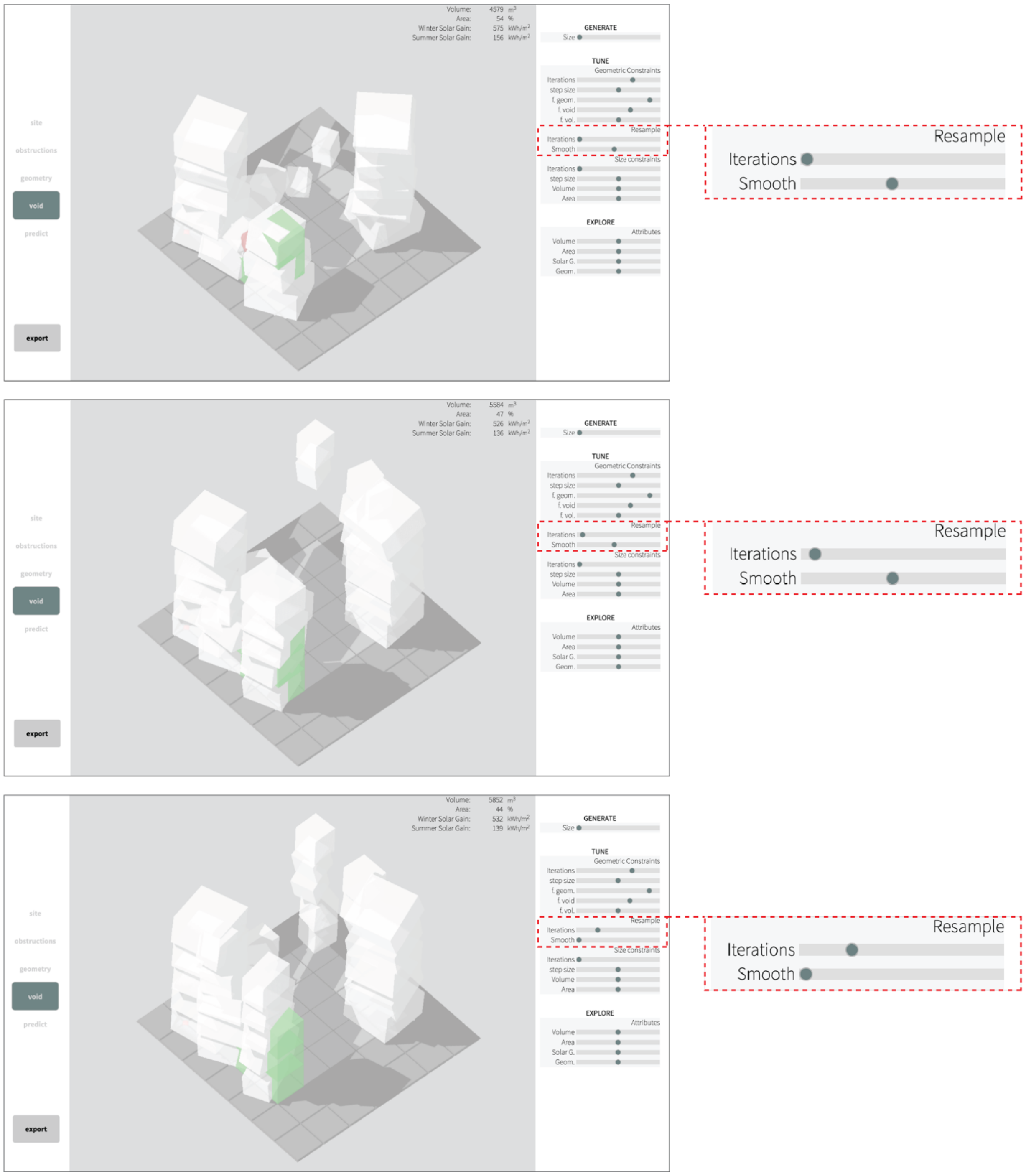


*Figure 7.7 Tuning the 'unbuilt' geometric constraints.*

Next, an optional geometry resampling step can be applied. This may be necessary in some edge cases when the enforcement of the geometric constraints results in a generated geometry that appears noisy. Two sliders control the number of iterative samples and the degree of change for each sampling step. Figure 7.8 shows such an example where, after resampling, the scattered volumes along the west border (Top) have been replaced by a tower formation in the northwest corner (Center and Bottom).


Volume: 4579 m³
Area: 54 %
Winter Solar Gain: 575 kWh/m²
Summer Solar Gain: 156 kWh/m²

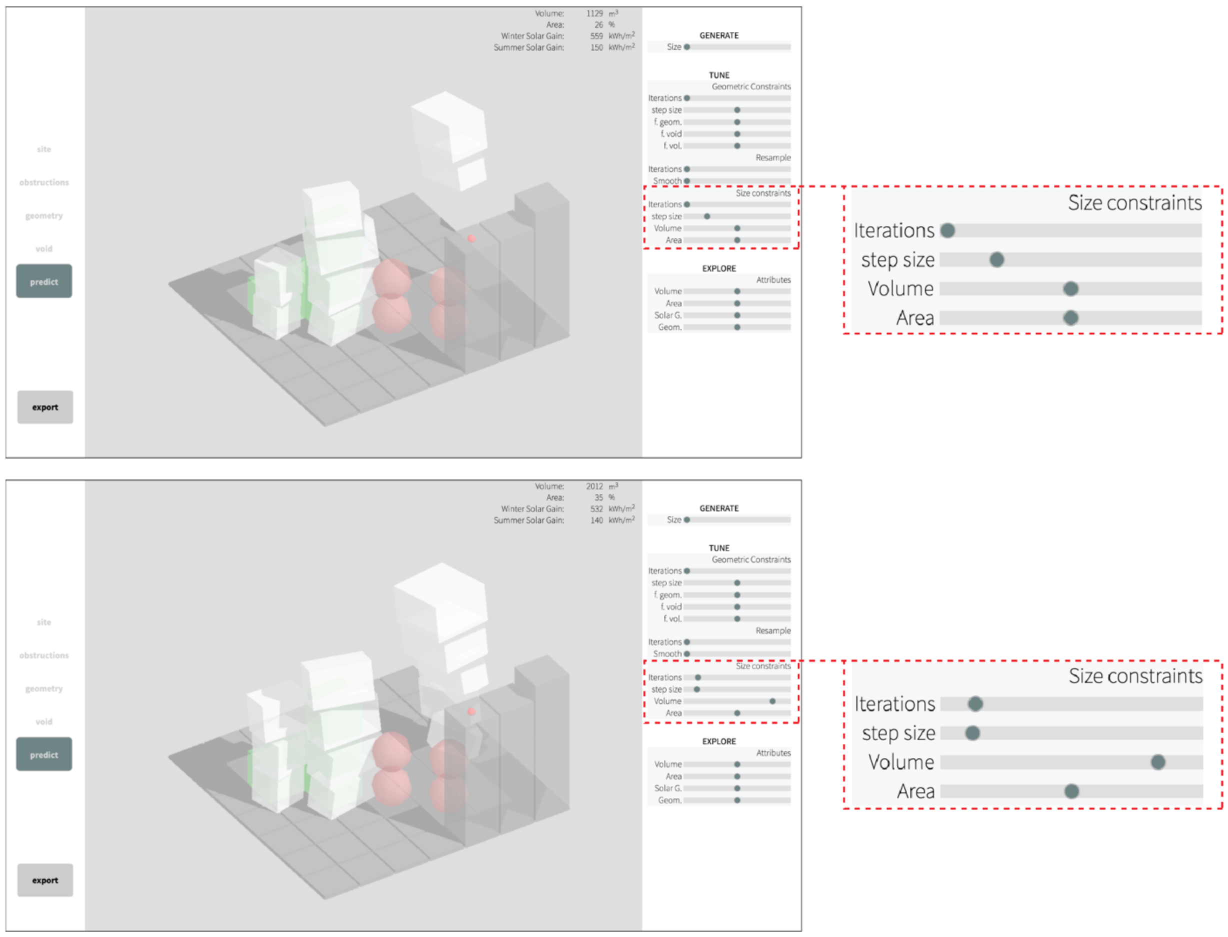
Volume: 5584 m³
Area: 47 %
Winter Solar Gain: 526 kWh/m²
Summer Solar Gain: 136 kWh/m²

Volume: 5852 m³
Area: 44 %
Winter Solar Gain: 532 kWh/m²
Summer Solar Gain: 139 kWh/m²

*Figure 7.8 Geometry resampling.*

At this stage, the geometry can be further tuned to comply with size-related constraints. The initial size selection during the generation phase acts as a rough, first estimate of the generated geometry’s size. Here, volume and area coverage can be individually controlled with a higher level of precision.

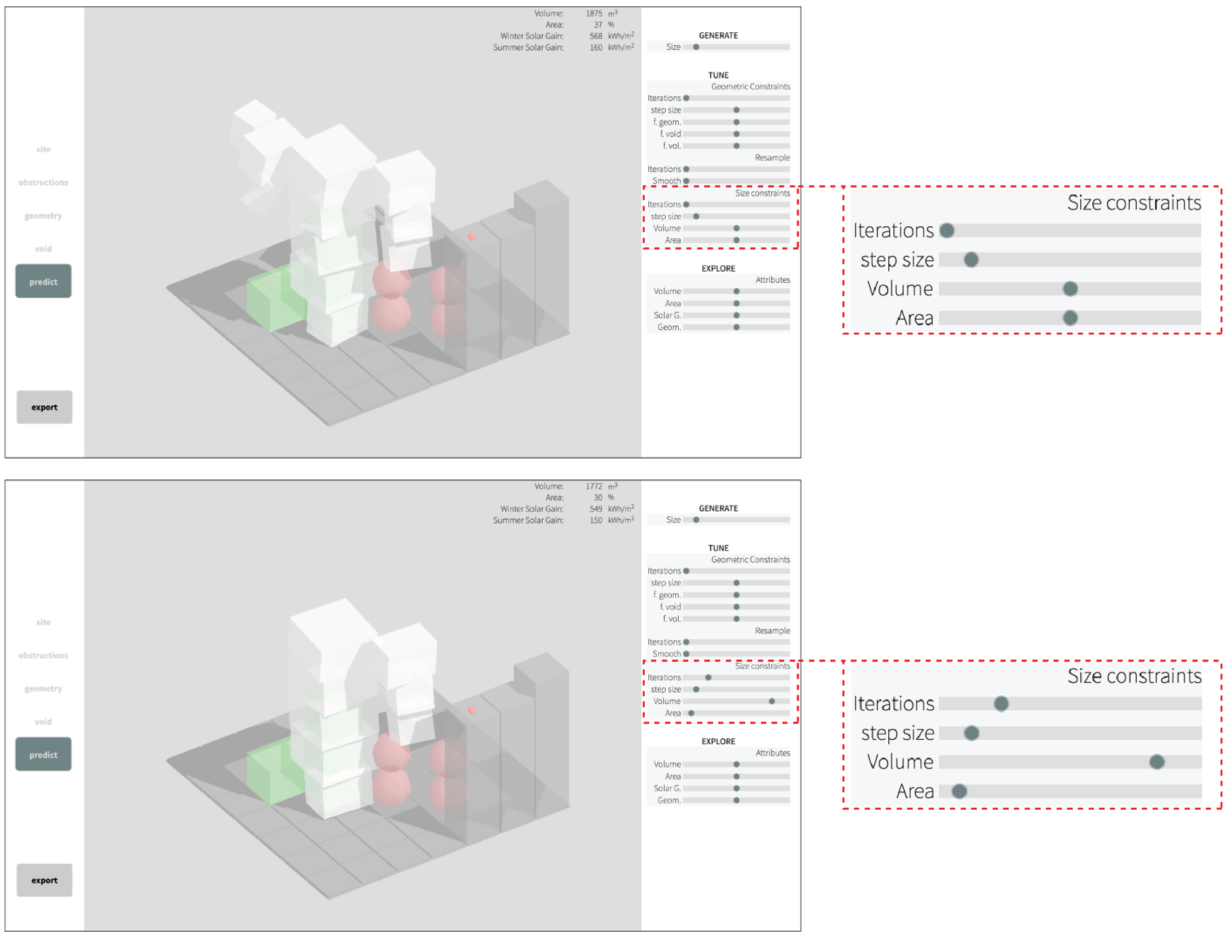
Figure 7.9 shows an example where the initially generated geometry’s volume was modified to match the desired building size. After controlling the volume modification slider, the system responded with a geometry where the existing cuboids have been scaled up, and additional cuboids have extended the geometry to more floor levels. Notably, an almost doubling of the volume (from 1129m$^3$ to 2012m$^3$) only led to a 9% change in the area coverage, which demonstrates how the volume and area modifiers have been decoupled. The solar gain performance has also been kept at similar levels (4% difference for winter and 6% difference for summer), showing that the underlying methods successfully allow geometry modifications without performance loss.


Volume: 1129 $m^3$
Area: 26 %
Winter Solar Gain: 559 $kWh/m^2$
Summer Solar Gain: 150 $kWh/m^2$

Volume: 2012 $m^3$
Area: 35 %
Winter Solar Gain: 532 $kWh/m^2$
Summer Solar Gain: 140 $kWh/m^2$

*Figure 7.9 Tuning the generated geometry's volume.*

Figure 7.10 shows a different example where the goal was to reduce the area coverage. The initial geometry's area coverage of 37% was reduced to 30%, while the volume remained at similar levels (from 1875m$^3$ to 1772m$^3$). In response to a change in the area modification slider, the system removed the northwest section of the geometry which contributed a significant amount to the area coverage but a smaller percentage to the volume. Then, a change to the volume slider caused the remaining cuboids to slightly expand, compensating for the volume loss from the first step.



Volume: 1875 m³
Area: 37 %
Winter Solar Gain: 568 kWh/m²
Summer Solar Gain: 160 kWh/m²

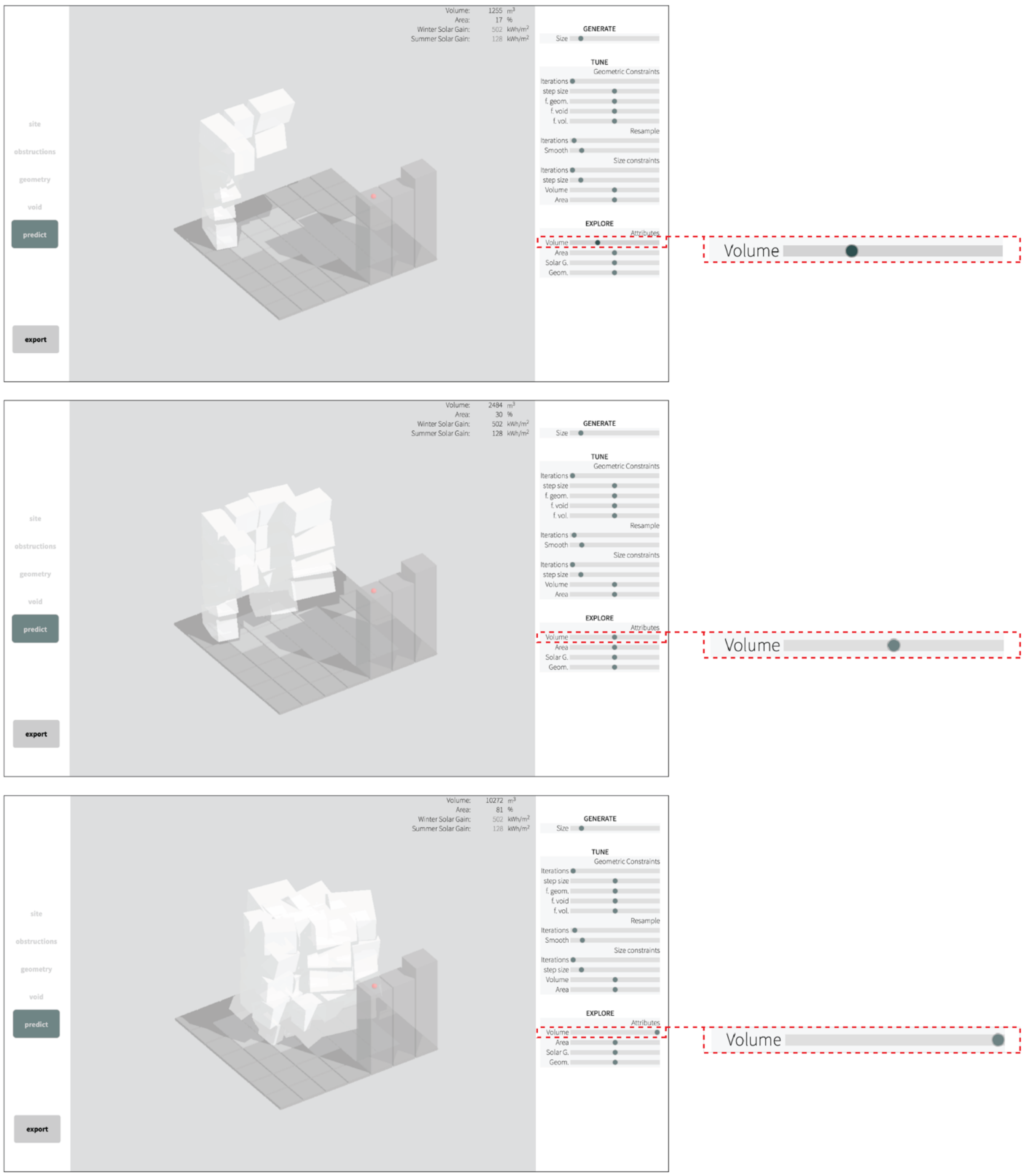
Volume: 1772 m³
Area: 30 %
Winter Solar Gain: 549 kWh/m²
Summer Solar Gain: 150 kWh/m²

*Figure 7.10 Tuning the generated geometry's area coverage.*

### 7.4.3. Explore

Last, to better understand the system's responses and the landscape of optimally performing geometries, the designer can explore what an optimal geometry would look like if various attributes were maximized or minimized. The set of attributes that can be explored consists of the volume, area coverage, solar gain performance, and the morphology-related metric defined in Equation ( 4.14 ). Figure 7.11 - Figure 7.14 show examples where a base geometry (Center), is modified to minimize (Top) or maximize (Bottom) each of the attributes. No constraints are taken into account in this process, other than the building site.

Figure 7.11 illustrates how minimizing the base geometry volume starts by removing mass from the lower levels of the building. First, this allows the area coverage to remain almost constant. Second, the higher levels of the building are less likely to be shaded, so keeping them as part of the building geometry ensures a good solar gain performance.

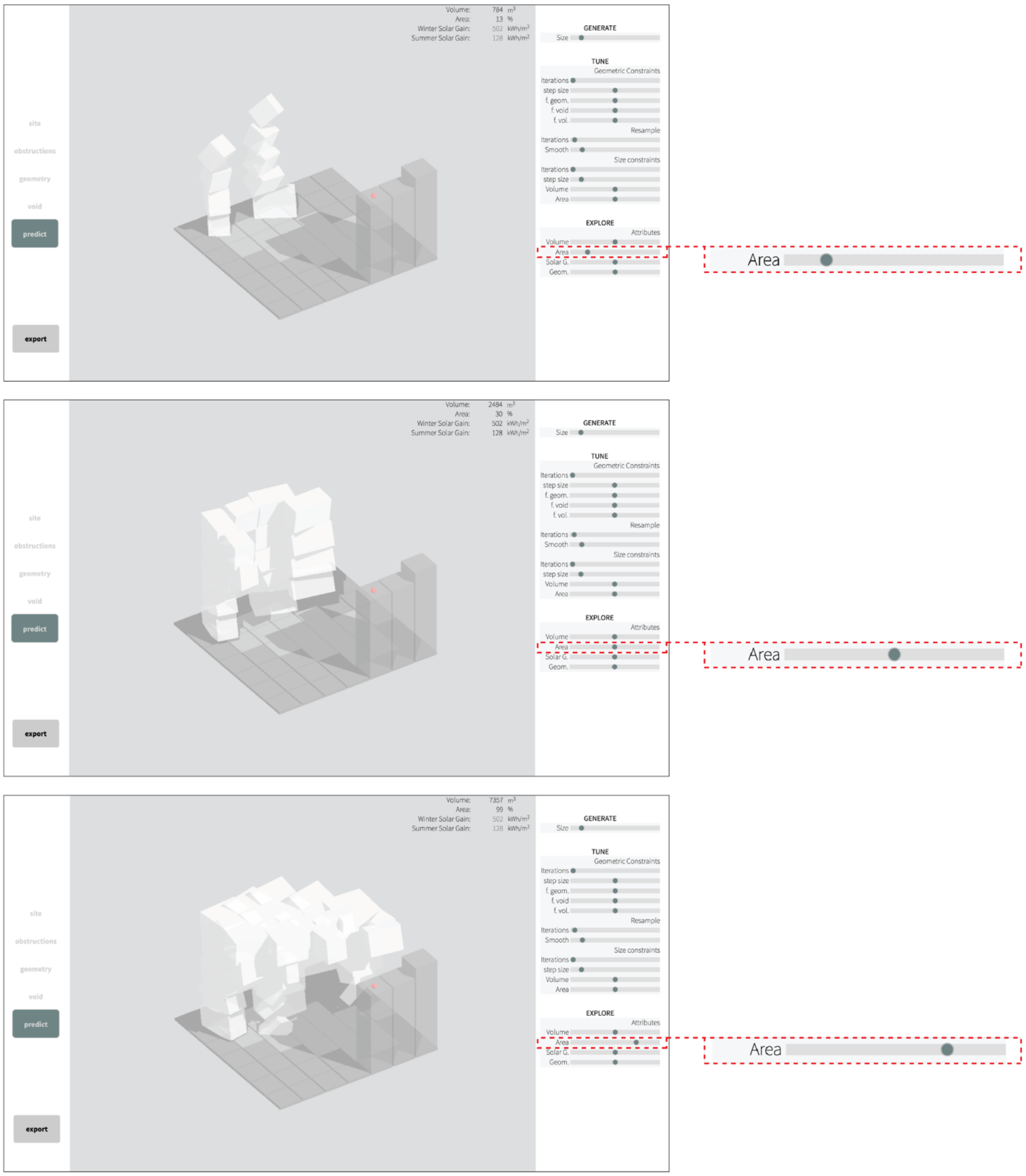

Volume: 1255 m³
Area: 17 %
Winter Solar Gain: 502 kWh/m²
Summer Solar Gain: 128 kWh/m²

Volume: 2484 m³
Area: 30 %
Winter Solar Gain: 502 kWh/m²
Summer Solar Gain: 128 kWh/m²

Volume: 10272 m³
Area: 81 %
Winter Solar Gain: 502 kWh/m²
Summer Solar Gain: 128 kWh/m²

*Figure 7.11 Volume exploration.*

Figure 7.12 shows the effect of modifying the area coverage, using the same base geometry. An increase in the area coverage results in the addition of mass at the maximum allowed height of the building plot. Similar to the previous example but inversely, the horizontal expansion affects the area coverage with small volume modifications, while the building geometry at the higher levels ensures good solar gain performance.



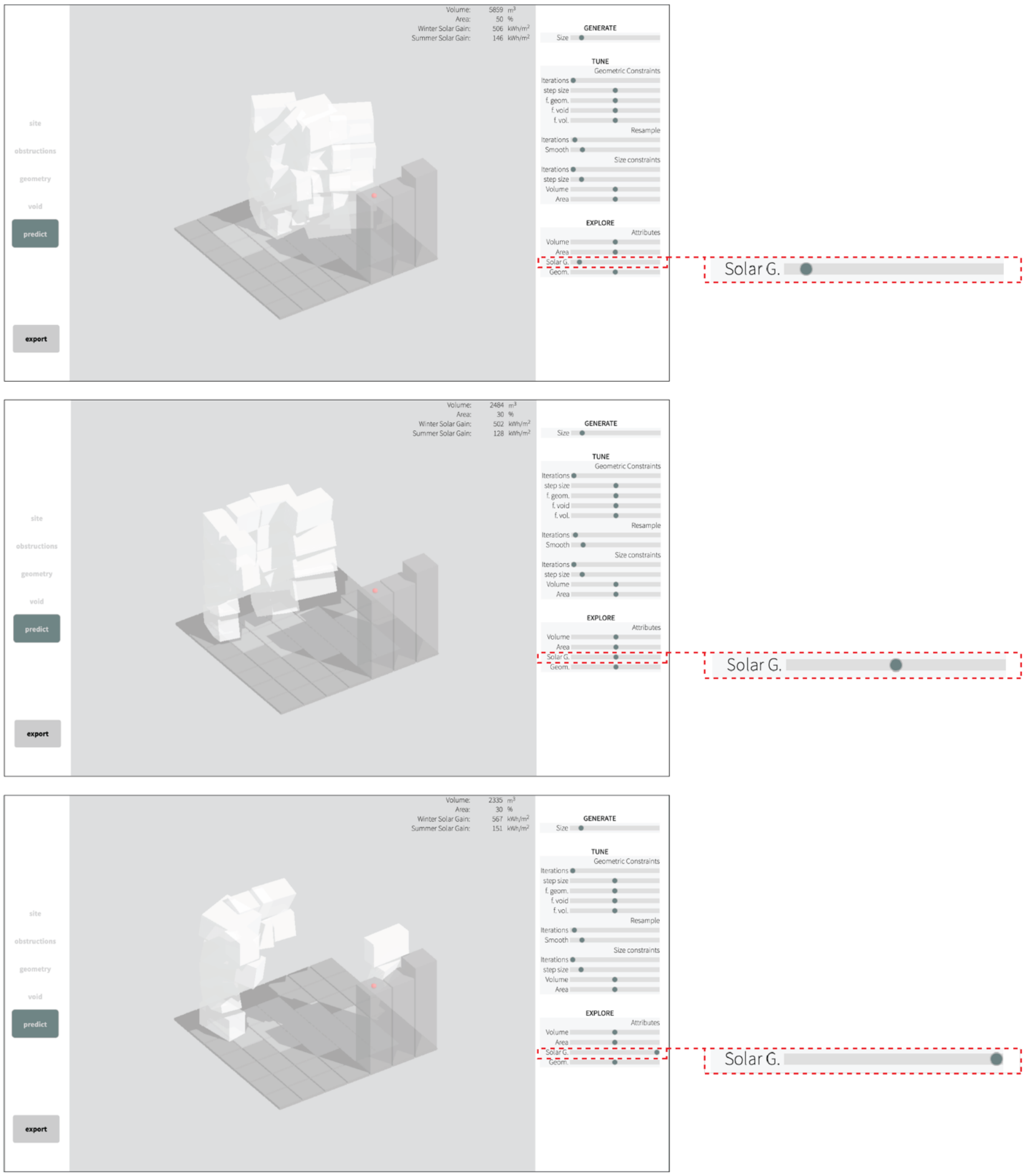
Volume: 784 m³
Area: 13 %
Winter Solar Gain: 502 kWh/m²
Summer Solar Gain: 128 kWh/m²

Volume: 2484 m³
Area: 30 %
Winter Solar Gain: 502 kWh/m²
Summer Solar Gain: 128 kWh/m²

Volume: 7357 m³
Area: 99 %
Winter Solar Gain: 502 kWh/m²
Summer Solar Gain: 128 kWh/m²

*Figure 7.12 Area exploration.*

Figure 7.13 shows how a better solar gain performance is achieved when the building mass is distributed in the building plot to avoid the shaded areas (Bottom), in contrast to more monolithic formations (Top).



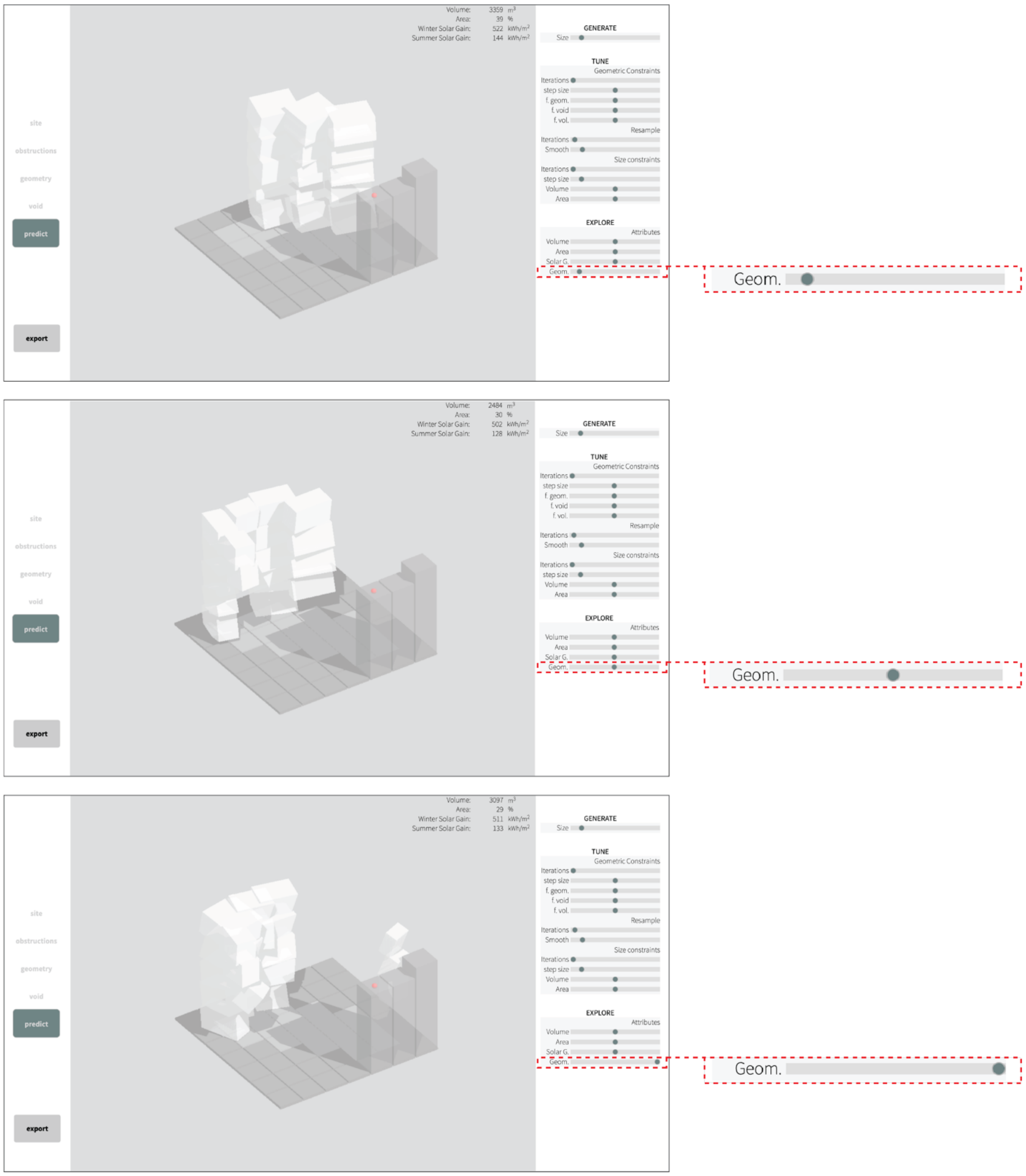
Volume: 5859 m³
Area: 50 %
Winter Solar Gain: 506 kWh/m²
Summer Solar Gain: 146 kWh/m²

Volume: 2484 m³
Area: 30 %
Winter Solar Gain: 502 kWh/m²
Summer Solar Gain: 128 kWh/m²

Volume: 2335 m³
Area: 30 %
Winter Solar Gain: 567 kWh/m²
Summer Solar Gain: 151 kWh/m²

*Figure 7.13 Solar gain exploration.*

Figure 7.14 shows a range of solutions with different morphologies that can be obtained for the current site. A geometry with a lower entropy – i.e., the evaluation of Equation ( 4.14 ) – is shown on the top, and a geometry with a higher entropy on the bottom.



```
         Volume:   3359  m³
           Area:     39  %
 Winter Solar Gain:   522  kWh/m²
Summer Solar Gain:   144  kWh/m²
```

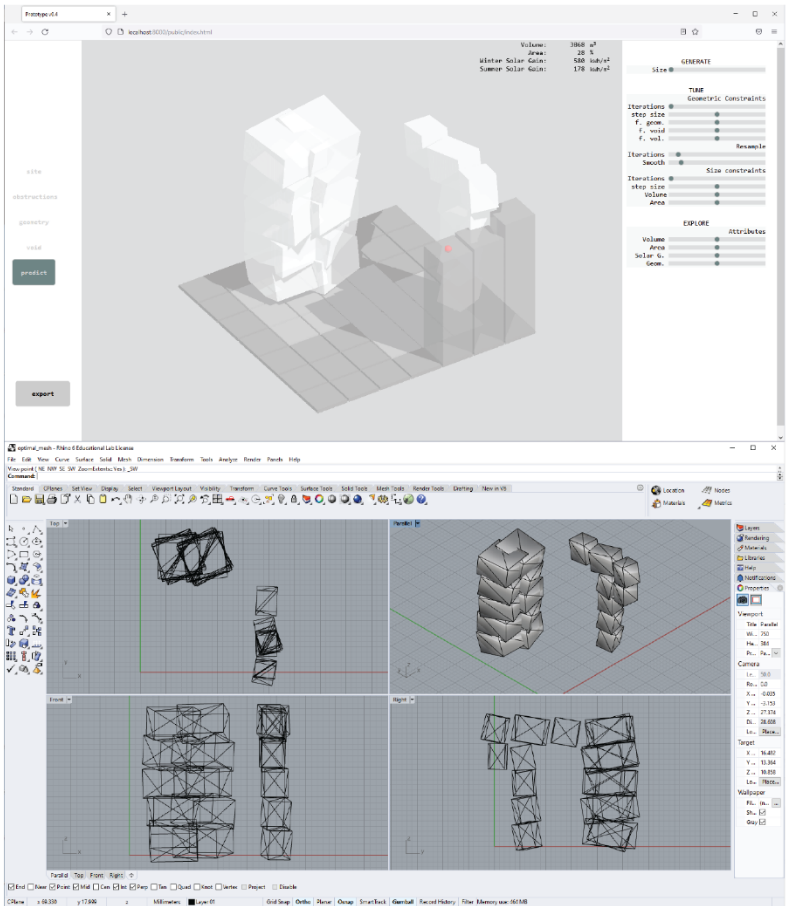
```
         Volume:   2484  m³
           Area:     30  %
 Winter Solar Gain:   502  kWh/m²
Summer Solar Gain:   128  kWh/m²
```

```
         Volume:   3097  m³
           Area:     29  %
 Winter Solar Gain:   511  kWh/m²
Summer Solar Gain:   133  kWh/m²
```

*Figure 7.14 Morphology exploration.*

### 7.4.4. Export

When the designer is satisfied with the generated solution's form and performance, the geometry can be exported for further development in any common 3d modeling software, as shown for example in Figure 7.15.



```
Volume:              3868  m³
Area:                  28  %
Winter Solar Gain:    580  kWh/m²
Summer Solar Gain:    178  kWh/m²
```

*Figure 7.15 Example of a generated geometry in the prototype (Top), and after it has been imported to Rhino (Bottom).*

## 7.5. Conclusion

This chapter demonstrates the applicability of the method that was developed. It illustrates how the trained model and the sampling techniques can be integrated within a design tool.

# 8. Conclusions

This thesis introduced a framework for responsive, intuitive, and comprehensible performance-driven design assistance for the early design phase. Using the case study of solar gain, it demonstrated how to generate synthetic data of optimal building geometries, and how to model the dataset distribution using a generative neural network. Then, it demonstrated how to use this generative model to solve the problems of optimal geometry generation and geometry modification driven by geometric or performance goals. Last, it tested the overall framework through a prototype that interactively generates optimally performing geometries in response to the designer's inputs of the building site, geometric form, and building constraints.

The framework has the capacity to meet the objective of responsiveness. The designer can interact with the generated solutions both by means of geometric modeling and by updating quantitative constraints. The interaction happens in real time, or with a few seconds of delay when iterative methods are used for a large number of repetitions.

In addition, the prototype illustrates how the framework enables low-effort, intuitive interaction. Given a performance goal, such as optimal solar gain, there is no required setup to start using the system. All interaction, including site setup and geometric preferences, is supported within the modeling environment. Quantitative constraints can be applied using simple controls. The results are generated in the form of ready-to-use geometries, which can be exported to other CAD software for further development.

Last, the real-time interaction and the goal and performance-driven geometry modifications have the capacity to allow the designer to explore the solution space in ways that promote an understanding of cause and effect between the site, geometry, constraints, and performance objectives.

In sum, to enable the suggested framework, this thesis presented the following contributions:

- A building geometry parametrization using orientable cuboids.
- Site and constraints sampling strategies for synthetic data generation using optimization.
- A performance-aware VAE for building geometries.
- A set of methods for constrained sampling of generative models.
- A set of methods for goal-driven navigation of a generative model's latent space.

## 8.1. High-level error terms

This work proposed the use of aggregate, building performance-related error terms in the loss function of a VAE. Such error functions are specific to the interpretation of the data as building geometries and can thus be seen as high-level formulations in contrast to representation agnostic vector metrics or probability functions.

These error terms were used in conjunction with a more typical, elementwise error, to better represent the difference between the input and reconstructed geometries. Proper weighting of the error terms is important to ensure accurate reconstructions. If the aggregate, performance-related terms are weighted too high, the morphology of the reconstructions may significantly deviate from the inputs. Inversely, if the performance terms are weighted too low, their effect is minimized, and the performance of the reconstructions may deviate from that of the inputs.

An interesting question that rises is how the introduction of an aggregate, morphology related error could contribute, and whether it would be able to substitute the elementwise error. The introduced framework works with synthetic data, whose geometric morphology has been constrained through base functions and further controlled by penalizing custom metrics. Therefore, a similar set of morphology metrics could also make its appearance in the reconstruction error of the VAE. Given the synthetic nature of the dataset and the stochasticity of the optimization process that generated it, a compliance to aggregate, morphology metrics may be a better measure of the reconstructed geometries' quality than an elementwise error.

## 8.2. Latent space structure

When random samples from the latent space were evaluated, a small percentage of them did not have optimal performance (6.1 Generative Model Samples). In addition, when the latent space was sampled with site constraints, using the pseudo-Gibbs and AE dynamics methods, the constraint error was minimized before the geometries converged, and before they achieved the highest performance (Figure 6.7 and Figure 6.10). Last, when latent space interpolation was performed between samples retrieved with only 1 or 5 pseudo-Gibbs iterations, the generated morphologies were not always consistent with the dataset (6.2 Comparison: Morphology). These observations highlight that the learned data distribution does not uniformly occupy all areas of the latent space. This is consistent with prior literature on latent spaces of generative models, which also observed the existence of 'dead zones' (White, 2016).

The site-constrained methods that involve iterative reconstructions (pseudo-Gibbs, AE dynamics) navigate away from such empty zones by converging to areas of higher probability. Experimental results regarding the generated geometries' performance and morphology suggest that gradient-based methods (latent space optimization with or without volume constraint) navigate away from such areas as well. However, as the constraints become stricter (e.g. higher factor for volume error) or their number increases (e.g. an additional constraint of a user-defined geometry), the appearance of noisy geometries suggests that the generated samples begin to deviate from the learned distribution. In the prototype, this issue was resolved by resampling the solution using iterative AE dynamics. The creation of a tighter-packed latent space would be another way to alleviate this problem.

## 8.3. Synthetic dataset

Regarding the goal-driven manipulation of geometries, both the attribute vectors and the gradient-following functions achieved a limited disentangling of the volume and the area coverage. In addition, efforts to control the height of the generated geometries were not successful. The limitation does not lie with the methods themselves, but rather with the training dataset. Dataset v2.6, which was used in this experiment, was generated without any kind of height regulation of the optimized geometries, other than the maximum height limitation dictated by the grid size. Therefore, the generated geometries populated the 3d grid starting from the highest levels, which typically maximizes the solar gain performance. The majority of the dataset geometries include tower-shaped forms, which almost uniformly occupy whole columns of the grid. As a result, there is minimal variability in the geometry heights of the dataset, and high interdependency between the cover area and volume.

The addition of a height constraint to the optimization process, a constructability objective, or a well-tuned height penalty, similar to the areaheight penalty used in dataset v2.01, would resolve both problems.

## 8.4. Layout and angle optimization

At different stages of this work, it was observed that the effect of the cuboids angle parameters was disproportionally small compared to the effect of the scaling parameters. To further improve the solar gain performance and morphologic consistency of both the layout and the geometric elements' orientation during the optimization process as well as during the generative model sampling, further work could consider a multi-stage approach. The scaling parameters, which dictate the layout, would be optimized first, with orientation angles following, as a separate process.

## 8.5. Generalizability

This work focused on generating geometries with optimal passive solar gain performance. However, the suggested framework can be applied with various types of building performance that are substantially affected by the building's geometric form. In addition, the system of transformable cuboids creates closed mesh geometries which would be compatible with a wide range of simulations (e.g. thermal simulations).

The framework can also support the use of multiple building performance objectives. This work reformulated the volume and area constraints as objectives of a multi-objective optimization problem and thus demonstrated how objective tradeoffs are handled during the optimization, as well as during the generative model sampling and the goal-driven geometry modification. Therefore, additional building performance objectives can easily be inserted and handled using the existing methodology.

This work introduced a general-use design tool for handling a wide range of building sites in Boston. On the other hand, the interactivity and responsiveness offered by the framework may also be leveraged for design assistance in one-off problems. In the case of a large, high-stakes project, the framework can be adjusted and applied on a single site with increased granularity of the design space (e.g., higher resolution of the cuboids grid), to help designers better navigate an exhaustive space of high-performing solutions.

## 8.6. Future Work

The definition of the building sites in this work has been limited regarding the site sizes and resolution, as well as surrounding buildings arrangement, due to time constraints. Future work may consider a wider range of sizes in a higher resolution, for both the plot shape and the building geometry grid. In addition, GIS data can be used to sample location-specific, real-world distributions of building sites.

Given a dataset that includes sites from multiple locations, the generative model may also be extended to condition the generated geometries on the location of the building site.

Additional variability in the dataset regarding multiple locations, building performance goals, and better building-height control may introduce challenges for the generative model. Prior research on latent space disentanglement can be used to better structure the learned latent space. Last, the addition of an adversarial objective may also be explored to better handle geometries of high resolution.

# 9. References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., & others. (2016). Tensorflow: A system for large-scale machine learning. *12th Symposium on Operating Systems Design and Implementation*, 265–283.

Aksöz, Z., & Preisinger, C. (2020). An Interactive Structural Optimization of Space Frame Structures Using Machine Learning. In C. Gengnagel, O. Baverel, J. Burry, M. Ramsgaard Thomsen, & S. Weinzierl (Eds.), *Impact: Design With All Senses* (pp. 18–31). Springer International Publishing. https://doi.org/10.1007/978-3-030-29829-6_2

Alammar, A., Jabi, W., & Lannon, S. (2021). Predicting Incident Solar Radiation on Building's Envelope Using Machine Learning. *Proceedings of the 12th Annual Symposium on Simulation for Architecture and Urban Design*. Simulation for Architecture and Urban Design.

Ampanavos, S., & Malkawi, A. (2021). Early-Phase Performance-Driven Design using Generative Models. *ArXiv:2107.08572 [Cs]*. http://arxiv.org/abs/2107.08572

Ampanavos, S., Nourbakhsh, M., & Cheng, C.-Y. (2021). Structural Design Recommendations in the Early Design Phase using Machine Learning. *ArXiv:2107.08567 [Cs]*. http://arxiv.org/abs/2107.08567

Ashour, Y. S. E.-D. (2015). *Optimizing Creatively in Multi-Objective Optimization* [M.E.Des., University of Calgary (Canada)]. http://search.proquest.com/docview/1759161420/abstract/E1E30D170B2E4A04PQ/1

Attia, S., Gratia, E., De Herde, A., & Hensen, J. L. M. (2012). Simulation-based decision support tool for early stages of zero-energy building design. *Energy and Buildings*, *49*, 2–15. https://doi.org/10.1016/j.enbuild.2012.01.028

Attia, S., Hamdy, M., O'Brien, W., & Carlucci, S. (2013). Assessing gaps and needs for integrating building performance optimization tools in net zero energy buildings design. *Energy and Buildings*, *60*, 110–124. https://doi.org/10.1016/j.enbuild.2013.01.016

Augenbroe, G. (1994). An Overview of the COMBINE project. *Proceedings of the First European Conference on Product and Process Modeling in the Building Industry (ECPPM'94)*, 12.

*Autodesk Revit*. (2000). [64-bit Windows]. Autodesk. www.autodesk.com/products/revit/overview

Bernhard, M., Kakooee, R., Bedarf, P., & Dillenburger, B. (2021). *Topology Optimization with Generative Adversarial Networks*. 22.

Blank, J., & Deb, K. (2020). Pymoo: Multi-Objective Optimization in Python. *IEEE Access*, *8*, 89497–89509.

Blender Foundation. (2021). *Blender*. Blender Foundation. https://git.blender.org/gitweb/gitweb.cgi/blender.git

Bradner, E., Iorio, F., & Davis, M. (2014). Parameters tell the design story: Ideation and abstraction in design optimization. *2014 Proceedings of the Symposium on Simulation for Architecture and Urban Design*, 26.

Bre, F., Roman, N., & Fachinotti, V. D. (2020). An efficient metamodel-based method to carry out multi-objective building performance optimizations. *Energy and Buildings*, *206*, 109576. https://doi.org/10.1016/j.enbuild.2019.109576

Brock, A., Donahue, J., & Simonyan, K. (2019). Large Scale GAN Training for High Fidelity Natural Image Synthesis. *ArXiv:1809.11096 [Cs, Stat]*. http://arxiv.org/abs/1809.11096

Brown, N. C. (2020). Design performance and designer preference in an interactive, data-driven conceptual building design scenario. *Design Studies*. https://doi.org/10.1016/j.destud.2020.01.001

Brown, N. C., & Mueller, C. T. (2017). Automated performance-based design space simplification for parametric structural design. *Proceedings of IASS Annual Symposia*, 10.

Brown, N. C., & Mueller, C. T. (2019). Design variable analysis and generation for performance-based parametric modeling in architecture. *International Journal of Architectural Computing*, *17*(1), 36–52. https://doi.org/10.1177/1478077118799491

Caldas, L. (2001). *An evolution-based generative design system: Using adaptation to shape architectural form* [Thesis, Massachusetts Institute of Technology]. http://dspace.mit.edu/handle/1721.1/8188

Caldas, L. G. ; S. (2012). Generation of Energy-Efficient Patio Houses with GENE_ARCH: Combining an Evolutionary Generative Design System with a Shape Grammar. *Digital Physicality - Proceedings of the 30th ECAADe Conference*, 459–470. http://papers.cumincad.org/cgi-bin/works/paper/ecaade2012_267

Carrara, G., Kalay, Y. E., & Novembri, G. (1991). A Computational Framework for Supporting Creative Architectural Design. In Y. E. Kalay, *Evaluating and predicting design performance*. Wiley.

*CGAL*. (2021). CGAL Open Source Project. www.cgal.org

Chaillou, S. (2020). ArchiGAN: Artificial Intelligence x Architecture. In P. F. Yuan, M. Xie, N. Leach, J. Yao, & X. Wang (Eds.), *Architectural Intelligence: Selected Papers from the 1st International Conference on Computational Design and Robotic Fabrication (CDRF 2019)* (pp. 117–127). Springer. https://doi.org/10.1007/978-981-15-6568-7_8

Chang, K.-H., & Cheng, C.-Y. (2020). Learning to Simulate and Design for Structural Engineering. *International Conference on Machine Learning*, 1426–1436. http://proceedings.mlr.press/v119/chang20a.html

Chang, K.-H., Cheng, C.-Y., Luo, J., Murata, S., Nourbakhsh, M., & Tsuji, Y. (2021). Building-GAN: Graph-Conditioned Architectural Volumetric Design Generation. *ArXiv:2104.13316 [Cs]*. http://arxiv.org/abs/2104.13316

Chronis, A., Tsigkari, M., Giouvanos, E., Aish, F., & Zaki, A. A. (2012). Performance Driven Design and Simulation Interfaces: A Multi-objective Parametric Optimization Process. *Proceedings of the 2012*

*Symposium on Simulation for Architecture and Urban Design*, 14:1-14:8. http://dl.acm.org/citation.cfm?id=2339453.2339467

Conti, Z. X., & Kaijima, S. (2018). Enabling Inference in Performance-Driven Design Exploration. In K. De Rycke, C. Gengnagel, O. Baverel, J. Burry, C. T. Mueller, M. M. Nguyen, P. Rahm, & M. R. Thomsen (Eds.), *Humanizing Digital Reality: Design Modelling Symposium Paris 2017* (pp. 177–188). Springer Singapore. https://doi.org/10.1007/978-981-10-6611-5_16

*Convolutional Variational Autoencoder | TensorFlow Core*. (n.d.). TensorFlow. Retrieved February 27, 2021, from https://www.tensorflow.org/tutorials/generative/cvae

Costa, A., & Nannicini, G. (2018). RBFOpt: An open-source library for black-box optimization with costly function evaluations. *Mathematical Programming Computation*, *10*(4), 597–629. https://doi.org/10.1007/s12532-018-0144-7

Creswell, A., & Bharath, A. A. (2019). Inverting the Generator of a Generative Adversarial Network. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(7), 1967–1974. https://doi.org/10.1109/TNNLS.2018.2875194

Çubukçuoğlu, C., Ekici, B., Tasgetiren, M., & Sariyildiz, S. (2019). OPTIMUS: Self-Adaptive Differential Evolution with Ensemble of Mutation Strategies for Grasshopper Algorithmic Modeling. *Algorithms*, *12*, 141. https://doi.org/10.3390/a12070141

Dabbeeru, M. M., & Mukerjee, A. (2008). Discovering Implicit Constraints in Design. In J. S. Gero & A. K. Goel (Eds.), *Design Computing and Cognition ’08* (pp. 201–220). Springer Netherlands.

Danhaive, R., & Mueller, C. T. (2021). Design subspace learning: Structural design space exploration using performance-conditioned generative modeling. *Automation in Construction*, *127*, 103664. https://doi.org/10.1016/j.autcon.2021.103664

Davis, D. (2013). *Modelled on software engineering: Flexible parametric models in the practice of architecture* [Ph.D., RMIT]. https://researchbank.rmit.edu.au/view/rmit:161769

Dawson-Haggerty et al. (2021). *Trimesh* (3.9.15). https://trimsh.org/

de Miguel, J. (2019). Deep Form Finding—Using Variational Autoencoders for deep form finding of structural typologies. *Sousa, JP, Xavier, JP and Castro Henriques, G (Eds.), Architecture in the Age of the 4th Industrial Revolution - Proceedings of the 37th ECAADe and 23rd SIGraDi Conference - Volume 1, University of Porto, Porto, Portugal, 11-13 September 2019, Pp. 71-80*. http://papers.cumincad.org/cgi-bin/works/BrowseTree=series=AZ/Show?ecaadesigradi2019_514

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182–197. https://doi.org/10.1109/4235.996017

Del Campo, M. M. (2019). Imaginary Plans. *Proceedings of the 39th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, 412–418. http://papers.cumincad.org/cgi-bin/works/paper/acadia19_412

*Design Exploration User's Guide* (User Manual 15.0; p. 296). (2013). ANSYS, Inc.

Dorst, K., & Cross, N. (2001). Creativity in the design process: Co-evolution of problem–solution. *Design Studies*, *22*(5), 425–437. https://doi.org/10.1016/S0142-694X(01)00009-6

*Dynamo*. (2022). [C#]. Autodesk. https://github.com/DynamoDS/Dynamo (Original work published 2012)

Evins, R. (2013). A review of computational optimisation methods applied to sustainable building design. *Renewable and Sustainable Energy Reviews*, *22*, 230–245. https://doi.org/10.1016/j.rser.2013.02.004

Fedorova, S. (2021). GANs for Urban Design. *ArXiv:2105.01727 [Cs]*. http://arxiv.org/abs/2105.01727

Gadelha, M., Gori, G., Ceylan, D., Mech, R., Carr, N., Boubekeur, T., Wang, R., & Maji, S. (2020). Learning Generative Models of Shape Handles. *ArXiv:2004.03028 [Cs]*. http://arxiv.org/abs/2004.03028

Gagnon, R., Gosselin, L., & Decker, S. (2018). Sensitivity analysis of energy performance and thermal comfort throughout building design process. *Energy and Buildings*, *164*, 278–294. https://doi.org/10.1016/j.enbuild.2017.12.066

Geman, S., & Geman, D. (1984). Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-6*(6), 721–741. https://doi.org/10.1109/TPAMI.1984.4767596

Gertraud Breitkopf, Micheal Alianza, Norman Koonce, James Dinegar, Richard Allen, Patrick Natale, Harry Mashburn, David Lukens, David Harris, Ric Jackson, Martin Fischer, Patrick MacLeamy, Ashok Raiji, Carl Galioto, Raymond Messer, Joseph Burns, Geoge Pontikes, William Skinner, Gregory Bentley, … Norbert Young. (2004). *Collaboration, Integrated Information and the Project Lifecycle in Building Design, Construction and Operation* (White Paper WP-1202). Construction Users Roundtable.

Ghojogh, B., Karray, F., & Crowley, M. (2020). Theoretical Insights into the Use of Structural Similarity Index In Generative Models and Inferential Autoencoders. *ArXiv:2004.01864 [Cs, Eess, Stat]*, *12132*, 112–117. https://doi.org/10.1007/978-3-030-50516-5_10

Goodfellow, I. (2017). NIPS 2016 Tutorial: Generative Adversarial Networks. *ArXiv:1701.00160 [Cs]*. http://arxiv.org/abs/1701.00160

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 27* (pp. 2672–2680). Curran Associates, Inc. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

Gossard, D., Lartigue, B., & Thellier, F. (2013). Multi-objective optimization of a building envelope for thermal performance using genetic algorithms and artificial neural network. *Energy and Buildings*, *67*, 253–260. https://doi.org/10.1016/j.enbuild.2013.08.026

Han, J. M., Choi, E. S., & Malkawi, A. (2021). CoolVox: Advanced 3D convolutional neural network models for predicting solar radiation on building facades. *Building Simulation*. https://doi.org/10.1007/s12273-021-0837-0

Harding, J. (2016). Dimensionality reduction for parametric design exploration. *Advances in Architectural Geometry 2016*, 274–287. https://doi.org/10.3218/3778-4_19

Harding, J. E., & Shepherd, P. (2017). Meta-Parametric Design. *Design Studies*, *52*, 73–95. https://doi.org/10.1016/j.destud.2016.09.005

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., & Lerchner, A. (2016). *beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework*. https://openreview.net/forum?id=Sy2fzU9gl

Holzer, D., Hough, R., & Burry, M. (2007). Parametric Design and Structural Optimisation for Early Design Exploration. *International Journal of Architectural Computing*, *5*(4), 625–643. https://doi.org/10.1260/147807707783600780

Hopfe, C., Hensen, J., & Plokker, W. (2007). UNCERTAINTY AND SENSITIVITY ANALYSIS FOR DETAILED DESIGN SUPPORT. *Building Simulation*, 6.

Huang, W., & Zheng, H. (2018). Architectural Drawings Recognition and Generation through Machine Learning. *Proceedings of the 38th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, 10.

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

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). *Image-To-Image Translation With Conditional Adversarial Networks*. 1125–1134. https://openaccess.thecvf.com/content_cvpr_2017/html/Isola_Image-To-Image_Translation_With_CVPR_2017_paper.html

Jones, N. L., & Reinhart, C. F. (2014). Physically Based Global Illumination Calculation Using Graphics Hardware. *Proceedings of ESim 2014: The Canadian Conference on Building Simulation*, 474–487.

Kämpf, J. H., & Robinson, D. (2010). Optimisation of building form for solar energy utilisation using constrained evolutionary algorithms. *Energy and Buildings*, *42*(6), 807–814. https://doi.org/10.1016/j.enbuild.2009.11.019

Kancharla, P., & Channappayya, S. S. (2018). Improving the Visual Quality of Generative Adversarial Network (GAN)-Generated Images Using the Multi-Scale Structural Similarity Index. *2018 25th IEEE International Conference on Image Processing (ICIP)*, 3908–3912. https://doi.org/10.1109/ICIP.2018.8451296

Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2018). Progressive Growing of GANs for Improved Quality, Stability, and Variation. *ArXiv:1710.10196 [Cs, Stat]*. http://arxiv.org/abs/1710.10196

Kheiri, F. (2018). A review on optimization methods applied in energy-efficient building geometry and envelope design. *Renewable and Sustainable Energy Reviews*, *92*, 897–920. https://doi.org/10.1016/j.rser.2018.04.080

Kingma, D. P., & Ba, J. (2017). Adam: A Method for Stochastic Optimization. *ArXiv:1412.6980 [Cs]*. http://arxiv.org/abs/1412.6980

Kingma, D. P., & Welling, M. (2014). Auto-Encoding Variational Bayes. *ArXiv:1312.6114 [Cs, Stat]*. http://arxiv.org/abs/1312.6114

Kintel, M., & Wolf, C. (2021). *OpenSCAD*. http://www.openscad.org/

Kleineberg, M., Fey, M., & Weichert, F. (2020). Adversarial Generation of Continuous Implicit Shape Representations. *ArXiv:2002.00349 [Cs]*. http://arxiv.org/abs/2002.00349

Konis, K., Gamas, A., & Kensek, K. (2016). Passive performance and building form: An optimization framework for early-stage design support. *Solar Energy*, *125*, 161–179. https://doi.org/10.1016/j.solener.2015.12.020

Kumar, R., Aggarwal, R. K., & Sharma, J. D. (2013). Energy analysis of a building using artificial neural network: A review. *Energy and Buildings*, *65*, 352–358. https://doi.org/10.1016/j.enbuild.2013.06.007

Lam, J. C., & Hui, S. C. M. (1996). Sensitivity analysis of energy performance of office buildings. *Building and Environment*, *31*(1), 27–39. https://doi.org/10.1016/0360-1323(95)00031-3

Larsen, A. B. L., Sønderby, S. K., Larochelle, H., & Winther, O. (2016). Autoencoding beyond pixels using a learned similarity metric. *ArXiv:1512.09300 [Cs, Stat]*. http://arxiv.org/abs/1512.09300

Ligget, R., & Milne, M. (2018). *Climate Consultant* (6.0). UCLA. http://www.energy-design-tools.aud.ucla.edu/climate-consultant/request-climate-consultant.php

Lin, S.-H. E., & Gerber, D. J. (2014). Designing-in performance: A framework for evolutionary energy performance feedback in early stage design. *Automation in Construction*, *38*, 59–73. https://doi.org/10.1016/j.autcon.2013.10.007

Lin, S.-H., & Gerber, D. J. (2014). Evolutionary energy performance feedback for design: Multidisciplinary design optimization and performance boundaries for design decision support. *Energy and Buildings*, *84*, 426–441. https://doi.org/10.1016/j.enbuild.2014.08.034

Liu, H., Longtai Liao, & Akshay Scrivastava. (2019). An Anonymous Composition. *Proceedings of the 39th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, 404–411. http://papers.cumincad.org/cgi-bin/works/paper/acadia19_404

Liu, Z., Luo, P., Wang, X., & Tang, X. (2015, December). Deep Learning Face Attributes in the Wild. *Proceedings of International Conference on Computer Vision (ICCV)*.

Loaiza-Ganem, G., & Cunningham, J. P. (2019). The continuous Bernoulli: Fixing a pervasive error in variational autoencoders. *ArXiv:1907.06845 [Cs, Stat]*. http://arxiv.org/abs/1907.06845

Lorenz, C.-L., Packianather, M., Spaeth, A. B., & Souza, C. B. D. (2018). Artificial Neural Network-Based Modelling for Daylight Evaluations. *2018 Proceedings of the Symposium on Simulation for Architecture & Urban Design*, 8.

Lun, Z., Gadelha, M., Kalogerakis, E., Maji, S., & Wang, R. (2017). 3D Shape Reconstruction from Sketches via Multi-view Convolutional Networks. *ArXiv:1707.06375 [Cs]*. http://arxiv.org/abs/1707.06375

Mahdavi, A. (1996). On the Structure and Elements of SEMPER. *Design Computation: Collaboration, Reasoning, Pedagogy [ACADIA Conference Proceedings / ISBN 1-880250-05-5] Tucson (Arizona / USA) October 31 - November 2, 1996, Pp. 71-84*. http://papers.cumincad.org/cgi-bin/works/Show?e02e

Mattei, P.-A., & Frellsen, J. (2018). Leveraging the exact likelihood of deep latent variable models. *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 3859–3870.

McCoy, J. T., Kroon, S., & Auret, L. (2018). Variational Autoencoders for Missing Data Imputation with Application to a Simulated Milling Circuit. *IFAC-PapersOnLine*, *51*(21), 141–146. https://doi.org/10.1016/j.ifacol.2018.09.406

Michalatos, P., & Kaijima, S. (2014). Eigenshells: Structural patterns on modal forms. In *Shell Structures for Architecture*. Routledge.

*ModeFRONTIER*. (2017). Esteco S.p.A. www.esteco.com

Mohammad, A. B. (2019). Hybrid Elevations using GAN Networks. *Proceedings of the 39th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*. ACADIA 19:UBIQUITY AND AUTONOMY. http://papers.cumincad.org/cgi-bin/works/paper/acadia19_370

Mokhtar, S., Sojka, A., & Davila, C. C. (2020). Conditional generative adversarial networks for pedestrian wind flow approximation. *Proceedings of the 11th Annual Symposium on Simulation for Architecture and Urban Design*, 1–8.

Morbitzer, C. A. (2003). *Towards the Integration of Simulation into the Building Design Process* [Doctoral Thesis, Uni versity of Strathclyde]. http://www.esru.strath.ac.uk/Documents/PhD/morbitzer_thesis.pdf

Mueller, C. T. (2013). *StructureFIT*. http://digitalstructures.mit.edu/page/tools#structurefit

Mueller, C. T., & Ochsendorf, J. A. (2015). Combining structural performance and designer preferences in evolutionary design space exploration. *Automation in Construction*, *52*, 70–82. https://doi.org/10.1016/j.autcon.2015.02.011

Musil, J., & Wilkinson, S. (2016). Automated prediction of preference level by artificial neural network for simple geometry. *2016 Proceedings of the Symposium on Simulation for Architecture and Urban Design*, 4.

Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, L., Wang, R., Zhao, D., & Benjamin, D. (2017). Project Discover: An Application of Generative Design for Architectural Space Planning. *2017 Proceedings of the Symposium on Simulation for Architecture & Urban Design*, 8.

Nauata, N., Chang, K.-H., Cheng, C.-Y., Mori, G., & Furukawa, Y. (2020). House-GAN: Relational Generative Adversarial Networks for Graph-constrained House Layout Generation. *ArXiv:2003.06988 [Cs]*. http://arxiv.org/abs/2003.06988

Nguyen, A.-T., Reiter, S., & Rigo, P. (2014). A review on simulation-based optimization methods applied to building performance analysis. *Applied Energy*, *113*, 1043–1058. https://doi.org/10.1016/j.apenergy.2013.08.061

Ochoa, C. E., & Capeluto, I. G. (2009). Advice tool for early design stages of intelligent facades based on energy and visual comfort approach. *Energy and Buildings*, *41*(5), 480–488. https://doi.org/10.1016/j.enbuild.2008.11.015

Østergård, T., Jensen, R. L., & Maagaard, S. E. (2016). Building simulations supporting decision making in early design – A review. *Renewable and Sustainable Energy Reviews*, *61*, 187–201. https://doi.org/10.1016/j.rser.2016.03.045

Østergård, T., Jensen, R. L., & Maagaard, S. E. (2017). Interactive Building Design Space Exploration Using Regionalized Sensitivity Analysis. *Proceedings of the International Building Performance Simulation Association*, 10.

Østergård, T., Maagaard, S. E., & Jensen, R. L. (2015). A stochastic and holistic method to support decision-making in early building design. *Proceedings of the International Building Performance Simulation Association*, 8.

Pantazis, E., & Gerber, D. (2018). A framework for generating and evaluating façade designs using a multi-agent system approach. *International Journal of Architectural Computing*, *16*(4), 248–270. https://doi.org/10.1177/1478077118805874

Parimala, K., & Channappayya, S. (2019). Quality aware generative adversarial networks. *Advances in Neural Information Processing Systems*, *32*, 2948–2958.

Park, J. J., Florence, P., Straub, J., Newcombe, R., & Lovegrove, S. (2019). DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. *ArXiv:1901.05103 [Cs]*. http://arxiv.org/abs/1901.05103

Paulson Jr, B. C. (1976). Designing to reduce construction costs. *Journal of the Construction Division*, *102*(4), 587–592.

Pfister, H., Zwicker, M., van Baar, J., & Gross, M. (2000). Surfels: Surface elements as rendering primitives. *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '00*, 335–342. https://doi.org/10.1145/344779.344936

Phelan, N., Davis, D., & Anderson, C. (2017). Evaluating Architectural Layouts with Neural Networks. *2017 Proceedings of the Symposium on Simulation for Architecture & Urban Design*, 7.

Pinto, G., Wang, Z., Roy, A., Hong, T., & Capozzoli, A. (2022). Transfer learning for smart buildings: A critical review of algorithms, applications, and future perspectives. *Advances in Applied Energy*, *5*, 100084. https://doi.org/10.1016/j.adapen.2022.100084

Primikiri, E., & Malkawi, A. (2001). Distributed Simulations: An Object Oriented Approach. *Proceedings of the Seventh International IBPSA Conference*, 6.

Radford, A. (1988). *Design by optimization in architecture, building, and construction*. Van Nostrand Reinhold.

Radziszewski, K., & Waczy, M. (2018). Machine Learning Algorithm-Based Tool and Digital Framework for Substituting Daylight Simulations in Early- Stage Architectural Design Evaluation. *2018 Proceedings of the Symposium on Simulation for Architecture & Urban Design*, 1.

Reed, K. (2016). *Machine Learning Applications in Generative Design*.

Reeves, C. R. (2005). Fitness Landscapes. In E. K. Burke & G. Kendall (Eds.), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (pp. 587–610). Springer US. https://doi.org/10.1007/0-387-28356-0_19

Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). Stochastic Backpropagation and Approximate Inference in Deep Generative Models. *ArXiv:1401.4082 [Cs, Stat]*. http://arxiv.org/abs/1401.4082

Roudsari, M. S., Pak, M., Smith, A., & others. (2013). Ladybug: A parametric environmental plugin for grasshopper to help designers create an environmentally-conscious design. *Proceedings of the 13th International IBPSA Conference Held in Lyon, France Aug*, 3128–3135.

Rutten, D. (2007). *Grasshopper*. Robert McNeel and associates. www.grasshopper3d.com

Rutten, D. (2014). *Navigating multi-dimensional landscapes in foggy weather as an analogy for generic problem solving*. 14.

Sato, G., Ishizawa, T., Iseda, H., & Kitahara, H. (2020). Automatic Generation of the Schematic Mechanical System Drawing by Generative Adversarial Network. *ECAADe 2020*, 8.

Scopatz, A. (2021). *Pyembree* [Python]. https://github.com/scopatz/pyembree (Original work published 2015)

Shaviv, E. (1999). Integrating energy consciousness in the design process. *Automation in Construction*, *8*(4), 463–472. https://doi.org/10.1016/S0926-5805(98)00101-0

Shi, X., & Yang, W. (2013). Performance-driven architectural design and optimization technique from a perspective of architects. *Automation in Construction*, *32*, 125–135. https://doi.org/10.1016/j.autcon.2013.01.015

Sileryte, R., D'Aquilio, A., Stefano, D. D., Yang, D., & Turrin, M. (2016). Supporting Exploration of Design Alternatives using Multivariate Analysis Algorithms. *2016 Proceedings of the Symposium on Simulation for Architecture and Urban Design*, 8.

Sjoberg, C., Beorkrem, C., & Ellinger, J. (2017). Emergent Syntax. Machine Learning for the Curation of Design Solution Space. *Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*. ACADIA, Cambridge, Massachusetts.

Śmieja, M., Kołomycki, M., Struski, Ł., Juda, M., & Figueiredo, M. A. T. (2020). Iterative Imputation of Missing Data Using Auto-Encoder Dynamics. In H. Yang, K. Pasupa, A. C.-S. Leung, J. T. Kwok, J. H. Chan, & I. King (Eds.), *Neural Information Processing* (Vol. 12534, pp. 258–269). Springer International Publishing. https://doi.org/10.1007/978-3-030-63836-8_22

Soares, N., Bastos, J., Pereira, L. D., Soares, A., Amaral, A. R., Asadi, E., Rodrigues, E., Lamas, F. B., Monteiro, H., Lopes, M. A. R., & Gaspar, A. R. (2017). A review on current advances in the energy and environmental performance of buildings towards a more sustainable built environment. *Renewable and Sustainable Energy Reviews*, *77*, 845–860. https://doi.org/10.1016/j.rser.2017.04.027

Tian, W. (2013). A review of sensitivity analysis methods in building energy analysis. *Renewable and Sustainable Energy Reviews*, *20*, 411–419. https://doi.org/10.1016/j.rser.2012.12.014

Tian, Z., Zhang, X., Jin, X., Zhou, X., Si, B., & Shi, X. (2018). Towards adoption of building energy simulation and optimization for passive building design: A survey and a review. *Energy and Buildings*, *158*, 1306–1316. https://doi.org/10.1016/j.enbuild.2017.11.022

Toulkeridou, V. (2019). Steps towards AI augmented parametric modeling systems for supporting design exploration. *Blucher Design Proceedings*, 81–92. https://doi.org/10.5151/proceedings-ecaadesigradi2019_602

Tseranidis, S., Brown, N. C., & Mueller, C. T. (2016). Data-driven approximation algorithms for rapid performance evaluation and optimization of civil structures. *Automation in Construction*, *72*, 279–293. https://doi.org/10.1016/j.autcon.2016.02.002

Turrin, M., von Buelow, P., & Stouffs, R. (2011). Design explorations of performance driven geometry in architectural design using parametric modeling and genetic algorithms. *Advanced Engineering Informatics*, *25*(4), 656–675. https://doi.org/10.1016/j.aei.2011.07.009

Vierlinger, R. (2012). *Octopus*. University of Applied Arts Vienna, and Bollinger+Grohmann Engineers. https://www.food4rhino.com/app/octopus

Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, *13*(4), 600–612. https://doi.org/10.1109/TIP.2003.819861

Westermann, P., & Evins, R. (2019). Surrogate modelling for sustainable building design – A review. *Energy and Buildings*, *198*, 170–186. https://doi.org/10.1016/j.enbuild.2019.05.057

Westermann, P., & Evins, R. (2021). Using Bayesian deep learning approaches for uncertainty-aware building energy surrogate models. *Energy and AI*, *3*, 100039. https://doi.org/10.1016/j.egyai.2020.100039

Westermann, P., Rousseau, G., & Evins, R. (2021). Buildingenergy.ninja: A web-based surrogate model for instant building energy time series for any climate. *Journal of Physics: Conference Series*, *2042*(1), 012012. https://doi.org/10.1088/1742-6596/2042/1/012012

Wetter, M., & Wright, J. (2004). A comparison of deterministic and probabilistic optimization algorithms for nonsmooth simulation-based optimization. *Building and Environment*, *39*(8), 989–999. https://doi.org/10.1016/j.buildenv.2004.01.022

Whalen, E., & Mueller, C. (2022). Toward Reusable Surrogate Models: Graph-Based Transfer Learning on Trusses. *Journal of Mechanical Design*, *144*(2), 021704. https://doi.org/10.1115/1.4052298

White, T. (2016). Sampling Generative Networks. *ArXiv:1609.04468 [Cs, Stat]*. http://arxiv.org/abs/1609.04468

Wortmann, T. (2017a). Opossum: Introducing and Evaluating a Model-based Optimization Tool for Grasshopper. *Protocols, Flows and Glitches, Proceedings of the 22nd International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA)*, 283–293. http://papers.cumincad.org/data/works/att/caadria2017_124.pdf

Wortmann, T. (2017b). Surveying design spaces with performance maps: A multivariate visualization method for parametric design and architectural design optimization. *International Journal of Architectural Computing*, *15*(1), 38–53. https://doi.org/10.1177/1478077117691600

Wortmann, T., Costa, A., Nannicini, G., & Schroepfer, T. (2015). Advantages of surrogate models for architectural design optimization. *AI EDAM*, *29*(4), 471–481. https://doi.org/10.1017/S0890060415000451

Wortmann, T., & Schroepfer, T. (2019). From Optimization to Performance-Informed Design. *2019 Proceedings of the Symposium on Simulation for Architecture & Urban Design*, 8.

Wortmann, T., Waibel, C., Nannicini, G., Evins, R., Schroepfer, T., & Carmeliet, J. (2017). Are Genetic Algorithms Really the Best Choice for Building Energy Optimization? *2017 Proceedings of the Symposium on Simulation for Architecture & Urban Design*, 8.

Wu, J., Zhang, C., Xue, T., Freeman, B., & Tenenbaum, J. (2016). Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in Neural Information Processing Systems*, 82–90.

Yang, D., Ren, S., Turrin, M., Sariyildiz, S., & Sun, Y. (2018). Multi-disciplinary and multi-objective optimization problem re-formulation in computational design exploration: A case of conceptual sports building design. *Automation in Construction*, *92*, 242–269. https://doi.org/10.1016/j.autcon.2018.03.023

Yang, D., Sun, Y., di Stefano, D., & Turrin, M. (2017). A computational design exploration platform supporting the formulation of design concepts. *2017 Proceedings of the Symposium on Simulation for Architecture & Urban Design*, 8.

Yang, D., Sun, Y., Sileryte, R., D'Aquilio, A., & Turrin, M. (2016). Application of Surrogate Models for Building Envelope Design Exploration and Optimization. *2016 Proceedings of the Symposium on Simulation for Architecture and Urban Design*, 4.

Yi, Y. K. (2008). *Integration Of Computational Fluid Dynamics (CFD) And Energy Simulation (ES) For Optimal Energy Form Generation* [Doctoral Thesis, University of Pennsylvania]. http://search.proquest.com.ezp-prod1.hul.harvard.edu/docview/304494177?accountid=11311

Yi, Y. K., & Malkawi, A. M. (2009). Optimizing building form for energy performance based on hierarchical geometry relation. *Automation in Construction*, *18*(6), 825–833. https://doi.org/10.1016/j.autcon.2009.03.006

Zhang, H., & Blasetti, E. (2020). 3D Architectural Form Style Transfer Through Machine Learning. *CAADRIA 2020*, 10.

# 10. Appendix

## A. Simulator Validation

Figure 10.1 shows the total incident radiation during cooling and heating periods for 400 procedurally generated geometries using the Surfels representation. Simulations performed using the custom solar radiation module are compared against simulations using Ladybug. Results from the two simulators are always very close to each other.

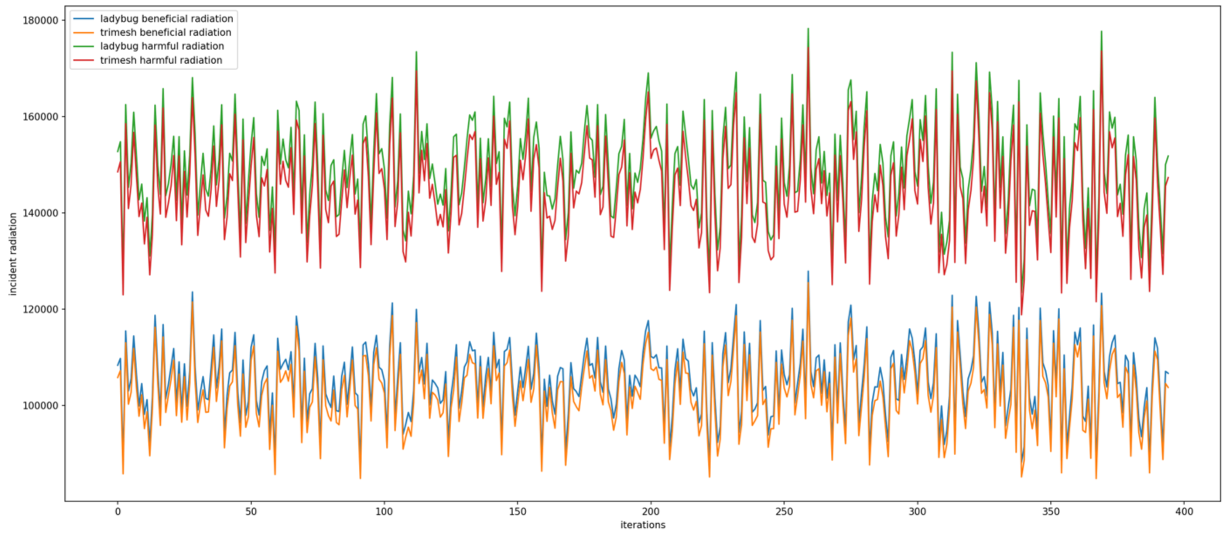


*Figure 10.1 Ladybug simulations vs custom module simulations.*

## B. Boolean Union Effect

The performance of geometries optimized without applying a Boolean union operation to the cuboids (dataset v2) is compared to that of geometries optimized with the union operation (v2.01). Optimized geometries from dataset v2 were post-processed by applying the union operation and their performance was simulated again. Figure 10.2 and Figure 10.3 show examples of the optimized geometries' performance distribution for 8 different sites. In some cases data from v2.01 form a better Pareto front than data from v2.0; however, in most cases the performance distributions appear very similar.

Figure 10.4 compares the hypervolumes of the Pareto sets from the two datasets with respect to the solar gain, volume, and area objectives. Dataset v2.01 does not appear to have a strong advantage over dataset v2. However, this may partially be due to the use of an additional objective (areaheight) in dataset v2.01. In this work the process without the Boolean union was selected because of the disproportionate computational cost of the union operation compared to its unclear building performance advantage.

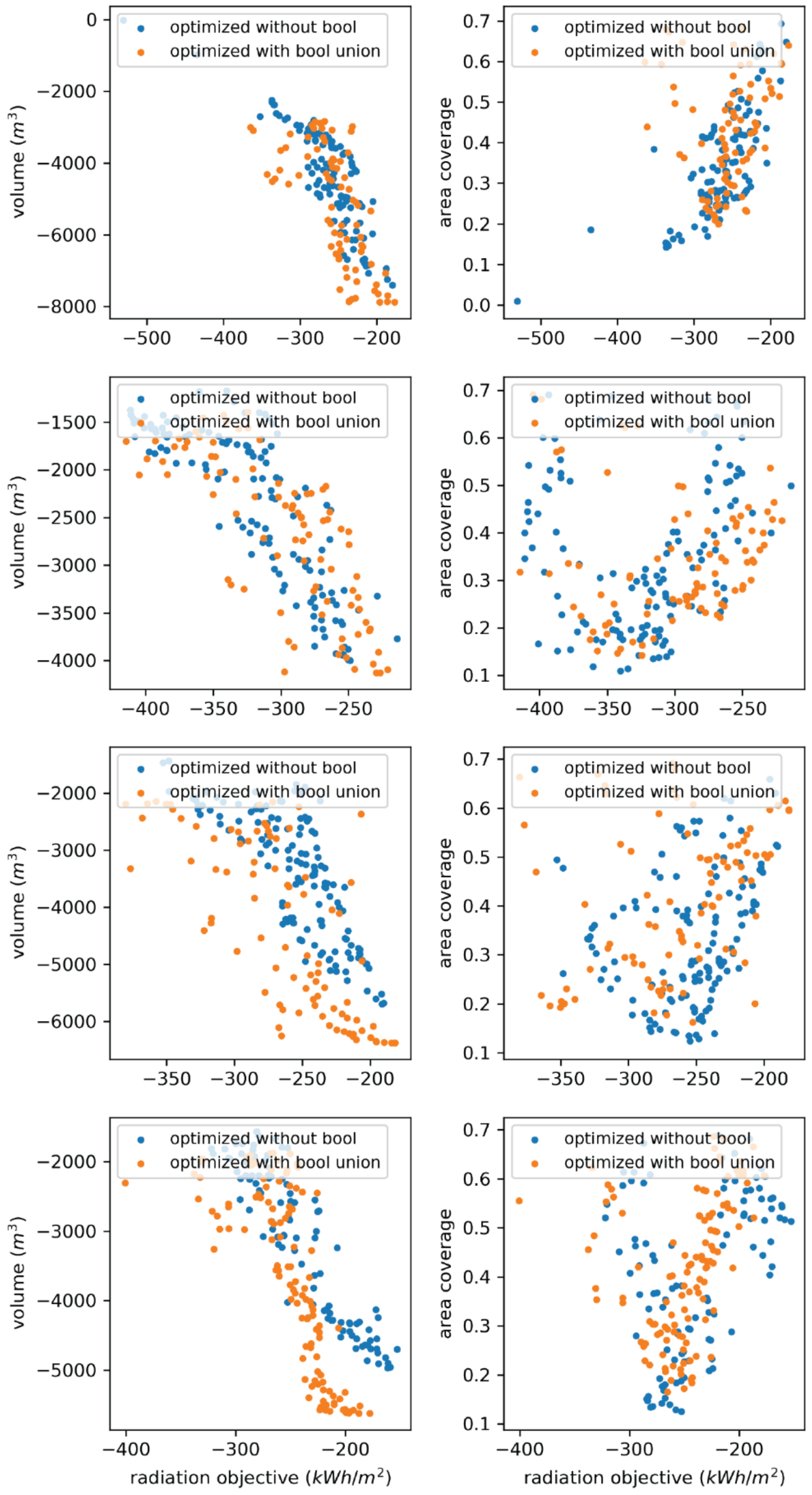


*Figure 10.2 Examples of performance distribution for geometries optimized with and without merging the cuboids through the Boolean union operation.*

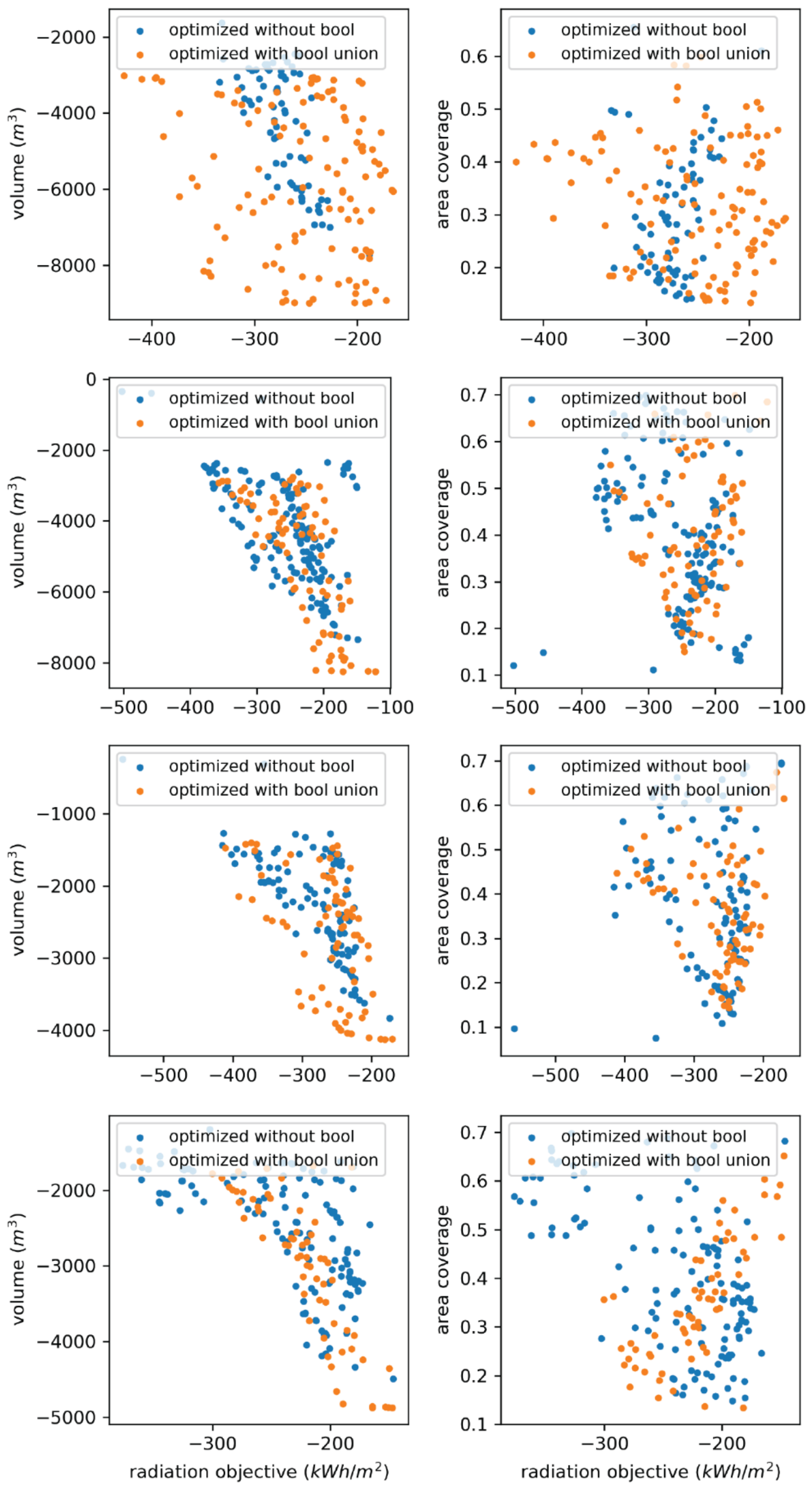


*Figure 10.3 Examples of performance distribution for geometries optimized with and without merging the cuboids through the Boolean union operation.*

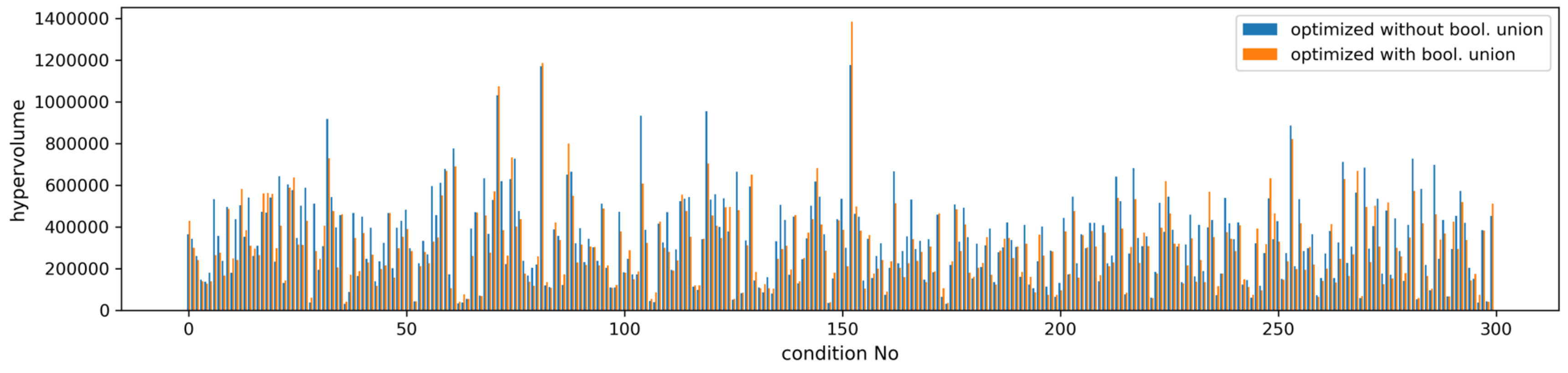


*Figure 10.4 Hypervolumes of performance objectives for all 300 conditions (sites) for geometries optimized with and without merging the cuboids through the Boolean union operation.*

## C. Aggregate performance loss

To better visualize the effect of aggregate error terms in the loss function of the VAE, a simple experiment was performed using the MNIST dataset. In analogy to the cuboids geometry representation, the MNIST dataset was interpreted as shapes made out of square elements, where a pixel brightness corresponds to the size (side length) of a square element. The analogous 'performance' terms to the volume and cover area of the cuboid geometries were formulated as the total area of the squares, and their projection length on the X-axis. Similar to the threshold applied at the cuboids, smaller elements ($v < 0.2$) were filtered out during the calculation of the area and projection.

The dataset was resampled to a resolution of 16X16, and a VAE model was created with an encoder using two convolutional layers (16 filters, kernel size 3, stride 1, followed by 32 filters, kernel size 3, stride 2) and a fully connected layer, and a mirrored decoder. A ReLu activation was used for all layers except for the final one which used a sigmoid. A latent space of 8 dimensions was used.

The baseline reconstruction error was defined as the regularized cross entropy (corresponding to the log loss of a continuous Bernoulli distribution). The projection and the total area errors were calculated as the squared difference of the respective total quantities. All models were trained for 50 iterations using the Adam optimizer with learning rate = 1e-4.

Figure 10.5 and Figure 10.6 visualize 8 test set samples and their reconstructions from VAE models that were trained using different combinations of reconstruction error terms. In Figure 10.5 the projection and area errors have been weighted too high. Reconstructions from the model that used the projection error appear to better match the total width of the original shapes comparing to reconstructions that only used the baseline error. However, the distribution of mass along the Y-axis may deviate from the ground truth. For example, the shape in the third row-third column matches the width of the ground truth more accurately than the baseline reconstruction, but this happens at the expense of the total shape similarity. In other cases, such as in row 5, a different, but not less accurate, reconstruction than the baseline was produced. Last, in row 7, the introduction of the projection term led to an overall more similar shape to the ground truth, than the one produced by the baseline. A similar pattern can be observed with the introduction of the area error term, as well as the combination of all three errors (columns 4 and 5).

On the other hand, the weighting of the projection and area terms have been fine-tuned in the VAE models used in Figure 10.6. As a result, the various combinations not only improve the respective metrics, but also the overall similarity of the reconstructed shapes to the ground truth. Rows 5 and 7 are notable examples, where the baseline reconstruction was very far from the ground truth, but the addition of the aggregate error terms significantly improved the resulting shape similarity. Row 8 is another interesting example, where each of the partial errors achieved a decent reconstruction, but the final combination produces the best result.

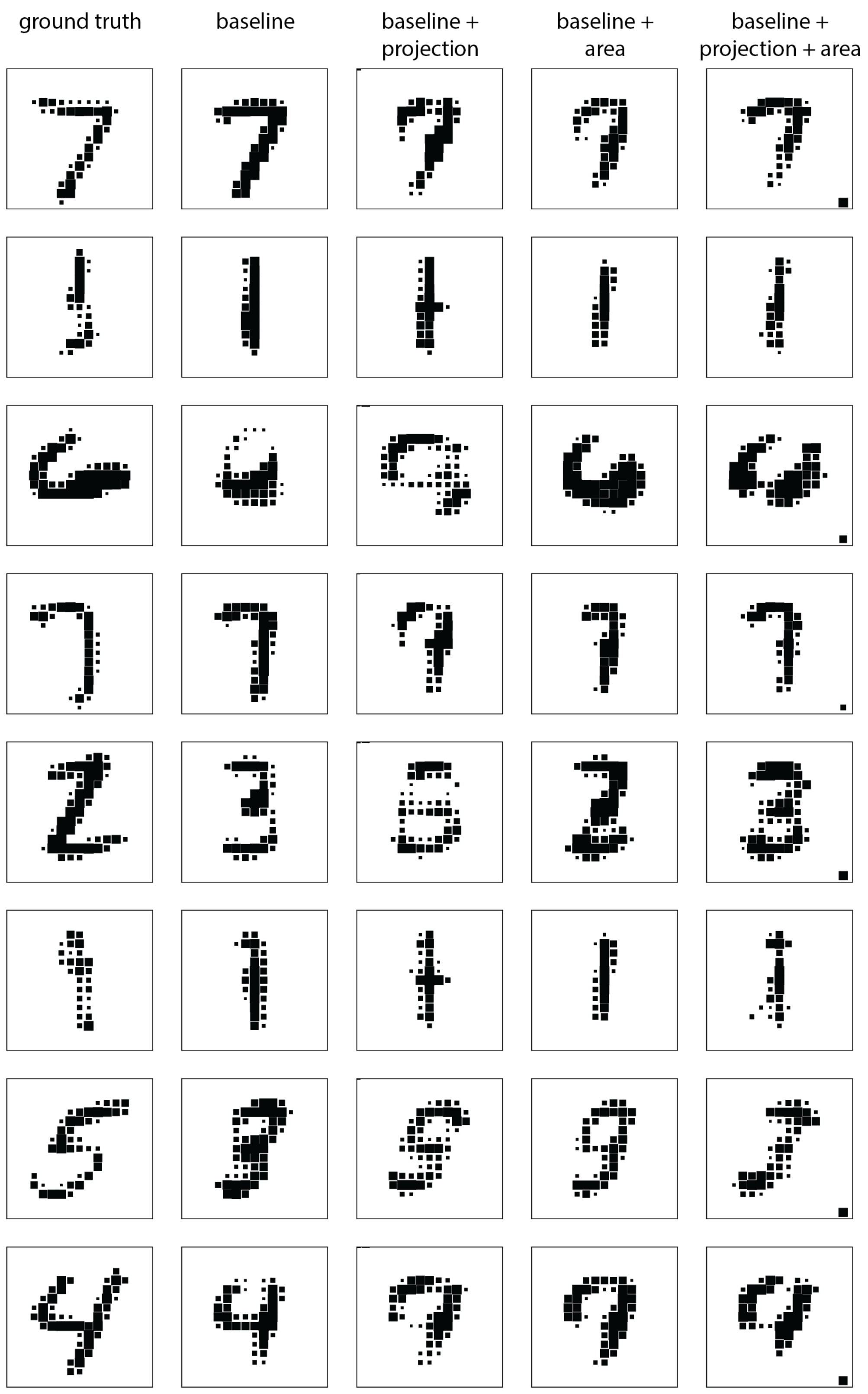


*Figure 10.5 Test set samples and reconstructions using different VAE models. Reconstruction error terms not properly weighted.*

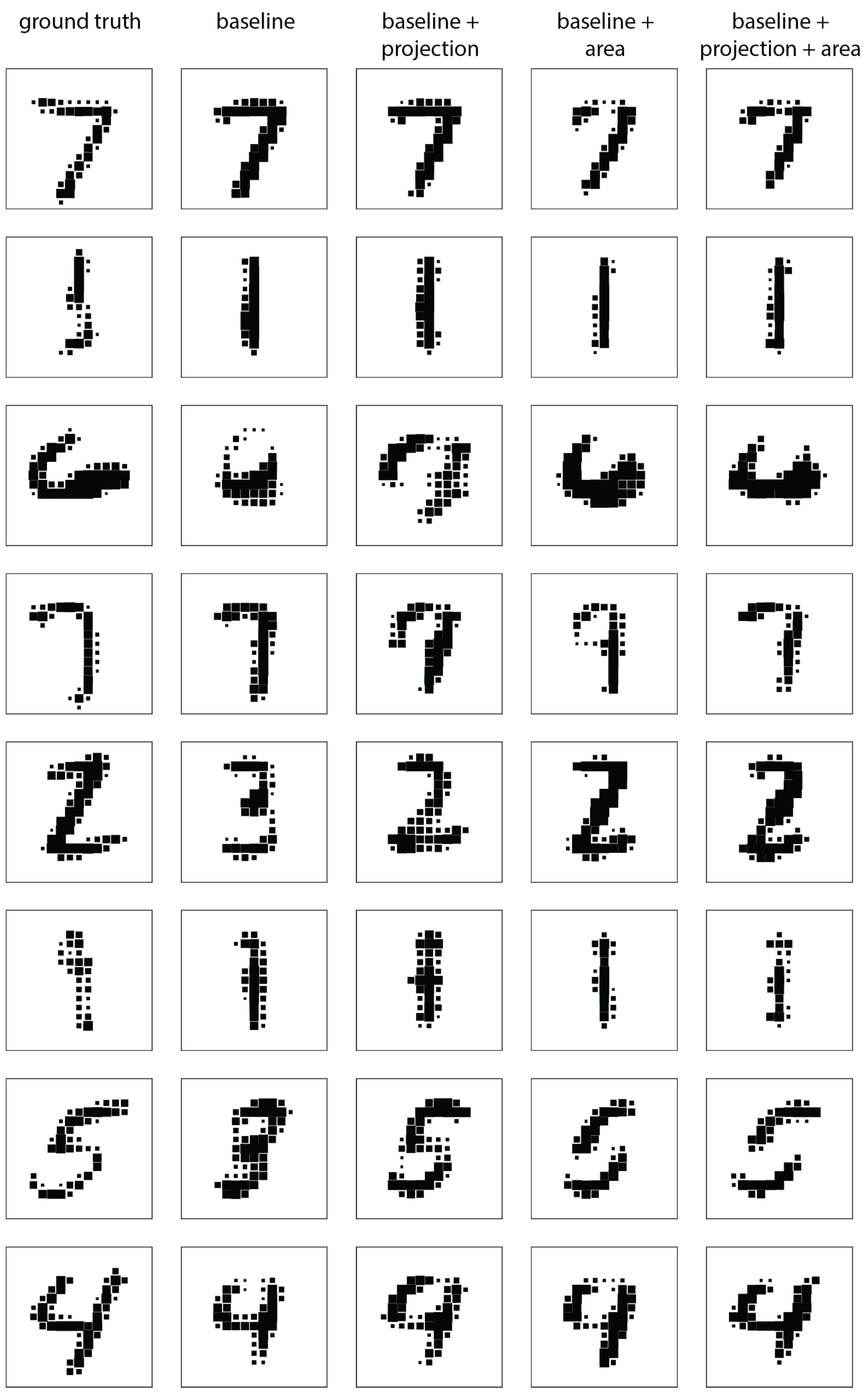


*Figure 10.6 Test set samples and reconstructions using different VAE models. Reconstruction error terms have been fine-tuned.*