Chair of Computational Modeling and Simulation
TUM School of Engineering and Design
Technical University of Munich

From Fabrication Information Models to Simulation Models

Scientific work to obtain the degree

Master of Science (M.Sc.)

at the TUM School of Engineering and Design of the Technical University of Munich.

Supervised by M.Sc. Oguz Oztoprak
M.Sc. Martin Slepicka
Chair of Computational Modeling and Simulation

Submitted by András László Aninger ([redacted])
[redacted]

Submitted on 27. May 2022

Acknowledgements

This thesis would not have been possible without the guidance of my supervisor, Oguz Oztoprak. I am sincerely grateful for supporting me with all of his expertise in Linux, whenever I struggled as a novice to get things up and running. I wish to thank him for his great explanations of different complex concepts in C++ and the Finite Cell Method and for providing all the RAM he could get his hands on to help me run my simulations. Finally, I am very grateful for his thorough overview of my thesis. His suggestions helped a lot to improve the quality of this work. I could not have wished for a better qualified person to support me while working on my thesis.

My sincere thanks go to my co-supervisor, Martin Slepicka for providing all the wall geometry examples on which I was able to showcase the capabilities of the tool implemented in the scope of this thesis. His expertise in Fabrication Information Modeling and 3D printing in general also helped me to gain a better understanding of this research area, and he kept me motivated by pointing out the possibilities, where my work can have practical use. Furthermore, I wish to thank him for the excellent LaTeX template he wrote. Thanks to it, I was able to focus on the content more, and had to put less effort into formatting everything correctly.

Also, I greatly appreciate Dr.-Ing. habil. Stefan Kollmannsberger's suggestions on how to make calculations run faster in the chair's simulation framework.

I am grateful to the Chair of Computational Modeling and Simulation, and the IT Team of TUM for providing me a workstation to run the necessary simulations on, which would not have been possible on my personal computer.

I would also like to thank my friend, Pedro, for always being there to share a good Carbonara with me and listen to my struggles.

Finally, I would like to express my sincere gratitude to my parents for enabling me to study what I have wished for, and for their love and support whenever I was struggling to achieve the goals I set for myself.

Abstract

In recent decades Additive Manufacturing (AM) has matured to a new, but viable alternative to traditional construction methods in certain areas of the industry. To follow up these developments and satisfy the needs of the industry, significant effort was already made to provide design frameworks better suited for the peculiarities of the technology. Fabrication Information Modeling (FIM) represents such a framework, which provides tools to extend the classical Building Information Modeling (BIM) with manufacturing models for AM, including the generation of print paths for 3D printers. The work done in this thesis aims to enhance FIM by providing a way to generate volumetric models based on the print path and to conduct numerical simulations on them. Physical insight to the behaviour of the ”as-planned” geometry then could be used to improve the print path design. To facilitate the later automation of finding best designs, some overall physical performance measures were suggested and made available in a simple file format as simulation output.

Zusammenfassung

In den letzten Jahrzehnten hat sich die Additive Fertigung (englisch: Additive Manufacturing, AM) zu einer neuen, aber geeigneten Alternative zu traditionellen Konstruktionsmethoden in bestimmten Bereichen der Bauindustrie entwickelt. Um diesen Entwicklungen nachzukommen und den Bedürfnissen der Industrie gerecht zu werden, wurden bereits erhebliche Anstrengungen unternommen, um Frameworks für Design und Planung bereitzustellen, die den Besonderheiten dieser Technologie besser gerecht werden. FIM stellt ein solches Framework dar, das Werkzeuge bereitstellt, mit denen ein klassisches BIM um Fertigungsmodelle für AM erweitert werden kann, einschließlich der Erzeugung von Druckpfaden für 3D-Drucker. In dieser Arbeit wird das FIM Framework erweitert, indem eine Möglichkeit geschaffen wird, volumetrische Modelle auf Basis des Druckpfades zu generieren und numerische Simulationen mit diesen Modellen durchzuführen. Ein solcher physikalischer Einblick in das Verhalten der Geometrie kann dann herangezogen werden um die Gestalt des Druckpfads zu verbessern. Zur Erleichterung der späteren Automatisierung bei der Suche nach den besten Gestaltungsmöglichkeiten, wurden einige allgemeine physikalische Größen vorgeschlagen und in einem einfachen Dateiformat als Simulationsausgabe zur Verfügung gestellt.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

2D	two dimensional
3D	three dimensional
AM	Additive Manufacturing
BC	boundary condition
BIM	Building Information Modeling
BOBYQA	Bounded Optimization By Quadratic Approximation
BVP	boundary value problem
CMS	Chair of Computational Modeling and Simulation
CPU	Central Processing Unit
CSG	Constructive Solid Geometry
ESA	European Space Agency
FCM	Finite Cell Method
FEM	Finite Element Method
FIM	Fabrication Information Modeling
GE®	General Electric®
Pardiso	Parallel Direct Sparse Solver
PMC	Point Membership Classification
RAM	Random-access memory
STL	stereolithography

Chapter 1

Introduction

1.1 Developments in 3D printing for construction

In the recent years, several successful commercial 3D printed construction projects of different companies were reported about, including the opening of Germany's first 3D printed residential building in Beckum (see fig. 1.1, furthermore, in PERI® (2020) and PERI® (2021)). This development proves that the method is not merely an area of intensive research anymore, but a viable alternative to traditional construction methods.

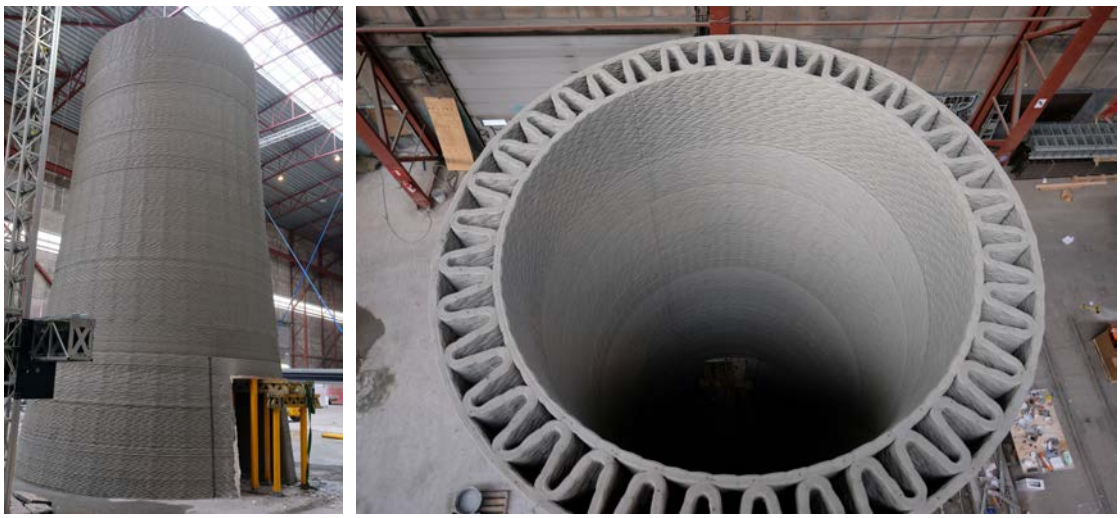
(a) (b)

Figure 1.1: 3D printed residential building in Beckum finished (a) (PERI®, 2021) and under construction (b) (PERI®, 2020)

There are several potential benefits that drive the broader adoption of the method in the housing industry, some of which are worth stating here:

1. It promises a faster overall construction time (but a more involved planning phase), due to the possibility of uninterrupted work thanks to automation.
2. It has the potential to significantly decrease construction costs. On the one hand, due to the aforementioned speed-up in construction time, on the other hand, because of the decreased amount of human labour involved.
3. It allows for a more flexible geometric design of the walls, therefore carrying the potential for a geometry optimized for load bearing capacity per unit weight or heat insulation properties. Furthermore, aesthetic and ergonomic benefits due to the less restricted planning are also not to be neglected.

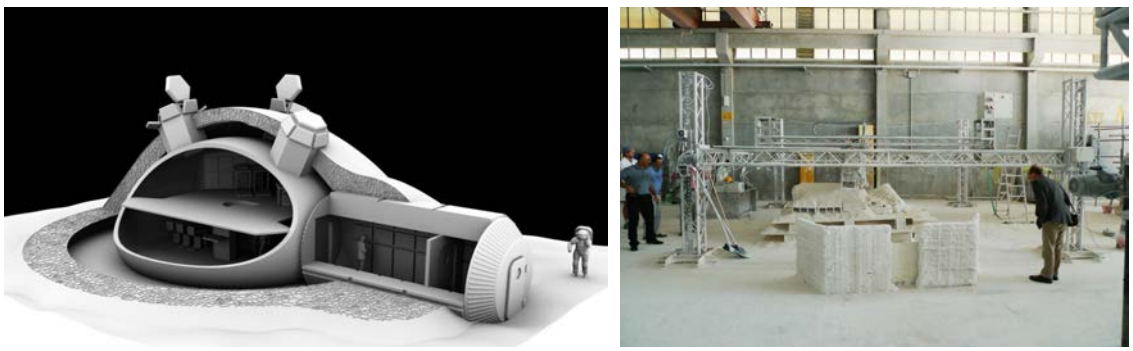
But commercial applications of the technology are not restricted to the housing industry. A collaboration between the companies GE Renewable Energy®, LafargeHolcim®, and COBOD® aims to develop an optimized 3D printed concrete base for wind turbines (GE®, 2020), of which a first prototype was already constructed (fig. 1.2). The main goal of the project is to eliminate the size constraints on the base - due to transportation restrictions - by constructing it on site with the help of 3D printing. A larger base then would allow for higher towers (up to 150-200 m), which then would allow for an increased power generation per unit with up to 33% (GE®, 2020).



(a) (b)

Figure 1.2: 3D printed concrete base for wind turbines (GE®, 2020)

The potential of full automation of the process would also allow humanity to construct housing units, where building with traditional methods is impossible. The European Space Agency (ESA) has already started exploring the possibility of 3D printing a base on the Moon by using local material (ESA, 2013). On fig. 1.3a, a concept of a lunar housing unit can be seen, while on fig. 1.3b, an ongoing printing experiment with lunar soil-like material was photographed.



(a) (b)

Figure 1.3: Concept of a lunar base (a) and test print with lunar soil-like material (b) (ESA, 2013)

Due to all of the aforementioned potentials of the technology and the rising needs of the industry behind it, more and more research is done to develop tools and techniques for the improvement of design and analysis of 3D printed walls. In the following section, a construction technique based design process, called Fabrication Information Modeling (FIM), will be introduced based on the work done by SLEPICKA (2021), and the scope of this thesis within that framework will be defined.

1.2 Fabrication Information Modeling - A construction technique based design process

The potential benefits of the 3D printing technology mentioned in the previous section come at the cost of a more involved planning phase for the engineers. Due to the complex technology of this construction method, it is best to consider the limitations and capabilities of the 3D printing process in the design as early as possible. Establishing a digital design tool for the purpose to bridge the gap between Building Information Modeling (BIM) and the actual construction is the aim of Fabrication Information Modeling (FIM). In the work of SLEPICKA (2021), the structure depicted on fig. 1.4 for combining the individual building blocks of FIM was suggested.

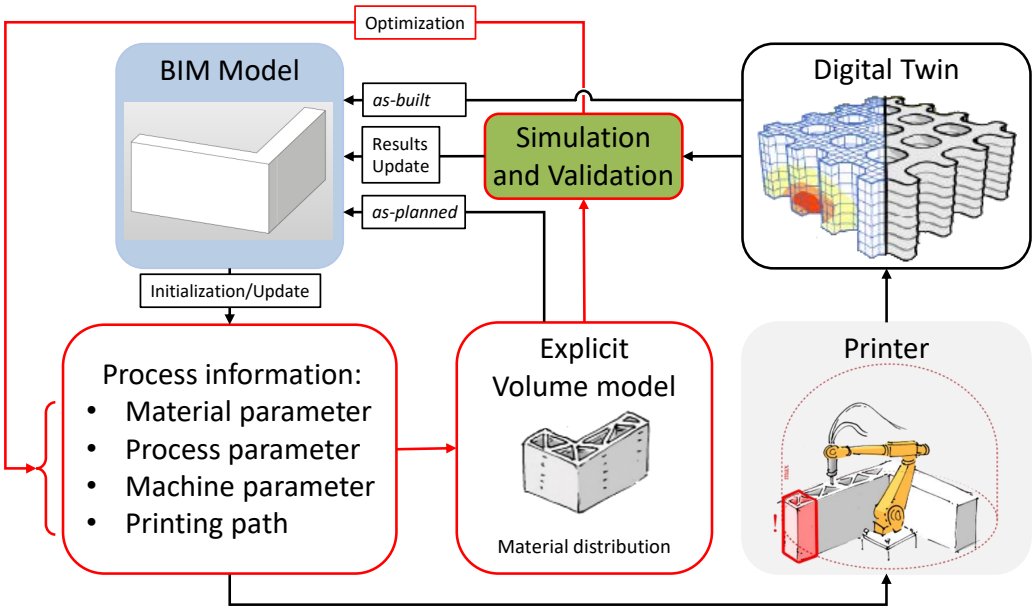


Figure 1.4: The building blocks of FIM with optimization loop (SLEPICKA, 2021)

Within the scope of 3D printing, the main responsibilities of FIM lie with deriving the process information from a BIM model of a building, and the creation of a print path based on this data. However, as described by SLEPICKA (2021), and depicted on fig. 1.4, a FIM can optionally include two other main components as well. On the one hand, one could record information about the structure during construction to create a digital twin of the object ”as-built”, which can then be attached to the BIM model of the building. On the

other hand, from the generated print path, a volumetric model of the wall could be created to represent the structure ”as-planned”. Using this geometry, simulations for assessing the wall design's physical behaviour can be made, and based on these results, the design can be updated and the print path regenerated. This optimization loop (highlighted in red on fig. 1.4) would provide a way to a potentially highly automated construction information based process for finding best designs according to certain criteria. This latter extension of the defined FIM process is in the focus of the current thesis, which will be elaborated on more in the next section (section 1.3).

1.3 The goal of the thesis

The aim of this work is to provide a tool which can close the mentioned optimization loop. For this, an approach for generating 3D printed wall geometries based on the print path information needs to be found. These geometries then can be used to perform numerical simulations on. Finally, an option must be provided to link back this information to the design process, so that a better path can be generated for the actual construction. Such a tool would enable engineers to find optimal designs for path geometries based on the estimated physical performance, while still working in the fabrication driven design environment.

At the Chair of Computational Modeling and Simulation (CMS), there was already extensive research done on how to do numerical analysis on different geometric representations. In the work by WASSERMANN (2020), several options for the description of a simulation geometry for computations were examined, and partially implemented by him in the chair's C++ simulation framework, called AdhoC++ (CMS, 2022) . The implementation makes use of a numerical technique called the Finite Cell Method (FCM) (PARVIZIAN et al., 2007), detailed in section 2.4.2, which allows for simulations on a wide range of geometric representations without the need for a boundary conforming mesh, in contrast to the Finite Element Method (FEM) of which FCM is a specific extension of. Therefore, the goal of this work - as depicted in fig. 1.5 - is to link simulations performed on print-path based geometric models into the FIM framework designed by SLEPICKA (2021), to provide a way to update the generated print paths, based on the determined physical properties of the design.

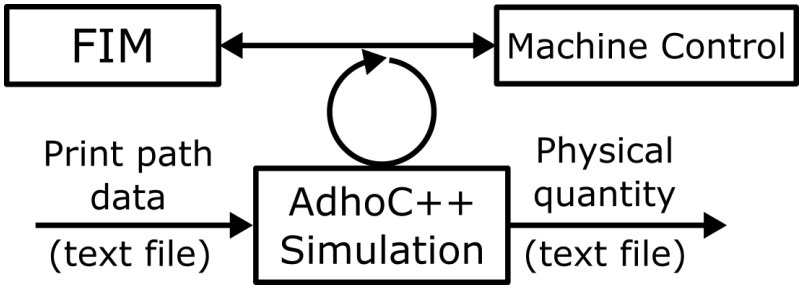


Figure 1.5: The place of AdhoC++ simulations in the design loop

Chapter 2

Theoretical background

To be able to provide the predictions about the physical behaviour of a print path design for the optimization process as described in section 1.3, a computational model must be set up. In this chapter all necessary theoretical background for that process will be reviewed, so that the context of the specific implementation in chapter 3 is properly set. After an initial review of the general modeling process for setting up computational models for physical phenomena in section 2.1, the individual steps of this workflow will be elaborated on in the context of the current application. In section 2.2, various options for representing geometric models in AdhoC++ (CMS, 2022) are described. Afterwards in section 2.3, the relevant mathematical models for linear heat conduction and linar elasticity will be reviewed, followed by the introduction of the Finite Cell Method (FCM) discretization procedure and its relation to the Finite Element Method (FEM).

2.1 The process of modeling physical phenomena

When trying to model the physical world around us, one must acknowledge its incredible complexity and our limited understanding of it. A natural phenomenon consist of several complicated processes taking place simultaneously, from molecular reactions within a body to the interactions of multiple objects on the macroscopic level. Describing all these intertwined natural occurrences to foresee the outcome of an event is impossible. Even if our theoretical knowledge would be advanced enough to describe all phenomena occurring in nature, the computational capacity we will ever posses is highly unlikely to be sufficient to solve such a model.

With all that acknowledged, an engineer still needs to make very accurate predictions about the physical behaviour of different structures. However, these much needed predictions are restricted to only some key aspects of the entirety of the phenomenon, e.g., for avoiding the plastic deformation of a planned machine part or the overheating of a computer chip. This strict focus on the most relevant processes is what allows engineers to build computational models for examining the expected behaviour of structures in the planning phase.

A general workflow for defining such computational models is described by Felippa (2004), and an adoption of the process defined in his work is depicted in fig. 2.1. The first step is a conceptual one. The engineer most identify the aforementioned relevant aspects of a structure and all the phenomena affecting it under its planned operation. This involves deciding about how detailed of a geometric description is needed, so that the idealized domain defines all relevant features which have a strong influence on how the physical phenomena of interest are taking place. For example in the case of examining 3D printed

walls, sharp rounded corners and the curved shape of the sides of a printed layer are relevant, but not the granular texture of the surface of printed concrete. A decision also has to be made about how to best store this information in a computer-readable format. In the work of WASSERMANN (2020), several possibilities for geometric representation are discussed in the context of the Finite Cell Method (FCM) (detailed in section 2.4.2). In section 2.2, only the relevant ones for the current application are detailed.

The second idealization step involves choosing (or in some cases deriving) a mathematical equation representing a physical phenomenon the engineer is interested in. In the case of 3D printed walls, for examining the heat insulation properties, the heat equation (section 2.3.1) is relevant, while for calculating the structural reliability of it, the equations of linear elasticity (section 2.3.2) will be used. For describing the interaction of the model with its environment, boundary conditions must be modeled and defined, of which the penalty approach (section 2.4.4) is of most relevance for the developed 3D printed wall modelling application.

The result of these idealization steps is a mathematical model, which already only describes a filtered version of reality. Nevertheless, for practical cases it is still seldom possible to get an analytical solution of it.

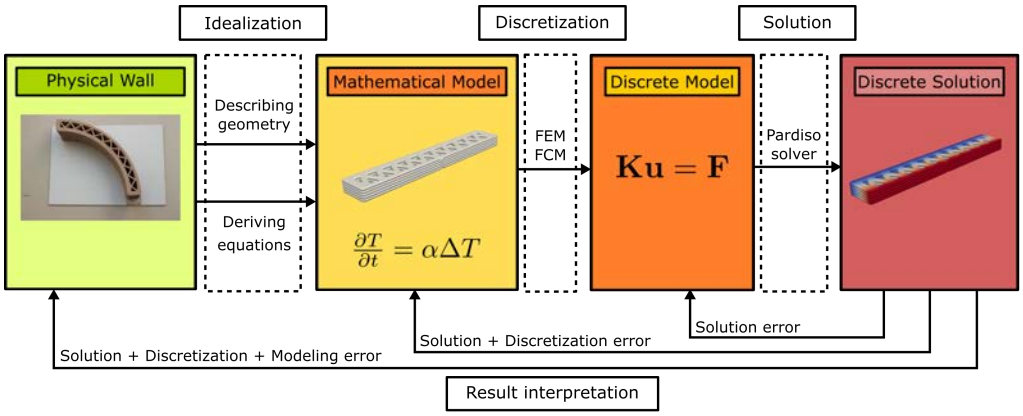


Figure 2.1: The modeling process adopted from (FELIPPA, 2004)

However, a fine approximation of the solution of this continuous mathematical model is achievable by utilizing computers for the calculation. For this, one needs to create a model, which can be stored and processed by computers. Therefore, a discretization scheme must be applied on the continuous model. By far the most well-known one of these is the Finite Element Method (FEM) (section 2.4.1 and (FELIPPA, 2004)). In this work, and in the AdhoC++ (CMS, 2022) framework in general, a modified higher order version of this, the Finite Cell Method (FCM) is used (section 2.4.2 and (PARVIZIAN et al., 2007)). These mathematical tools help engineers to turn the defined continuous description into a discrete model, which is representable by a system of linear equations.

Such a model description can be already processed and solved on a computer with the help of a linear solver. These can be iterative or direct, like the Parallel Direct Sparse Solver (Pardiso) from the Intel® oneAPI Math Kernel Library, which the AdhoC++ (CMS, 2022) project is using for models with penalty boundary conditions (section 2.4.4) applied.

This last solution step is usually the source of the least amount of error in the entire modeling procedure. It comes down to the finality of the computer-representable floating point numbers, and the specifics of the solver algorithm. The other two steps - idealization and discretization - are at most importance to get it right. It requires a good engineering judgement to decide about the proper mathematical models and discretization schemes, to end up with a model that's predictions are indeed of physical relevance for the problem at hand. The results one gets from such computations must always be interpreted in the context of the assumptions made in the mathematical model, and strategies picked to discretize it.

2.2 Representation of the model geometry

The first one of the idealization steps - mentioned in the previous section - is choosing a representation for the geometry of the object to be analyzed. For the current application, it was decided that this representation must be capable of resolving smaller details of a printed wall, like smaller, sharp corners, and the curved sides of the cross section of a layer, but the inner volume of it was assumed to behave the same way everywhere. Therefore, there was no need for representing subdomains of a layer, where e.g., different material behaviour could be assigned.

In the AdhoC++ (CMS, 2022) framework, there are three options found relevant for this project. On the one hand, computation on a boundary representation (WASSERMANN, 2020) of the domain is possible using the stereolithography (STL) file format, briefly introduced in section 2.2.1. On the other hand, solid models of the computational geometry can be generated by either using a voxel representation (section 2.2.2), or building up the domain from smaller primitives with the help of Constructive Solid Geometry (CSG), detailed in section 2.2.3. For the implementation of the analysis tool, detailed in chapter 3, the latter of the three option was chosen.

Regardless of the exact implementation, one criteria has to be met by all geometric representation that is intended to be used for analysis in AdhoC++ (CMS, 2022) . Namely, a Point Membership Classification (PMC) method must be provided with it, which determines whether a certain point is part of the geometry or not.

2.2.1 The stereolithography (STL) file format

One of the earliest format to be used in Additive Manufacturing (AM) was the STL standard (described by LIOU (2007) and ALL3DP (2021)). It approximates the boundary of the geometry to be printed with the help of small triangles, like the ones shown in fig. 2.2.

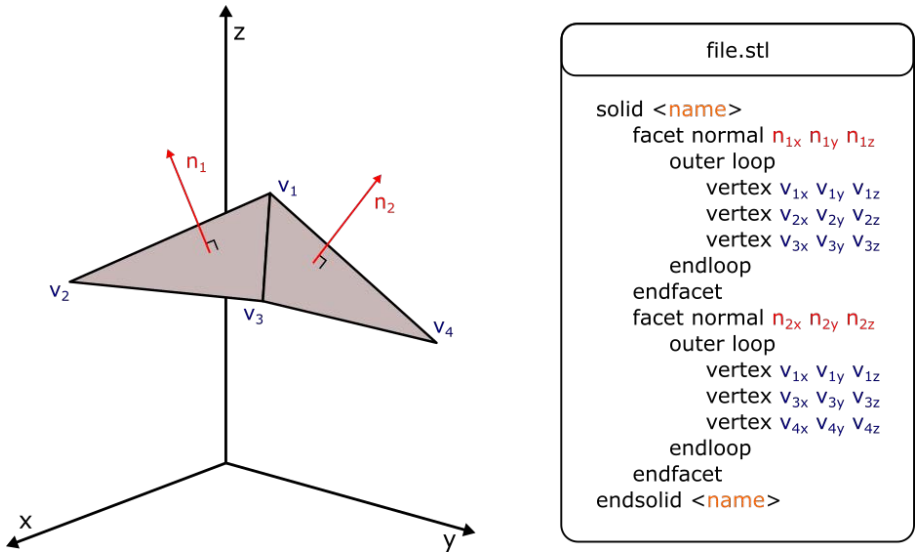


Figure 2.2: The ASCII STL boundary representation format

The information that is needed to be stored about each triangle is its face normal and its three vertices in a counter-clockwise order, like it is shown on the right side of fig. 2.2. This information can be encoded in an ASCII text base format, which is used in AdhoC++ (CMS, 2022) as well, or in a binary file.

A PMC algorithm for STL surface description using ray tracing was implemented by WASSERMANN (2020), and successfully used to do numerical analysis on flawed STL boundary representation models like the one depicted on fig. 2.3.

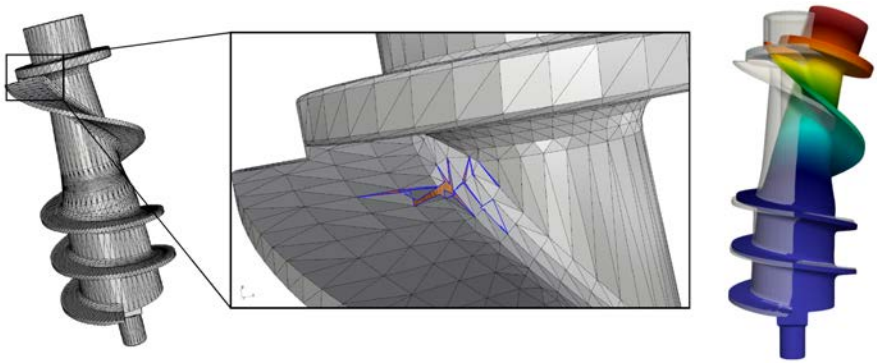


Figure 2.3: Analysis on a flawed STL geometry of a screw (WASSERMANN, 2020)

2.2.2 Voxel representation of a geometry

Opposing to the boundary surface description of an STL format, a voxel geometry encodes volumetric information about an object. To represent a geometry with the help of voxels, one must provide a way to generate a fine enough regular grid in 3D, so that by marking individual cubes on the domain as part of the geometry, the final volume represents the object with the desired accuracy. In general, in a voxel file each voxel in the grid can represent a single or multiple scalar values. So by assigning values to the different voxel cells below and above a certain threshold, the domain of the geometry can be separated from the cells describing its surroundings, like it is depicted in fig. 2.4 for a 2D example.

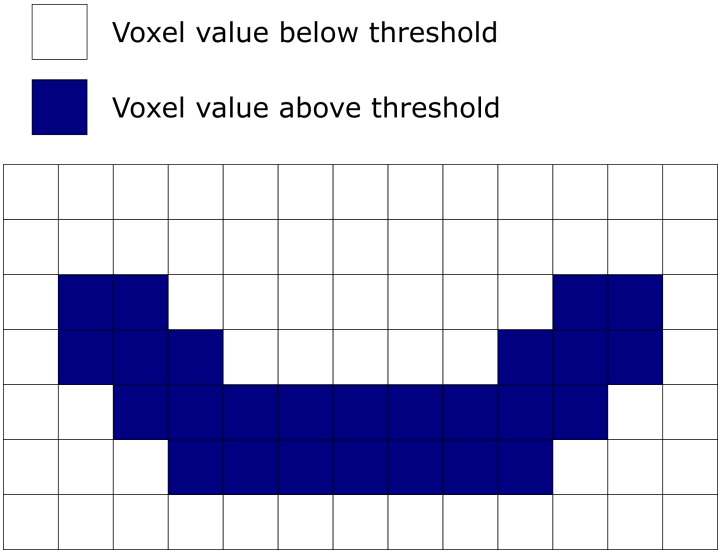


Figure 2.4: The voxel representation of a geometry

The main field of application of voxel geometries for simulations is the medical one, due to the fact that many diagnostic tools provide the output information in this format. Extensive research on the use of voxel geometries for simulations was done by YANG (2011). In the work of YANG et al. (2012), an efficient way to combine the PMC checks into the integration process was suggested and used to analyze the mechanics of bones (fig. 2.5).

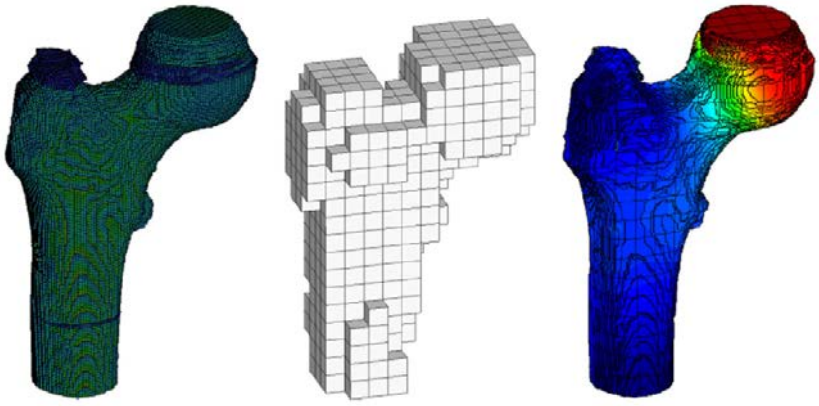


Figure 2.5: Analysis example on voxel domain, depicting the voxel geometry (left), the used mesh (center) and the final results(right) (YANG et al., 2012)

The benefit of using a voxel geometry for analyzing 3D printed wall structures would lie in the possibility to assign multiple scalar values to a single cell. This would mean that not only information about the inclusion into the geometry can be stored cell-wise, but also data about e.g., material properties. A geometric representation like this would allow for studying the effects of changes to the material composition over the structure.

2.2.3 Constructive Solid Geometry (CSG)

In the case of Constructive Solid Geometry (CSG), the volume of the solid geometry is still built up by combining individual building blocks, like it was done in the case of voxel cells, but primitives different from cubes are allowed. The way to unite them; however, does not happen by means of a structured grid, but by building a binary tree with the help of logical operations on pairs of primitives. A basic example (taken from the work of WASSERMANN (2020)) for building up a complex shape from primitives of cylinders, cubes and spheres can be seen on fig. 2.6.

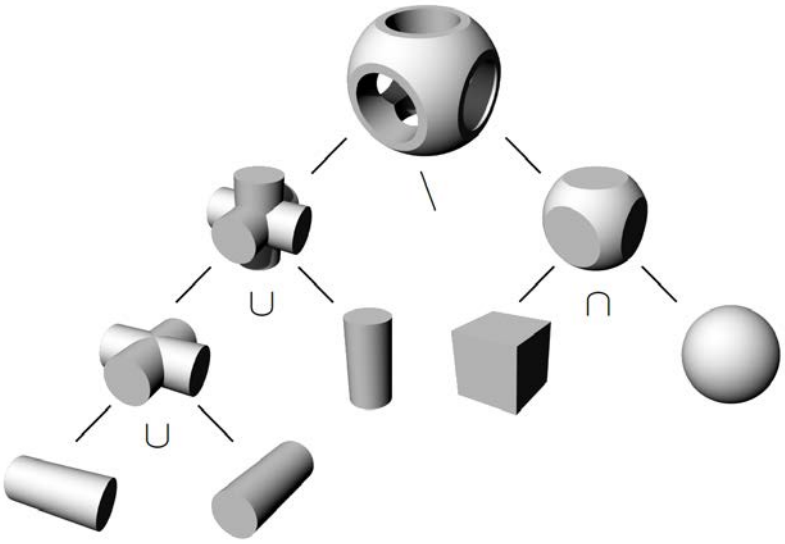


Figure 2.6: The CSG representation of a geometry (WASSERMANN, 2020)

A Point Membership Classification on CSG can be conducted in the following manner:

Each parent node in the binary tree calls the point inclusion test on its child nodes and combines the returned information (”true” for inside and ”false” for outside) with the help of the logical operation defined at the node. This call is repeated recursively, until a leaf node with a primitive geometry is reached. For each of the primitives, there is a PMC method implemented, which returns the desired information. Then this information from the primitives gets propagated back to the root of the tree, while performing the logical operations on them, defined at each node.

Since the geometric representation, implemented in this thesis (see chapter 3), creates the final domain of the wall as union of sweep primitives, the generation of them, and the conduction of the PMC tests will be briefly overviewed in the next section.

PMC on a primitive sweep

A sweep geometric primitive is defined by a cross section and a path on which the cross section is ”swept” along, as illustrated in fig. 2.7. Therefore, at every point of the curve, the cross section is the same as it is defined at the beginning of the path, just oriented differently. This fact is made good use of in the PMC algorithm provided for sweeps by WASSERMANN (2020).

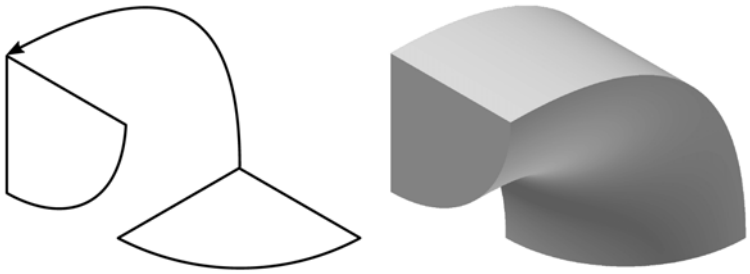
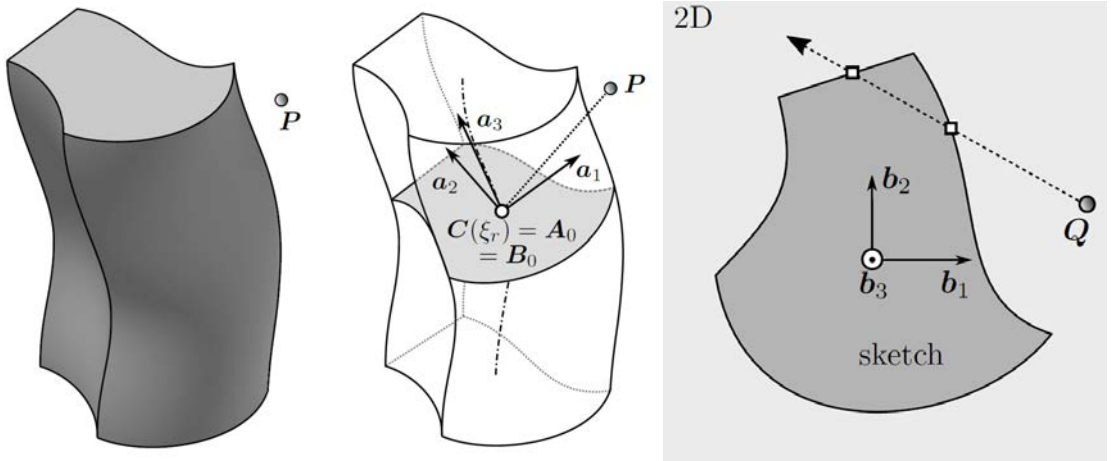


Figure 2.7: The generation of a CSG spweep primitive (WASSERMANN, 2020)

The Point Membership Classification for a sweep geometry (fig. 2.8a) consists of two main steps. First, the cross section corresponding to the tested point needs to be found (fig. 2.8b). If the local coordinate system of the cross section was defined to agree at every point with the Frenet-basis of the curve, for the tested point, the cross section at the closest curve point to it can be taken. For more complicated cases the reader is referred to WASSERMANN (2020). Then, the tested point ”P” needs to be transformed into the local coordinate system of the cross section (”Q” on fig. 2.8c). Finally, having the point in the same reference frame, a point inclusion test on the cross section geometry in 2D can be carried out with the help of ray tracing (WASSERMANN, 2020).



(a) (b) (c)

Figure 2.8: To test the inclusion of ”P” into the sweep geometry (a), the right point on the path with the corresponding cross section has to be found (b), then the PMC can be performed on the initial cross section of the geometry with the help of ray tracing (WASSERMANN, 2020).

2.3 Physical models for the simulation

The simulation tool is intended to examine the heat conducting properties and load bearing capacity of 3D printed wall designs. The goal is - as it was stated earlier - to determine best performing print paths according to certain physical quantities used as performance measures (see section 3.6.2 for further details) and update the FIM data accordingly.

To model the two physical phenomena mathematically, linear, steady state equations were taken (no transient processes are of interest for this application), briefly summarized in section 2.3.1 and section 2.3.2. The mentioned sections only give an overview of the different equations of the final idealized model, which is then discretized and solved using the AdhoC++ (CMS, 2022) framework. This short introduction to the two models meant to shed more light on the idealization modeling step for the physics that govern the simulations, and to help to better understand, how the mathematical models of the problems are structured (section 2.3.3). This general structure of the models is made good use of in AdhoC++ (CMS, 2022) for building up a differential operator for the primary solution field of a problem, mentioned also in section 3.1.

Since the tool implemented in the scope of this thesis heavily relies on the work done by WASSERMANN (2020) and his implementation of those concepts in AdhoC++ (CMS, 2022) , for the review of the linear elasticity equation and the introduction of the Finite Element Method (FEM) and the Finite Cell Method (FCM) his PhD thesis will be followed. A more detailed description of the same physical modeling steps, however, can be found in the work of KRYSL (2010). For the derivation of the heat conduction model and an introduction to the finite element formulation of it, the reader is referred again to KRYSL (2010). Since his notation is analogous to the one in the work of WASSERMANN (2020) used for the linear elastic case, it was kept for the introduction of the heat conduction model in section 2.3.1, with slight modification at the notion of boundary condition domains and the notation of the gradient operation, to be consistent with the notation used by WASSERMANN (2020).

2.3.1 The model for heat conduction

For examining the thermal behaviour of the 3D printed wall models, the linear, steady state heat equation was used. The *primary variable* for the model of heat conduction, as introduced by KRYSL (2010), is temperature. The temperature scalar field over a domain Ω can be written as

$$T(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega. \tag{2.1}$$

The conduction of heat is governed by the changes in the temperature field; therefore, the so called *kinematic equation* for the problem is solely the gradient of this field

$$\mathbf{g} = \nabla T\,. \tag{2.2}$$

Heat fluxes then can be determined based on the temperature gradient, with the help of the *constitutive equation*

$$\mathbf{q} = -\kappa \mathbf{g}, \tag{2.3}$$

where κ is the conductivity matrix of the material, and for the current case the material is modelled as thermally isotropic

$$\kappa = \begin{bmatrix} \kappa & 0 & 0 \\ 0 & \kappa & 0 \\ 0 & 0 & \kappa \end{bmatrix}, \tag{2.4}$$

with the conductivity κ needed as the only material parameter to define the model. The *equilibrium equation* for describing the steady state balance between heat fluxes ($\mathbf{q}$) and heat sources (Q) can be written as

$$-div\mathbf{q} + Q = 0\,. \tag{2.5}$$

Finally, the boundary value problem (BVP) as a model for the heat conduction in terms of the primary variable, with the Dirichlet and Neumann boundary conditions (illustrated in fig. 2.9) stated, can be formulated as

$$\begin{aligned} div(\kappa \nabla T) + Q &= 0\,, \\ T &= \bar{T} \quad \forall \mathbf{x} \in \Gamma_D\,, \\ \mathbf{q} \cdot \mathbf{n} &= \bar{q}_n \quad \forall \mathbf{x} \in \Gamma_N\,. \end{aligned} \tag{2.6}$$

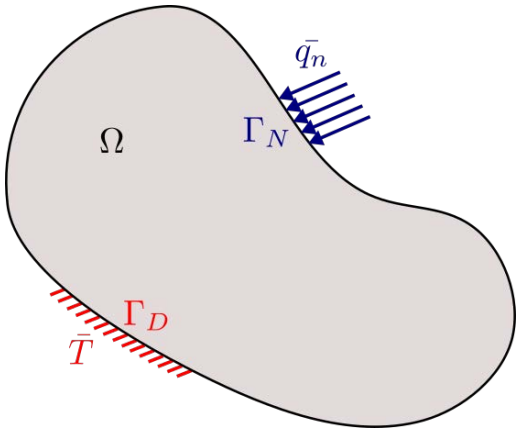


Figure 2.9: BVP for heat conduction

2.3.2 The model for elasticity

A detailed derivation of the model for linear elasticity, briefly introduced here, can be read in the work of WASSERMANN (2020). Here only the concepts of that work that are most relevant for the current application will be reviewed. The primary variable of the problem is the vector field of displacements defined over the computational domain Ω

$$\mathbf{u}(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega. \tag{2.7}$$

The linearized strains from the displacement field can be determined using the *kinamatic equation* of linear elasticity

$$\varepsilon = \frac{1}{2}\left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T\right). \tag{2.8}$$

With the help of the elasticity tensor, $\mathbf{C}$, the *constitutive equation* determines the connection between the strains and stresses

$$\sigma = \mathbf{C} : \varepsilon. \tag{2.9}$$

Finally, the *equilibrium equation* of linear elasticity describing the balance of stresses (σ) and body loads ($\mathbf{b}$) reads as

$$div\sigma + \mathbf{b} = \mathbf{0}. \tag{2.10}$$

With the help of the *kinematic* and *constitutive equations*, the above expression can be also rewritten in terms of the primary variable $\mathbf{u}$. Adding the boundary conditions (depicted in fig. 2.10), the final boundary value problem (BVP) reads as

$$\begin{aligned} div\left(\mathbf{C} : \frac{1}{2}\left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T\right)\right) + \mathbf{b} &= \mathbf{0}, \\ \mathbf{u} &= \bar{\mathbf{u}} \quad \forall \mathbf{x} \in \Gamma_D, \\ \mathbf{t} = \mathbf{n}\sigma &= \bar{\mathbf{t}} \quad \forall \mathbf{x} \in \Gamma_N. \end{aligned} \tag{2.11}$$

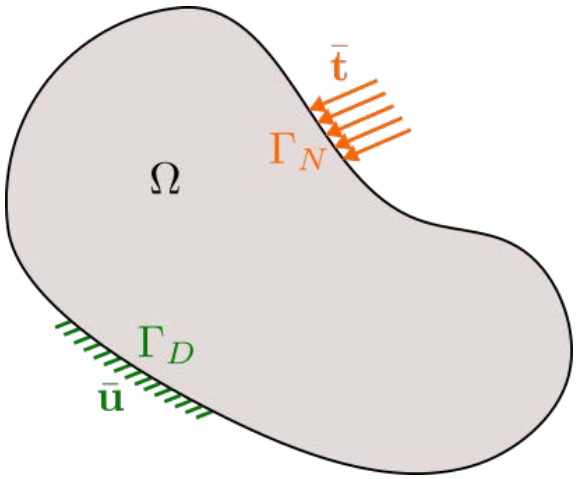


Figure 2.10: BVP for elasticity

2.3.3 On the structure of the equations of the applied models

Looking back at the introduction of the idealized models of heat conduction (section 2.3.1) and linear elasticity (section 2.3.2), a general pattern can be observed regarding the construction of the final boundary value problem. The *equilibrium equation*, defining the balance of some physical quantities, can be traced back to the primary variable with the help of the *constitutive equation* and *kinematic equation*, ending up with a model where the sole unknown is the primary variable. The Dirichlet and Neumann boundary conditions then can be applied on this final model. The components of the heat conduction and linear elasticity models according to this pattern (described also by KRYSL (2010)) are collected in table 2.1.

This structure is not unique to these two equations, but can be observed in several other mathematical models; therefore, providing a general framework for defining mathematical idealizations for physical phenomena. This fact is heavily relied on in the implementation of AdhoC++ (CMS, 2022) for providing a highly modular way of setting up mathematical models for different problems.

Table 2.1: Sub components of equations describing relevant physical phenomena for the current application

Math. model component	Elasticity	Heat conduction
Primary variable	$\mathbf{u}(\mathbf{x},t)$	$T(\mathbf{x},t)$
Kinematic equation	$\varepsilon = \frac{1}{2}(\nabla\mathbf{u} + (\nabla\mathbf{u})^T)$	$\mathbf{g} = \nabla T$
Constitutive equation	$\sigma = \mathbf{C} : \varepsilon$	$\mathbf{q} = -\kappa\mathbf{g}$
Equilibrium equation	$div\sigma + \mathbf{b} = \mathbf{0}$	$-div\mathbf{q} + Q = 0$
Dirichlet BC	$\mathbf{u} = \bar{\mathbf{u}} \quad \forall\mathbf{x} \in \Gamma_D$	$T = \bar{T} \quad \forall\mathbf{x} \in \Gamma_D$
Neumann BC	$\mathbf{t} = \mathbf{n}\sigma = \bar{\mathbf{t}} \quad \forall\mathbf{x} \in \Gamma_N$	$\mathbf{q}\cdot\mathbf{n} = \bar{q}_n \quad \forall\mathbf{x} \in \Gamma_N$

2.4 Model discretization

As it was already stated in section 2.1, to be able to utilize computers for the solution of mathematical models, the continuous formulation of the problem at hand needs to be discretized. To this end, several procedures exist of which the Finite Element Method (FEM), relevant for this project, is among the most prominent ones. In the AdhoC++ (CMS, 2022) framework, a certain variation of FEM, the so called Finite Cell Method (FCM) (PARVIZIAN et al., 2007) is implemented for the discretization of continuous models. As it will be briefly addressed in section 2.4.2, this method is a higher order version of FEM, where the physical domain is embedded into and meshed together with a fictitious domain, which allows for creating finite elements in a structured grid.

As the implementation of the CSG geometric library for FCM in AdhoC++ (CMS, 2022) was done in the scope of the the PhD thesis of Benjamin Wassermann (WASSERMANN, 2020), the entirety of this thesis heavily relies on his work. Due to this fact, and to be more consistent with the previous section on the mathematical model of linear elasticity (section 2.3.2), his introduction will be followed to the topics of FEM and FCM as well.

Here, first the general concept of the Finite Element Method will be briefly introduced, followed by addressing the necessary modifications to it for the Finite Cell Method. The discussion of the two concepts will be done based on the equations of linear elasticity (as it was addressed by WASSERMANN (2020)). For an introduction to the Finite Element Method (FEM) for heat conduction, the reader is referred again to KRYSL (2010).

2.4.1 The Finite Element Method

The derivation of the discretization for the linear elastic model starts with the so called *strong* or *differential form*, as the boundary value problem was introduced in section 2.3.2,

$$\begin{aligned} div(\sigma) + \mathbf{b} &= \mathbf{0}, \\ \mathbf{u} &= \bar{\mathbf{u}} \quad \forall \mathbf{x} \in \Gamma_D, \\ \mathbf{t} = \mathbf{n}\sigma &= \bar{\mathbf{t}} \quad \forall \mathbf{x} \in \Gamma_N, \end{aligned} \tag{2.12}$$

where σ can be expressed in terms of $\mathbf{u}$ with the help of the *constitutive* and *kinematic equations* in a way it was described in section 2.3.2.

This formulation of the model is, however, hardly ever solvable analytically for practical problems. This necessitates the approximate solution via a discretized model in the first place. But even for an approximating function, this formulation poses too strong criteria on differentiability. To overcome this, an equivalent *integral* or *weak form* of the model can be derived. As it was done by WASSERMANN (2020) as well, one can use the fact that the zero function is orthogonal to any other function, therefore the *strong form's* scalar product with an arbitrary test function $\mathbf{v}$ always yields zero

$$\int_{\Omega} (div\sigma + \mathbf{b}) \cdot \mathbf{v} d\Omega = \int_{\Omega} div\sigma \cdot \mathbf{v} d\Omega + \int_{\Omega} \mathbf{b} \cdot \mathbf{v} d\Omega = 0. \tag{2.13}$$

Applying the product rule in the integrand of the first integral of the summation and then Gauss's theorem on the integral itself, as detailed by WASSERMANN (2020), one arrives to the following expression

$$-\int_{\Omega} \sigma : \nabla \mathbf{v} d\Omega + \int_{\Omega} \mathbf{b} \cdot \mathbf{v} d\Omega + \oint_{\Gamma} \sigma \mathbf{v} \cdot \mathbf{n} d\Gamma = 0. \tag{2.14}$$

From which - by utilizing the symmetry of σ, the *kinematic equation* and the Neumann boundary condition as described by WASSERMANN (2020) - the final, *weak form* of the problem can be expressed

$$\underbrace{-\int_{\Omega} \sigma : \delta\varepsilon d\Omega}_{-\delta W_{int}} + \underbrace{\int_{\Omega} \mathbf{b} \cdot \mathbf{v} d\Omega + \oint_{\Gamma_N} \bar{\mathbf{t}} \cdot \mathbf{v} d\Gamma_N}_{\delta W_{ext}} = 0, \tag{2.15}$$

with $\mathbf{u}$ (on which σ depends) assumed to satisfy the Dirichlet boundary conditions (eq. (2.16)), while $\mathbf{v}$ is taken to satisfy homogeneous boundary conditions (eq. (2.17)) on Γ_D. With this choice of $\mathbf{v}$, the surface integral in eq. (2.14) becomes zero on Γ_D, leaving only the contribution of the Neumann boundary conditions on Γ_N as given in eq. (2.15).

$$\mathbf{u}(\mathbf{x}) = \bar{\mathbf{u}} \quad \forall \mathbf{x} \in \Gamma_D \tag{2.16}$$

$$\mathbf{v}(\mathbf{x}) = \mathbf{0} \quad \forall \mathbf{x} \in \Gamma_D \tag{2.17}$$

More detailed criteria on the spaces the above functions have to be taken from, are addressed by WASSERMANN (2020). The different terms in eq. (2.15) can be recognized as the expression for the *principal of virtual work* (HOLZAPFEL, 2000), which states that for a system in equilibrium, the internal and external *virtual work* equal

$$-\delta W_{int} + \delta W_{ext} = 0. \tag{2.18}$$

For the discretization of the weak form in eq. (2.15), the Bubnov-Galerkin approach was taken, as described by WASSERMANN (2020). In this procedure, the approximation functions for the solution field and the test function are taken from the same function space, where the functions themselves are defined as the combination of a chosen set of *shape functions* and their corresponding *degrees of freedom* as

$$\mathbf{u}(\mathbf{x}) \approx \mathbf{u}^h(\mathbf{x}) = \mathbf{N}(\mathbf{x})\tilde{\mathbf{u}} \quad \text{and} \tag{2.19}$$

$$\mathbf{v}(\mathbf{x}) \approx \mathbf{v}^h(\mathbf{x}) = \mathbf{N}(\mathbf{x})\tilde{\mathbf{v}}, \tag{2.20}$$

where $\mathbf{N}(\mathbf{x})$ is a matrix of shape functions and $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{v}}$ are vectors containing the corresponding degrees of freedom. For example for $\mathbf{u}(\mathbf{x}) = [u(\mathbf{x}), v(\mathbf{x}), w(\mathbf{x})]$ in the case of eq. (2.19), the expression would expand to

$$\mathbf{N}(\mathbf{x})\tilde{\mathbf{u}} = \begin{bmatrix} N_1(\mathbf{x}) & 0 & 0 & \cdots & N_n(\mathbf{x}) & 0 & 0 \\ 0 & N_1(\mathbf{x}) & 0 & \cdots & 0 & N_n(\mathbf{x}) & 0 \\ 0 & 0 & N_1(\mathbf{x}) & \cdots & 0 & 0 & N_n(\mathbf{x}) \end{bmatrix} \begin{bmatrix} \tilde{u}_1 \\ \tilde{v}_1 \\ \tilde{w}_1 \\ \vdots \\ \tilde{u}_n \\ \tilde{v}_n \\ \tilde{w}_n \end{bmatrix}. \tag{2.21}$$

With these functions, the approximations for the *internal* and *external virtual work* can be formulated as

$$\delta W^h_{int} = \tilde{\mathbf{v}} \int_\Omega \mathbf{B}^T \mathbf{C} \mathbf{B} d\Omega \tilde{\mathbf{u}} = \tilde{\mathbf{v}} \mathbf{K} \tilde{\mathbf{u}} \quad \text{and} \tag{2.22}$$

$$\delta W^h_{ext} = \tilde{\mathbf{v}} \int_\Omega \mathbf{N}^T \mathbf{b} d\Omega + \tilde{\mathbf{v}} \int_{\Gamma_N} \mathbf{N}^T d\Gamma_N = \mathbf{f_b} + \mathbf{f}_N = \mathbf{f}, \tag{2.23}$$

where $\mathbf{C}$ is the elasticity tensor from eq. (2.9) and

$$\mathbf{B}(\mathbf{x}) = \mathbf{L}\mathbf{u}(\mathbf{x}) = \mathbf{L}\mathbf{N}(\mathbf{x})\tilde{\mathbf{u}}, \tag{2.24}$$

with $\mathbf{L}$ being the differential operator as described by WASSERMANN (2020).

In eq. (2.22), $\mathbf{K}$ is called the global stiffness matrix of the discretized model, while $\mathbf{f}$ in eq. (2.23) is the global load vector, composed of the vectors of body loads $\mathbf{f_b}$ and external loads $\mathbf{f}_N$.
Factoring out $\tilde{\mathbf{v}}$, the expression for the *principal of virtual work* becomes

$$\tilde{\mathbf{v}} \left(\mathbf{K}\tilde{\mathbf{u}} + \mathbf{f} \right) = 0. \tag{2.25}$$

Since $\mathbf{v}^h(\mathbf{x})$ was taken as arbitrary and therfore so was $\tilde{\mathbf{v}}$, to have the equation hold all the time, the expression in the brackets must be zero

$$\mathbf{K}\tilde{\mathbf{u}} + \mathbf{f} = 0. \tag{2.26}$$

Which is already the standard from (see FELIPPA, 2004) of the linear system of equations representing the discretized model for linear elasticity.

By chosing shape functions ($\mathbf{N}(\mathbf{x})$) with compact support on different subdomains of Ω, usually referred to as *elements*, the integration of $\mathbf{K}$ and $\mathbf{f}$ can be carried out piece-wise on these subdomains and the results assembled into $\mathbf{K}$ and $\mathbf{f}$ as stated by WASSERMANN (2020) and detailed by FELIPPA (2004).

2.4.2 The Finite Cell Method

As mentioned already at the beginning of section 2.4, the Finite Cell Method (FCM) (PARVIZIAN et al., 2007) is a higher order version of the Finite Element Method, with a fictitious domain (Ω_{fict}) approach (depicted on fig. 2.11) that extends the original physical domain (Ω_{phy}) to make the combined computational domain (Ω) easily meshable with a structured grid of *finite cells*.

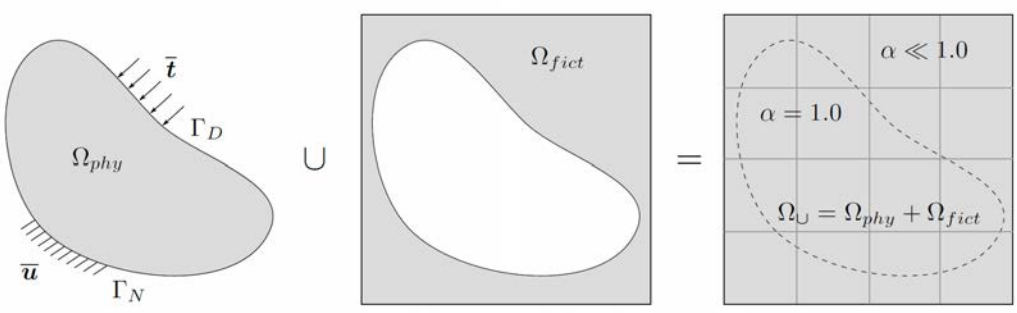


Figure 2.11: The fictitious domain approach as illustrated in (WASSERMANN, 2020)

In the AdhoC++ (CMS, 2022) implementation, for the formulation of the element (*cell*), hierarchical shape functions were taken. As described by WASSERMANN (2020), the used integrated Legendre polynomials, N_i, defined on the interval $\xi \in [-1.0, 1.0]$ can be written as

$$N_1(\xi) = \frac{1}{2}(1-\xi), \tag{2.27}$$

$$N_2(\xi) = \frac{1}{2}(1+\xi), \tag{2.28}$$

$$N_i(\xi) = \Phi_{i-1}(\xi), \;\; i = 3, ..., p+1, \tag{2.29}$$

with

$$\Phi_i(\xi) = \sqrt{\frac{2j-1}{2}} \int_{-1}^{\xi} L_{j-1}(x)dx = \frac{1}{4j-2}(L_j(\xi) - L_{j-2}(\xi)), \;\; j = 2, 3, ... \tag{2.30}$$

With the above shape functions defined, the weak formulation in eq. (2.15), using the notation from eq. (2.22) and eq. (2.23), can be given as stated by WASSERMANN (2020)

$$\int_{\Omega} [\mathbf{Lv}]^T \alpha \mathbf{C} [\mathbf{Lu}] d\Omega = \int_{\Omega} \mathbf{v}^T \mathbf{b} d\Omega + \int_{\Gamma_N} \mathbf{v}^T \bar{\mathbf{t}} d\Gamma_N. \tag{2.31}$$

with $\alpha(\mathbf{x})$ being the so called *indicator function* (WASSERMANN, 2020) defined as

$$\alpha(\mathbf{x}) = \begin{cases} 1 & \forall x \in \Omega_{phy} \\ 10^{-q} & \forall x \in \Omega_{fict} \end{cases}. \tag{2.32}$$

This *indicator function* weights the material matrix $\mathbf{C}$ and the body forces $\mathbf{b}$ based on whether its argument is contained by the geometry. The *indicator function* can be constructed in practice with the help of e.g., one of the implicit geometric descriptions introduced in section 2.2 to decide about the inclusion of a point in Ω_{phy}.

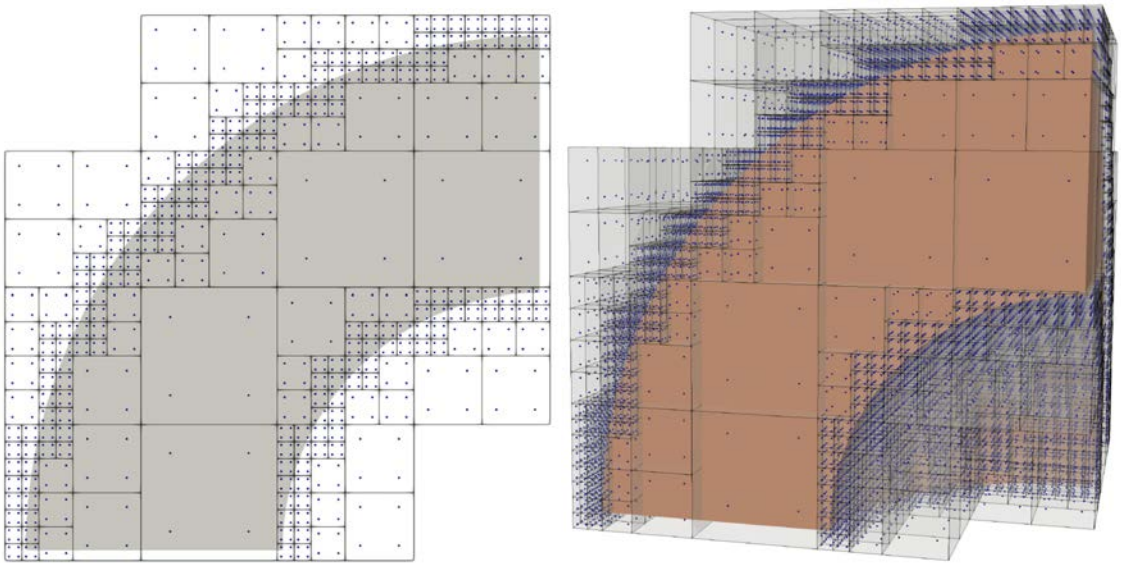
As it is stated by WASSERMANN (2020), if $q = \infty$ at the limit in eq. (2.32), the original FEM formulation of the weak form is recovered. However, forcing $\alpha = 0$, usually renders the system describing the model ill-conditioned (for explanations of the causes see WASSERMANN (2020)). In practice, taking a "large enough" value for q can already ensure accurate results (WASSERMANN, 2020).

2.4.3 Integration methods in FCM

Looking at eq. (2.31), one can observe one of the main characteristics of the Finite Cell Method, namely, that the geometry is resolved during the integration phase. This makes mesh generation easier, since there is no need for generating meshes conforming the boundary of complex geometries. However, to achieve reliable results, very accurate numerical integration of the element stiffness matrices is necessary. Due to the discontinuous nature of the *indicator function* $\alpha(\mathbf{x})$ from eq. (2.32), for the FCM, special quadrature rules were developed, from which the two addressed by WASSERMANN (2020) were used and tested (see section 4.3) within the scope of this thesis as well. Namely, composed integration and moment fitting. Therefore, the background of these will be shortly revised here. One must note that several other methodologies have been developed, with each with its own merits. For example, integration with the smart octree, an improved version of the composed integration discussed here (KUDELA et al., 2016).

Composed integration

The main idea behind composed integration, as described by WASSERMANN (2020), is to distribute more integration points around the boundary of the geometry with the help of space trees ($\mathbf{T}_{int}$). With a sufficiently refined integration tree, $\mathbf{T}_{int}$, as depicted on fig. 2.12, taken from WASSERMANN (2020), the higher number of integration leaves ($\mathbf{c}_{int}$) at the boundary can provide a dense enough cluster of points to sufficiently resolve the effects of the jump due to the discontinuity in $\alpha(\mathbf{x})$. Usually, in each leaf the same amount of integration points are distributed in each dimension (WASSERMANN, 2020). For example, on fig. 2.12, in both cases 2 points were added in every dimension.



(a) Integration points distributed with a quadtree (b) Integration points distributed with an octree

Figure 2.12: Distribution of integration points with the help of a quadtree fig. 2.12a in 2D and octree fig. 2.12b in 3D on basic geometries (WASSERMANN, 2020)

Once the points are properly distributed, the standard Gauss quadrature can be used to carry out the integration. In general (with the notation of WASSERMANN (2020)), it can be formulated as

$$\int_a^b f(x)dx \approx \sum_{j=1}^{n_{c_{int}}} \sum_{i=1}^{n_{GP}} w_i \cdot f(x(t_i)) \cdot det\mathbf{J}_t(t_i) \cdot det\mathbf{J}_{c_{int,j}}(t_i), \tag{2.33}$$

where the last two determinants account for the mappings from the local coordinate spaces of the leaves to the global system.

Moment fitting

The second approach addressed here is the method of moment fitting (HUBRICH et al., 2017). In this integration scheme (as explained by WASSERMANN (2020) as well) one attempts to find an ideal distribution of integration points and their weights on each of the finite cells of the mesh by solving the equation

$$\sum_{i=1}^{n_{GP}} N_j(\mathbf{x}_i) w_i = \int_{\Omega_{cell_{phy}}} N_j(\mathbf{x}) d\Omega \tag{2.34}$$

with N_j being the m independent basis function defined on the domain of the cell (Ω_{cell}). The problem can be reformulated as a system of equations

$$\begin{bmatrix} N_1(\mathbf{x}_i) & \cdots & N_1(\mathbf{x}_{n_{GP}}) \\ \vdots & \ddots & \vdots \\ N_m(\mathbf{x}_i) & \cdots & N_m(\mathbf{x}_{n_{GP}}) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_{n_{GP}} \end{bmatrix} = \mathbf{A}\mathbf{w} = \mathbf{b} = \begin{bmatrix} \int_{\Omega_{cell_{phy}}} N_1(\mathbf{x}) d\Omega \\ \vdots \\ \int_{\Omega_{cell_{phy}}} N_m(\mathbf{x}) d\Omega \end{bmatrix} \tag{2.35}$$

with the so called moments at the right hand side. In some cases these integrals can be calculated symbolically as it was done by HUBRICH et al. (2017) with the help of the *Wolfram Mathematica* language. But in a more general approach, these volume integrals can be numerically determined with a standard or a smart octree (KUDELA et al., 2016), as mentioned by WASSERMANN (2020), since their evaluation with these methods is still much cheaper computationally then applying them directly on the cell itself. The hope is by determining the moments with these methods, that one can come up with a scheme, which provides a less computation intensive way to carry out the integration of a cell's stiffness and force contributions.

Furthermore, as stated by WASSERMANN (2020), with the help of the divergence theorem, the volume integrals of the moments can be turned into surface integrals, which can reduce the cost of integrating them even more. The conversion is done as

$$\int \Omega_{cell_{phy}} N_j(\mathbf{x}) d\Omega = \int_{\Gamma_{cell}} \mathbf{g}_i(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) d\Omega, \tag{2.36}$$

with $\mathbf{n}(\mathbf{x})$ being the boundary normal and $\mathbf{g}_i(\mathbf{x})$ standing for the antiderivatives, which can be computed as

$$\mathbf{g}_i = \frac{1}{3} \begin{bmatrix} \int N_i dx \\ \int N_i dy \\ \int N_i dz \end{bmatrix}. \tag{2.37}$$

2.4.4 Penalty boundary conditions

Boundary conditions in the current implementation are applied using the penalty approach and STL files for the surface description. This method (as introduced by FELIPPA (2004) but described by WASSERMANN (2020) as well) leads to a modified system of equations for modeling the problem by minimizing the following extended energy functional (WASSERMANN, 2020)

$$\Pi_P^h = \frac{1}{2}\tilde{\mathbf{u}}^T \mathbf{K} \tilde{\mathbf{u}} - \tilde{\mathbf{u}}^T \mathbf{f} + \frac{1}{2}\beta ||\mathbf{A}\tilde{u} - \mathbf{b}||^2 \longrightarrow \underset{\tilde{\mathbf{u}}}{min}. \tag{2.38}$$

The mentioned modified system of equations derived based on the above functional reads

$$\left(\mathbf{K} + \beta \mathbf{M}^P\right) \tilde{\mathbf{u}} = \mathbf{f} + \beta \mathbf{f}^P, \tag{2.39}$$

with

$$M_{ij}^P = \left[\mathbf{A}^T \mathbf{A}\right]_{ij} = \int_{\Gamma_D} N_i N_j d\Gamma, \tag{2.40}$$

$$f_i^P = \left[\mathbf{A}^T \mathbf{b}\right]_i = \int_{\Gamma_D} N_i \tilde{\mathbf{u}} d\Gamma. \tag{2.41}$$

As described by FELIPPA (2004) and WASSERMANN (2020), in the limit of $\beta \longrightarrow \infty$ the penalty method fulfills the Dirichlet boundary conditions perfectly. However, a too large value taken for β can easily render the system ill-conditioned. Specially in the the context of FCM, care must be taken not to chose the penalty factor too high, since with the the indicator function $\alpha(\mathbf{x})$, small entries are already introduced into the system matrix. One reliable option to mitigate the effects of this on the solution is the usage of direct solvers. While the iterative approaches usually fail to converge for ill-conditioned systems, direct solvers are less affected. For the current project the Pardiso solver from the Intel® oneAPI Math Kernel Library was used, as detailed in section 3.1. This, together with reasonably chosen penalty values, ensured that boundary conditions could be enforced accurately enough, while still obtaining reliable simulation results.

2.5 Goals of the implementation

With all these theoretical knowledge revised, the goals for the implementation can be made more specific. As already stated in section 1.3, for closing the optimization loop defined in the scope of the FIM framework proposed by SLEPICKA (2021), first a print path-based generation of a volumetric model is needed.

For this purpose, the Constructive Solid Geometry (CSG) approach, described in section 2.2.3 will be used, since on the one hand, it follows the way of the fabrication process quite neatly. Sweeping a cross section along a path for csg primitives is analogous to 3D printing layers along a given print path, which makes the geometry generation intuitive for the current application case. On the other hand, as it will be addressed in section 3.4.3 and section 3.4.4, some CSG primitives are capable of making use of the vertically periodic nature of the wall geometry and speed up the point inclusion tests on the geometry.

For the analysis, the models of linear heat conduction (section 2.3.1) and linear elasticity (section 2.3.2) will be used, discretized by the Finite Cell Method (FCM) implemented in the AdhoC++ (CMS, 2022) framework. For applying the necessary Dirichlet and Neumann boundary condition, the penalty approach (section 2.4.4) will be utilized, with STL files (section 2.2.1) providing the relevant surface description for it. In the following chapter (chapter 3), it will be addressed, how these different components got combined into a single tool for the analysis of 3D printed wall designs.

Chapter 3

Implementation of the simulation tool in the AdhoC++ framework

As it was already mentioned in section 1.3, the goal of this thesis is to enable numerical simulations on fabrication information based geometries. This chapter is dealing with the implementation of a tool enabling that, and is organized as follows: section 3.1 offers a brief overview about how the AdhoC++ (CMS, 2022) framework implements the modeling steps addressed in section 2.1. In section 3.2 the process of the geometry generation is discussed, including how the data file is parsed (section 3.3), and how this information then gets turned into a volumetric model with the help of sweep operations (section 3.4.1) and the building of a CSG tree (section 3.4.2). As the performance of the point inclusion tests on the suggested model is a fundamental issue, section 3.4.3 and section 3.4.4 focus on improving the initial geometric representation by making use of the vertical periodicity observable on 3D printed walls. In section 3.5, the way boundary conditions can be defined is addressed. Finally, in section 3.6 the simulation outputs will be discussed. To illustrate the concepts worked on throughout this chapter and the following one (chapter 4), a simple 3D printed straight wall geometry will be used as depicted on fig. 3.1.

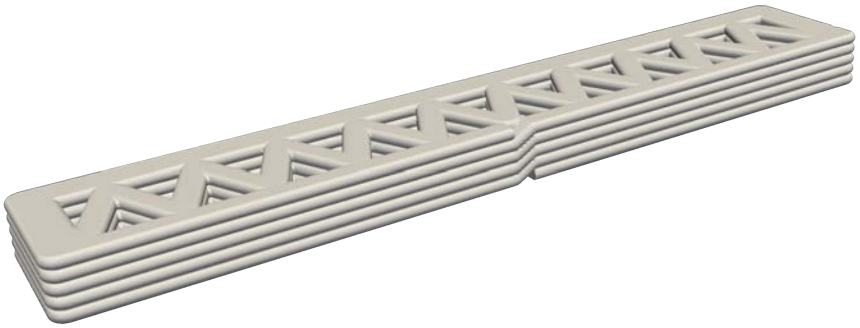


Figure 3.1: The wall geometry used to illustrate the concepts addressed in this chapter

3.1 Modeling the problem in the AdhoC++ framework

In the following, the workflow for setting up a simulation in AdhoC++ (CMS, 2022) will be briefly overviewed, ordering the individual components into the main modeling steps defined by FELIPPA (2004). On fig. 3.2, the shown elements of a simulation model are color-coded according to these general steps already seen on fig. 2.1 in section 2.1.

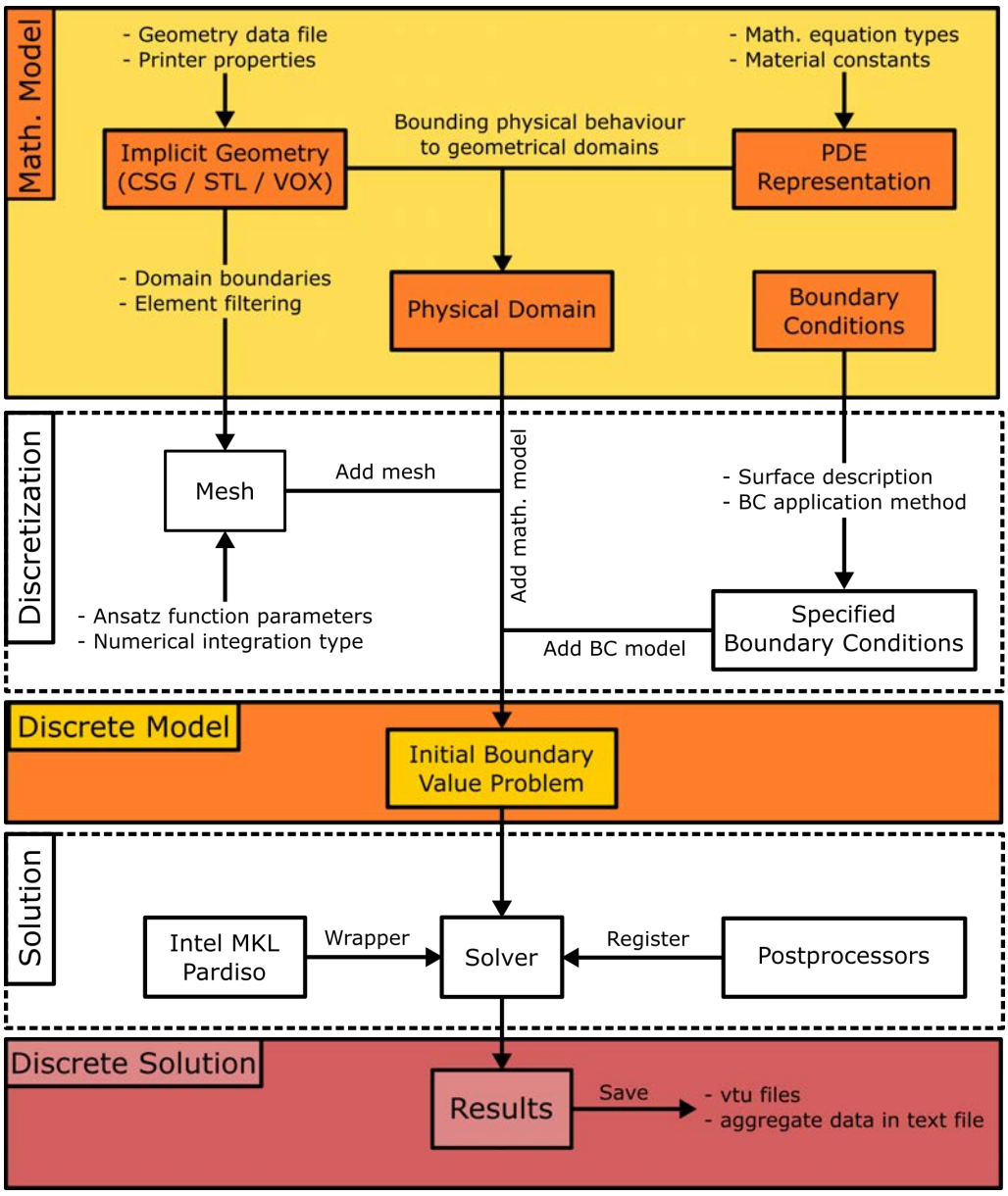


Figure 3.2: The modeling process implemented in the AdhoC++ (CMS, 2022) framework

First, the two idealization steps - addressed in section 2.2 and section 2.3 - have to be done by choosing a way to represent the analyzed object with an implicit geometric description, and defining a partial differential equation governing the physical behaviour of the model. These two definitions then get combined into a single idealized physical domain, bounding the physical model to the extent of the geometry.

Boundary conditions at this point of the modeling are only conceptually present. The depiction of them in the "mathematical model" part of fig. 3.2 serves merely to be consistent with the underlying classification of the different steps, since fundamentally, boundary conditions are also continuously defined mathematical idealizations of the interaction of the object with its surroundings (see section 2.1).

Before the boundary conditions could be applied though, the computational domain needs to be discretized by a finite cell mesh, with an appropriate ansatz space chosen, and a numerical integration strategy defined. After the meshing of the entirety of the domain (physical and fictitious as described in section 2.4.2) is done, the elements, which are completely outside of the wall geometry get filtered out and deactivated, as detailed by WASSERMANN (2020) and WASSERMANN et al. (2017).

For defining boundary conditions in AdhoC++ (CMS, 2022) the type of them, a surface description and an application strategy must be provided. Within the scope of this thesis, STL files (section 2.2.1) for the surface description, and the penalty method (section 2.4.4) as application strategy was used.

After the mesh was set up and the boundary conditions defined, they can be combined with the physical domain into an initial boundary value problem, which corresponds to the discrete model in the modeling process.

This discrete representation then can be processed with the help of the Pardiso solver from the Intel® oneAPI Math Kernel Library, for which a wrapper interface was implemented in the AdhoC++ (CMS, 2022) project. During the solution phase, all the registered postprocessing subroutines get executed as well.

Results of the postprocessors finally then can be analyzed with the help of the software ParaView (PARAVIEW DEVELOPERS, 2022) . For the FIM optimization loop, two aggregate values were also defined, being the U-Value and the average vertical displacement. The definition of these and more details about the procedure of outputting results can be found in section 3.6.

3.2 Overview of the geometry generation

The AdhoC++ (CMS, 2022) workflow introduced in the previous section is implemented in a manner that it works with any geometric representation which provides a point inclusion test. Therefore, adding new simulation modules to the software very often comes down to providing an algorithm to generate the application specific geometry. This case is no exception and the main focus of the work was on coding a tool, which is able to create geometries of 3D printed walls, using the print path data files generated by the FIM model.

The implementation for generating the 3D wall geometries consists of three phases, as depicted on fig. 3.3. The process starts with the data files provided by a FIM model, which contains information about the path the printing machine's nozzle should follow. This file then gets parsed into a print path, already made up of curve objects defined within the AdhoC++ (CMS, 2022) framework (section 3.3). Finally, with the help of a sweeping operation using a pre-defined cross section, they get turned into primitive volumes, which then get combined together into the final geometry using the Constructive Solid Geometry approach (section 3.4.1 and section 3.4.2).

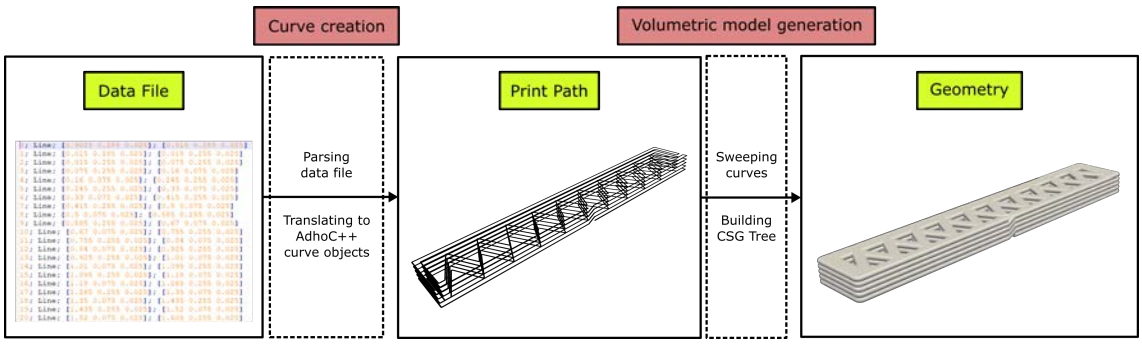


Figure 3.3: The geometry generation process

After conducting some experiments with this way of generating the volumetric model, it was found that point inclusion tests for complex models can take a significant amount of time due to the size of the CSG binary tree and the number of point inclusion evaluations this fact results in in practice. To improve on this situation, a second version of the algorithm made use of the vertically repeating geometry of a printed wall, by defining vertically periodic domains with sweeps created from the bottom layer curves as unit cells, as described in section 3.4.3 and section 3.4.4. As a result, the final CSG domain is generated with the periodic primitives, offering point inclusion evaluations with much better performance.

3.3 Creating the print path

The creation of a print path consist of two stages. In the first one, the data file gets parsed into curve objects defined within the AdhoC++ (CMS, 2022) framework, and stored in a single array on which other algorithms can loop through later on. In the second step, another subroutine first identifies kinks in the continuous print path defined by the aforementioned array of curves. After such a curve connection is found, it determines an arc tangentially connected to both curves, which then get trimmed at the appropriate points and the rounding arc gets inserted into the array, between the two consecutive curves the rounding operation was called on. In the long term, this second rounding phase will not be necessary since the FIM model is planned to be extended in a way that all paths it generates are C2 continuous. Yet, the proposed way of trimming the kinks in the model is found to be necessary for now, as discussed in section 3.3.2.

3.3.1 Parsing the data file into curve objects

The structure of the data files containing the print path information, coming with a ".pp" extension, is quite simple as it can be seen on the basic example given in fig. 3.4. It describes the path the nozzle of the printer has to take to create the geometry. This means that the path given runs always at the top of a printed layer. Every line of the file contains information about a single curve of the entire path, which can be a line, an arc or a spline.

The small number of options for curve types are due to the fact that on the one hand, with only these three types of curves already highly complex shapes can be generated, as it can be seen later on in section 5.1.1 and section 5.1.2. On the other hand, more importantly, these are the curve types the printing robot can interpret and traverse with its nozzle.

```
0; Arc; [0.0 0.0 0.025]; [0 0 1.0]; 1.985; 45.6494; 89.567
1; Line; [0.015 1.9849 0.025]; [0.015 1.9549 0.025]
2; Arc; [0.0 0.0 0.025]; [0 0 -1.0]; 1.955; 270.433; 272.1982
3; Line; [0.075 1.9536 0.025]; [0.2003 1.7637 0.025]
4; Line; [0.2003 1.7637 0.025]; [0.3649 1.9206 0.025]
5; Line; [0.3649 1.9206 0.025]; [0.4606 1.7142 0.025]
6; Spline; [0.0927 0.7415 0.025]; [0.0872 0.742 0.025]; [0.0615 0.7438 0.025]
7; Line; [0.015 0.7751 0.05]; [0.015 0.7451 0.05]
8; Spline; [0.0927 0.7415 0.05]; [0.0872 0.742 0.05]; [0.0615 0.7438 0.05]
```

Figure 3.4: Section of a data file containing print path curves according to the ".pp" input file format

To be able to examine the input file data types better, their structure was depicted in a more generic manner in the bottom box of fig. 3.5. As it can be seen, each type starts with the number of the specific line, followed by the type of the curve it describes. This is the point, where the first decision is made by the parser algorithm. Given this data type, it can decide about which of the available AdhoC++ (CMS, 2022) curve objects (relevant ones depicted in the upper box of fig. 3.5) has to be constructed.

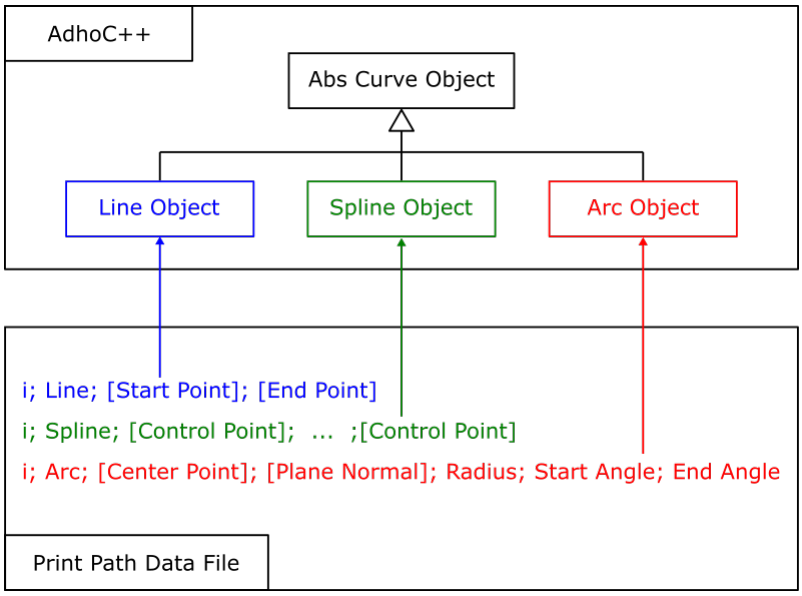


Figure 3.5: Connection between the data file types and the AdhoC++ (CMS, 2022) curve objects

In AdhoC++ (CMS, 2022) , all of the depicted curve objects inherit from a certain abstract curve type, what insures that all of them have the following properties (among others), which is going to be more relevant in section 3.3.2, but comes natural state them upon talking about their creation:

1. They are parametric curves with the parameter running from -1.0 to 1.0, giving the start and end points of the curve at these values respectively.
2. They have a method, which can return a point of the curve at a provided parameter value in the [-1.0, 1.0] range.
3. They have a method, which can return a local system of basis vectors at a provided parameter value in the [-1.0, 1.0] range (e.g. the Frenet-basis at the specific point, but other basis types can be defined by the user as well).

Furthermore, all the above three object types posses constructors corresponding to their respective types in the data file. This means that a Line object can be constructed directly from the start and end point coordinates of the Line file data type. A Spline object can be generated directly from the provided list of control points, using the assumption made by the file format that all splines are cubic and are defined with an open, uniform knot vector with inner knot multiplicities considered to be one. Finally, an Arc curve object also can be directly created from center point, plane normal, radius and starting and ending angles. However, a conversion to radians from the angles given in degrees in the data file has to be made.

As the geometric information gets parsed and turned into curves of different types, like described before, each of the newly generated curves gets added to an array as depicted in fig. 3.6. This array can be then later looped over during the generation of the volumetric geometry of the wall, as it will be introduced in section 3.4.

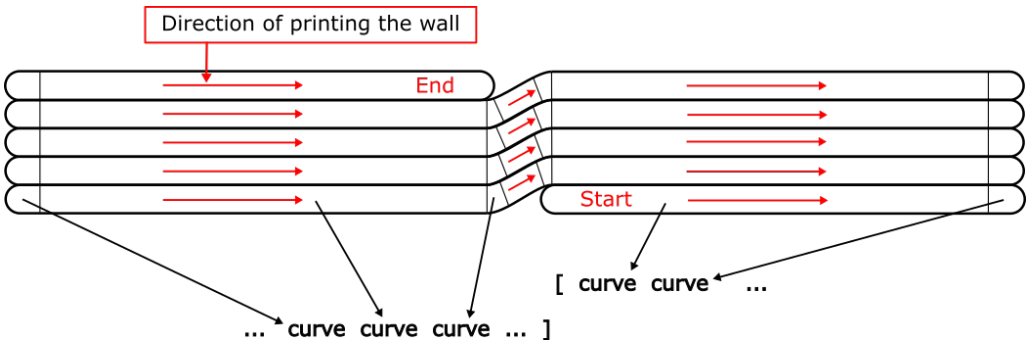


Figure 3.6: The print path data array for the curves

3.3.2 Rounding kinks in the print path

As it was mentioned already at the beginning of section 3.3, the lack of tangential connections of curves at certain points of the provided print path necessitates the insertion of a rounding arc. This must be done for several reasons. The first ones are technology related in nature. The printing machine is just simply unable to create perfectly sharp corners; therefore, a technical radius with a certain size at the kinks in the planned print path will be always present in the actually printed geometry. Furthermore, at certain components of the robot, there are acceleration limits set, so that damaging the machine can be avoided. The lack of higher order continuity at specific points in the path would lead to accelerations

exceeding these limits. Therefore, the controlling mechanism will slow down the nozzle of the printer, resulting in more material distributed near the problematic point and in a different shape, compared to the intended one. Since the goal of a FIM model is to take limitations of the construction technology into account during the design process, these issues have to be accounted for in the planning phase, and only corners with sufficient roundings designed. A further shortcoming related to the geometry generation is shown on fig. 3.7. The example geometry was created from a rectangular path without rounding at the corners. One can see, that without the tangential rounding curves - due to the nature of the sweeping operation - gaps at the corners of the geometry occur. This is clearly not something what accurately would represent the geometry ”as-planned”.

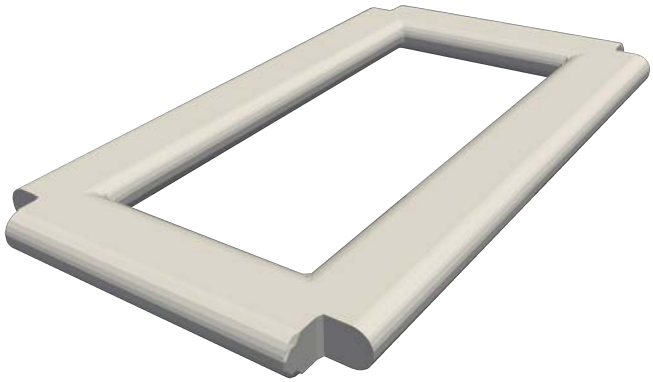


Figure 3.7: Geometry created from a print path without rounding

Introducing the algorithm

Due to all the aforementioned reasons, an algorithm to modify non-smooth connections had to be implemented. This was done by inserting a tangentially connected arc with a pre-defined rounding radius between the pair of curves of interest, and trimming the original curves at the connection points with the new arc. For such an algorithm, in the current application case the following requirements had to be made:

1. The algorithm should work in 2D and 3D as well.
2. It should provide a tangentially connected rounding arc for all possible pairs of curve types within the scope of the project (lines, arcs and splines), given that it is theoretically possible to find such an arc for a given setup.
3. It should not require an implementation of specific subroutines for each of the possible different type pairs.
4. It should not require a modification of the already implemented curve types.

A type pair agnostic behaviour of the algorithm was achieved by utilizing only the properties inherited from their parent class, introduced in section 3.3.1, namely:

1. They are parametric curves with a parameter range of [-1.0,1.0].
2. They have a method for returning a curve point at a given parameter value.
3. They have a method for returning a (tangential) local basis at a parameter value.

For the explanation of the steps of the implemented algorithm, it makes sense to start with a 2D example first, provided in fig. 3.8a. For each of the curves (curve DE and EF) it is possible (in accordance with the parent curve class properties listed earlier) to get a local basis at some given initial parameters with one basis being the tangent vector of the curves at those parameters.

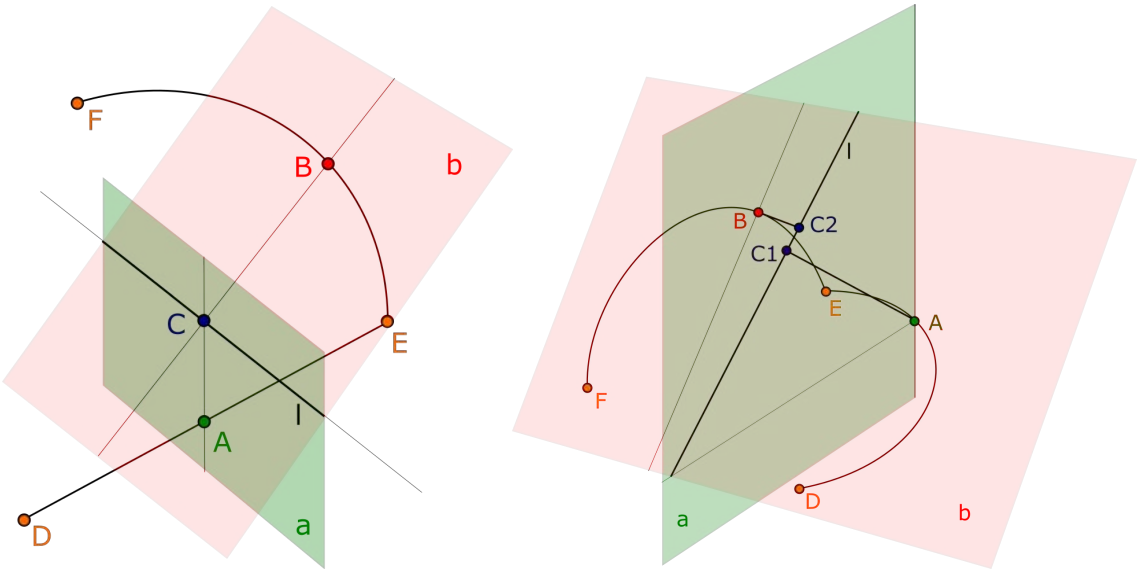
Having these local basis, one can create a plane for each curve (planes a and b in fig. 3.8a) passing through the current curve point (points A and B) and being perpendicular to the local tangent vector of the curve. This two planes (if they are not parallel, which is checked against) have a line of intersection (l) which must be perpendicular to the plane defined by the points D, E and F, since the two curves are co-planar and so are their tangent vectors. Therefore both of the planes having these tangents as normal vectors are perpendicular to the same plane and so is their intersection line. On this line l the closest point to A and B can be found. (The shortest distance being the length of the shortest section from all the sections connecting the intersection line l with A and B respectively. This is always the section perpendicular to the intersection line.) In 2D, this closest point (C) coincides for the two cases.

The closest point on the intersection line to A (and B) can be determined rather easily by using the approach suggested by KRUMM (2000). This algorithm (which in the end comes down to solving a 5x5 linear system) returns the closest point on the line of intersection of two planes to another user provided point.

If one takes the user input point as A (or B), the algorithm will return the desired point C. After this step, to get the rounding curve defined, it only has to be made sure, that the following two criteria holds:

$$\begin{aligned} |C(t,s) - A(t)| - r = 0, \\ |C(t,s) - B(s)| - r = 0, \end{aligned} \tag{3.1}$$

where $t \in [-1.0, 1.0]$ is the parameter of the first curve, $s \in [-1.0, 1.0]$ is the parameter of the second curve and r is the user provided radius for the rounding curve. If the parameter values for t and s which satisfy the above criteria are found, the rounding arc is defined by three points: C as its center and A and B as the endpoints of the arc.



(a) Kink with curves lying in the same plane

(b) Kink with curves lying in different planes

Figure 3.8: Two examples for curve connections needed to be rounded

For 3D cases, the situation becomes more complicated, since nothing guarantees that the closes points to A and B on the intersection line will coincide. Such a scenario can be seen on fig. 3.8b, where the closest point to A is $C1$ and the closest to B is $C2$. This fact necessitates to extend the conditions given in eq. (3.1) with the requirement that $C1$ and $C2$ should coincide at the parameter values of t and s, where the rounding arc was found:

$$\begin{aligned} |C1(t,s) - A(t)| - r &= 0, \\ |C2(t,s) - B(s)| - r &= 0, \\ |C2(t,s) - C1(t,s)| &= 0. \end{aligned} \tag{3.2}$$

To find the right parameter values for t and s, the above criteria were summed up in a single objective function:

$$\begin{aligned} f(t,s) = &(|C1(t,s) - A(t)| - r)^2 + (|C2(t,s) - B(s)| - r)^2 + \\ &+ (|C2(t,s) - C1(t,s)|)^2 , \end{aligned} \tag{3.3}$$

$$\tag{3.4}$$

with the parameter bounds $t \in [-1.0, 1.0]$ and $s \in [-1.0, 1.0]$.

Looking at the structure of the objective function in eq. (3.4), it can be clearly seen, that it takes a non-negative value for every t and s, and that it only becomes zero, when each individual squared term in the summation becomes zero, which means that all of the criteria stated in eq. (3.3) were met, and the parameters for the rounding arc were found.

For finding the optimum of the objective, the Bounded Optimization By Quadratic Approximation (BOBYQA) algorithm introduced by POWELL (2009) was utilized. The C++ implementation of it comes from PITZL (2018). This optimization algorithm has the advantage that the derivatives of the objective functions do not need to be calculated.

Existence of a rounding arc

In general, it can be stated that a rounding arc tangentially connecting to both curves in question can not be found, if co-planar tangential lines to the two curves do not exist. This means that for the 2D case, such an arc can always be found, since the all tangential lines to the curves lie in the same plane as the curves themselves.

Cases where finding such a rounding arc becomes impossible arise, when considering three dimensional curve connection cases, such as the one depicted on fig. 3.9. The case there is quite clear: all tangent lines of the curve FE are perpendicular to the $x-y$ plane; however, the tangent lines to curve ED all live in the $x-y$ plane, just as the curve itself. Therefore, founding a plane for a rounding arc, which would require co-planar tangent lines, is impossible.

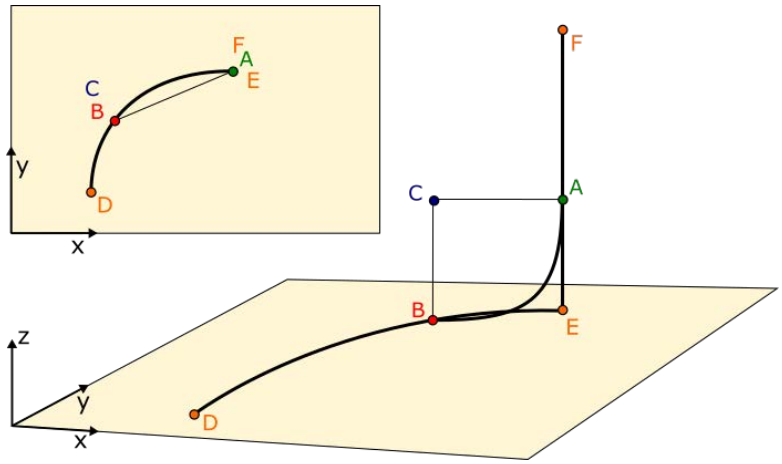


Figure 3.9: Unfeasible curve connection for rounding

In the end, these unfeasible scenarios were not found to be very restrictive to utilize the algorithm for the current application, since:

1. Most connected curves lie in the same layer, therefore they are co-planar.
2. 3D curve connections only occur for curves that connect two layers, and these are usually not from the unfeasible type of 3D curve connections.
3. Even if they are, the algorithm still provides a "best fit" for the problem, which for the small technical radii used in the current application does not really make a significant impact on the generated simulation geometry. In fact, it is almost impossible to notice that those connections are not exactly tangential.

With all of these being stated, the described print path generating algorithm provides a reliable tool to parse the provided data into a representation, from which a volumetric model can be created (as detailed in section 3.4). As an example, on fig. 3.10, the generated print path for the wall geometry from the beginning of the chapter (fig. 3.1) is presented.

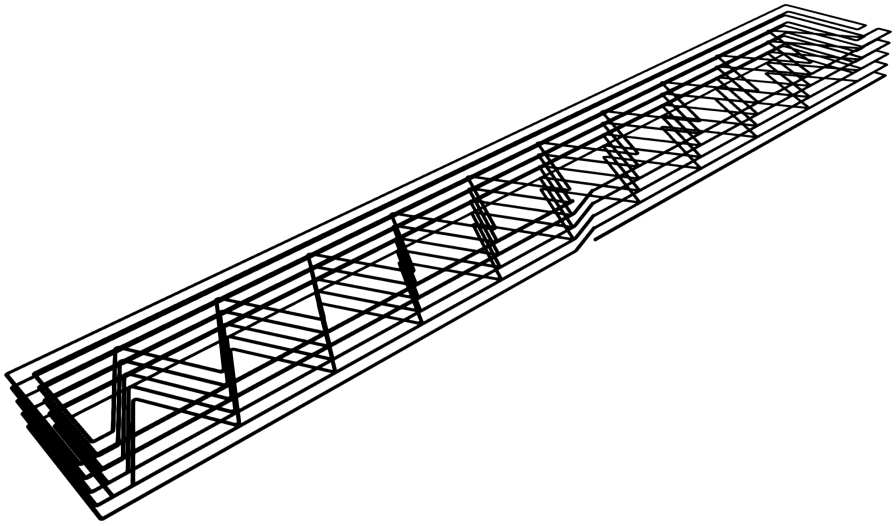


Figure 3.10: The print path of the geometry depicted on fig. 3.1 at the beginning of the chapter generated with a only a small technical radius

3.4 Volumetric model generation

The creation of a 3D geometry from the parsed print path consist of two stages as well. First, individual primitives must be created from the curves building up the print path. This can either be done by conducting a sweeping operation on the curves individually, as it is described in section 3.4.1. Or one can make use of the fact, that curves in the bottom layer are very often equivalent up to a translation with the ones above them in the higher layers; therefore, periodic domains with a primitive sweep as unit cell can be created as described in section 3.4.3 and section 3.4.4. This latter approach turned out to be much faster considering PMC times. The second stage in both cases is the creation of a CSG tree from the generated primitives providing the final geometry.

3.4.1 Cross section and sweep operation

The creation of a primitive volume from the individual 3D curves of the print path happens by sweeping a pre-defined cross section along these curves. How this concept is actually realized in the AdhoC++ (CMS, 2022) framework's CSG kernel in an implicit manner was already addressed briefly in section 2.2.3, but more detailed explanations can be read by WASSERMANN (2020) and WASSERMANN et al. (2017).

As stated in section 3.3.1 and visualized in fig. 3.11 the print path parsed from the data file always runs at the top of the layer, since it is the path the nozzle of the printing robot has to take while laying the specific layer. This fact has to be taken into account, when defining the cross section in the local coordinate system of the curves, as it is required by the AdhoC++ (CMS, 2022) implementation.

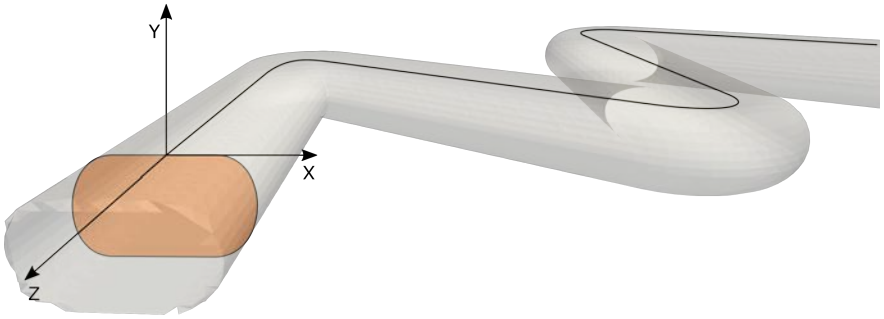


Figure 3.11: Sweeping of a print path with a given cross section

To be more exact, the implementation requires the definition of a work plane at one end of the 3D curve, with the work plane's origin usually taken as the end point of the curve. A work plane then has a local coordinate system defined, in which a sketch can be drawn, which will be swept along the curve. When drawing this sketch (cross section) for the current application case, one has to take into account the aforementioned fact, that the print path runs at the top of the layer. Therefore, the origin of the work plane is defined also there, so the sketch has to lie entirely in the negative y domain using the coordinate system depicted in fig. 3.11. Furthermore, the shape of the cross section has to be defined in a manner that it best represents the actual cross section after the printing of the geometry is ready. In most cases, it means a deformed rectangle-like shape, similar to the colored cross section in fig. 3.11, with a flat top and bottom and slightly curved sides, with the widest point being at half of the layer height. This is merely an approximation though, and a more exact cross section can be defined knowing all the details of the printing machine and its nozzle, the material behaviour, printing speed, etc. In general, the shape should be determined in accordance with all the parameters in mind which can have a strong influence on it.

The storage of the generated individual sweeps happens in an array, just as it was the case with the curves previously, depicted in fig. 3.6. Every single time a new sweep is created, it gets added to this array, until the sweep generation process is done.

3.4.2 Building the CSG tree

After the individual sweeps are generated from the curve elements of the print path and collectively made accessible by storing them in an array, an algorithm can be called that builds up the final geometry in form of a CSG tree from the primitives.

The concept of a CSG tree was already addressed in section 2.2.3, so it will not be elaborated on further here. The uniqueness of the CSG tree built in the current application is, that it only involves union operations, since all the created individual sweeps only need to be unified into a single geometry, while looping through the aforementioned array. The process is illustrated in fig. 3.12.

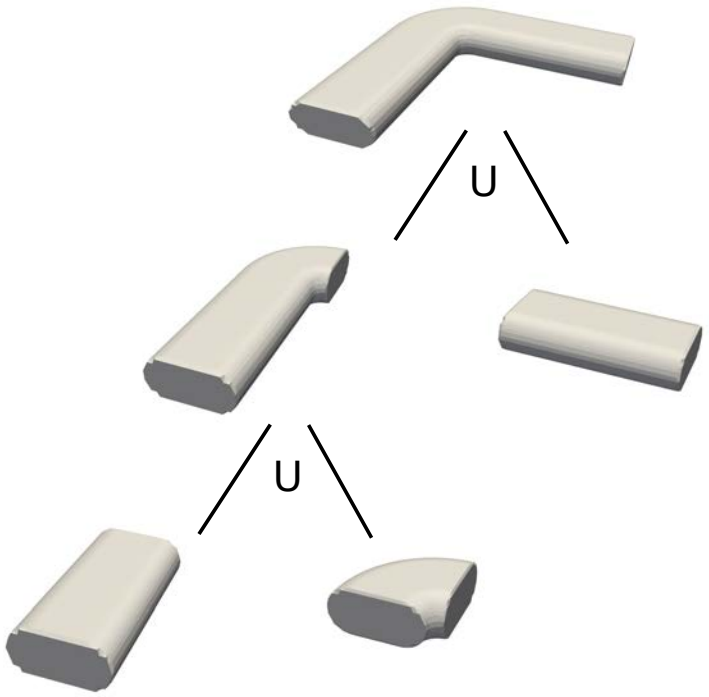


Figure 3.12: Building a CSG tree from primitive sweep volumes

Creating the final geometry in this manner; however, leads to quite expensive point inclusion tests, as it was found in section 4.2. This is mainly due to the two following facts:

1. Due to the shear number of the primitive sweeps that need to be created for a complex wall geometry, the binary tree becomes very deep. With the depth of the tree, on average, the cost of deciding, whether a point is inside the the geometry or not increases significantly as well.

2. Since every new sweep added to the tree gets unified with the tree built from all the previous primitives, the resulting final tree is highly unbalanced, meaning that the first branch at almost every level is significantly deeper than the second. For union operations it is sufficient to return true if one of the branches evaluates to true, which can significantly speed up the PMC tests. However, with an unbalanced tree, one will have a branch which can be evaluated very quickly, but it can almost never be avoided to call the other branch, since a shorter branch contains only a small portion of the geometry, and for most test points it will evaluate to false. But on the other hand, the deeper branch will be significantly deeper than in case of a balanced binary tree, and also taking more time to evaluate.

3.4.3 Vertical periodic domain

As it was addressed at the beginning of section 3.4 and at the end of the previous section (section 3.4.2); furthermore, tested in section 4.2, building up the CSG tree as described in section 3.4.2 from individually created sweeps results in expensive point inclusion tests as the geometry of the model gets more and more complex.

One of the reasons for this, as addressed in the preceding section (section 3.4.2), is the very high number of primitives that have to be combined into the final geometry. This number can be significantly reduced by making use of the vertical periodicity present in almost every printed wall geometry. With this approach, ideally, one only needs to generate the sweeps for the curves in the bottom layer and all the higher sections of the layers just get "pulled back" into these so called unit cells for evaluation as it will be described in this section later on. The resulting periodic domains do not add a very significant overhead to a simple sweep evaluation, since the defined "pull-back" operation only consist of a few extra additions and multiplications, which are way cheaper in comparison to carrying out a sweep PMC operation.

But before continuing with the description of the implementation, one final consideration has to be made. Some wall geometries of interest have their layers not simply printed vertically on top of each other, but in a slightly slanted manner, as it is the case for the wall showed in fig. 3.13. Since this shift of the layers can happen in every direction parallel to the previous layer and with varying amount, an implementation taking advantage of vertical periodicity has to account for this application case as well, so that the final tool is robust enough.

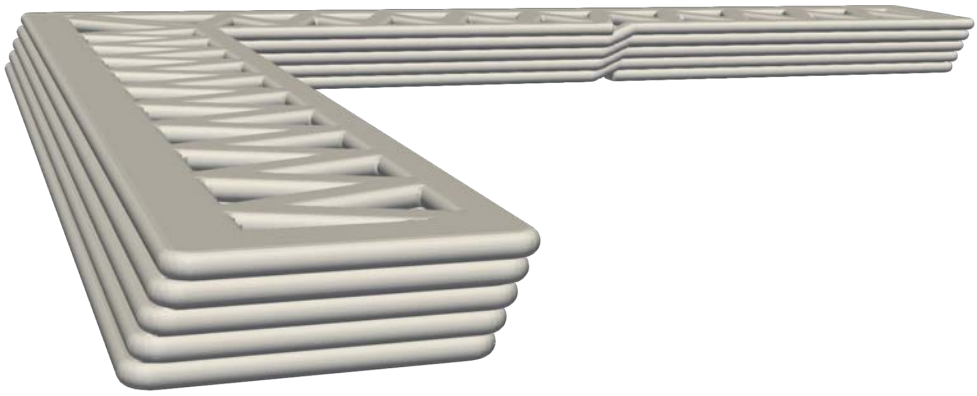


Figure 3.13: Slanted wall geometry with vertical periodicity

To determine the additional calculations required by a primitive periodic domain, let us consider a schematic depiction of a section of a slanted wall domain showed in fig. 3.14.

The setup is the following: there is primitive sweep as unit cell defined with its local Cartesian coordinate system (with coordinates lower case x, y and z), pointed to by the vector $\mathbf{o}$ from the global Cartesian frame (denoted by capital X, Y and Z). The unit cell also has a bounding box defined by the dimensions l_x, l_y and l_z, with the index referring to the local coordinate axis. The geometry at hand is such that there are two other layers above the bottom one printed, each containing a section, which is equivalent to the defined unit cell up to a translation. The sweep in the second layer can be obtained by shifting the unit cell by the vector $\mathbf{v}$ with respect to its local origin. The same holds for the third layer sweep, but with a shift of $2\mathbf{v}$ with respect to the unit cell origin. One must observe that disregarding the differences due to the translations, the three different sweeps would give the exact same results for a point inclusion test called with any point of the entire computational domain. Therefore, the idea is to first transform points into the local frame of the unit cell, then find out to which layer they correspond to, and then ”pull” them back to the bottom layer, where the standard sweep point inclusion test can be carried out with the help of the unit cell.

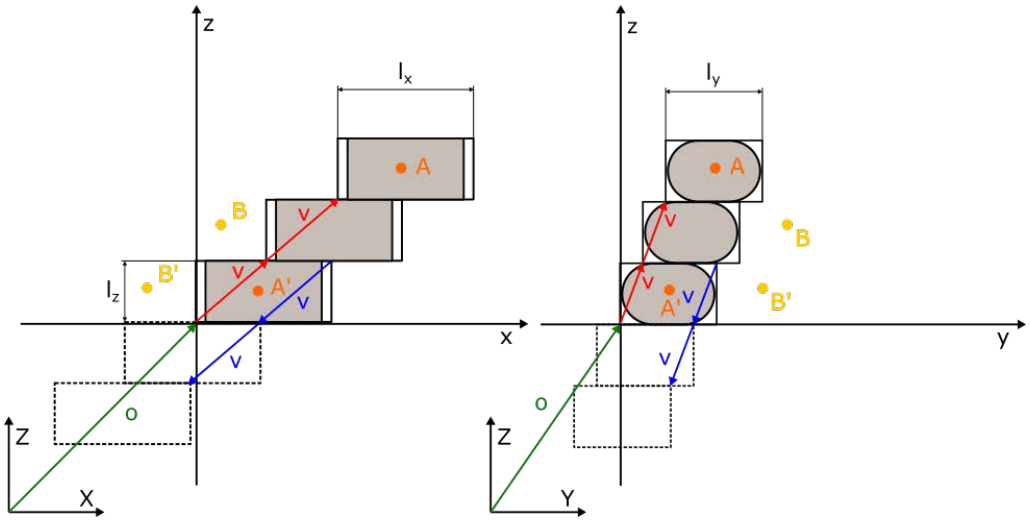


Figure 3.14: Vertical periodic domain

For example, when calling the PMC scheme on point A in fig. 3.14, first the vector $\mathbf{o}$ must be subtracted from the global coordinates of A to transform it into the local system of the unit cell. Then - since it is in the third layer - the vector $\mathbf{v}$ has to be subtracted twice from it which would translate it into A'. Finally, before calling the point inclusion test of the sweep of the unit cell, $\mathbf{o}$ must be added again to the ”pulled-back” A' point, since the PMC algorithm of the sweep expects a point in global coordinates. After the transformation back to global coordinates, the PMC method of the unit cell sweep would found A' to be inside the domain; therefore, the same would be returned for A as well.

Carrying out the same procedure with B, corresponding to the second layer from the same figure, would end up calling the point inclusion test on B', which would correctly return that it is outside, and so is the original point B. The switching from global to local frame happens, because determining the layer corresponding to the point tested is easier there, as it will be showed later.

If the translational relation between the stacked up CSG sweep primitives would have to be described with a vector with negative local z coordinates (which is not expected in the case of 3D printed walls, since they are built from the bottom up), the same arguments still can be made, and the algorithm implemented accordingly. Looking at a second scenario depicted in fig. 3.14, with only the dashed contour lines of the bounding boxes of the geometries shown (but still using the same unit cell), one can notice that the only significant difference is that the unit cell lies in a different quadrant of the local coordinate system than all the other geometries, all the points it contains having positive z coordinate values. This problem can be easily solved by subtracting the vector $\mathbf{v}$, describing the translation, one times more, if $\mathbf{v}$ has a negative z coordinate. This one extra "pull-back" step will bring the points to the same quadrant the unit cell is in. One can conceptually think about this extra step, as moving the local coordinate frame to the upper right corner and doing everything as in the original case, but in case of this example, with the help of the depicted blue $\mathbf{v}$ translation vector. But the main take away from this example is, that if the z coordinate of $\mathbf{v}$ is negative, an extra "pull-back" step has to be made.

Let us describe now the the above process in a more robust manner, mathematically. For denoting the vector pointing to the tested point in the global frame, $\mathbf{X}(X, Y, Z)$ will be used, while referring to the same point in the local coordinate system of the unit cell of the periodic domain, the same notion with lower case letters will be applied ($\mathbf{x}(x, y, z)$). The number of cells building up the periodic domain is noted as n. For everything else, the same notation from the previous example depicted on fig. 3.14 is kept.

So finally, the point inclusion test for the vertical periodic domain implemented works as described in Algorithm 3.1.

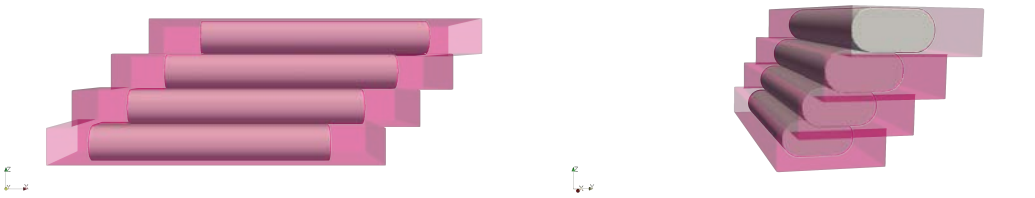
Algorithm 3.1: PMC for a vertical periodic domain

```
1 procedure isInsideDomain( Vector X )
2     if ( !myBoundingBox⟶contains( X ) ) return false
3     x = X − o
4     k = min( floor( x[2] / v[2] ), n − 1 )
5     x′ = x − (k + int(v[2] < 0)) · v
6     X′ = x′ + o
7     return myUnitCell⟶isInsideDomain( X′ )
8 end
```

At the beginning of the algorithm the point $\mathbf{X}$ is tested against the bounding box of the entire periodic domain. If it was found inside, it gets translated to the local coordinate system of the unit cell by subtracting the coordinates of the unit cell origin. Then it is determined how many times the translation vector $\mathbf{v}$ has to be subtracted to "pull-back" the point into the unit cell's layer. The number of maximum "pull-backs" is bounded by the total number of cells. After this, the point has to be shifted back to the unit cell layer with the help of k, determined before. If the z coordinate of the translation vector $\mathbf{v}$ is negative, an extra "pull-back" step - as discussed in the example beforehand - has to be performed.

Finally, the shifted point is transformed back into the global coordinate frame and the PMC method of the unit cell sweep is called on $\mathbf{X}'$ to decide, whether it is part of the geometry or not.

Having this algorithm implemented already allows for creating vertically periodic domains for walls with a geometry slanted in an arbitrary direction. As a basic demonstration, a periodic domain of four sweeps was generated, where the geometry is slanted in both the x and the y direction as showed in fig. 3.15.



(a) Vertically periodic domain - X-Z view (b) Vertically periodic domain - Y-Z view

Figure 3.15: Vertically periodic domain example

Problems with periodic domain for layer switching curves

The above implementation already works fine on almost the entirety of a wall domain, with the one notable exception of the subdomain where the switching between layers happens. Since the bounding boxes of the primitives are always defined aligned with the global Cartesian coordinate system, sweeps which are rotated compared to this alignment can not touch without having their bounding boxes overlap as it is depicted in fig. 3.16.

For the previously described algorithm this would pose a serious problem, since it was relying heavily on the fact that each geometry lies in distinct boxes, and the z dimension of these boxes (corresponding to the z coordinate of the translation vector $\mathbf{v}$) was used to determine to which layer of curves a tested point should belong to. This then defined, how much it had to be ”pulled back”, so that it ends up at the right place relative to the unit cell domain.

But for the scenario with overlapping bounding boxes, this approach is clearly unfeasible Therefore, a new algorithm was developed for this case, which will be elaborated more on in section 3.4.4.

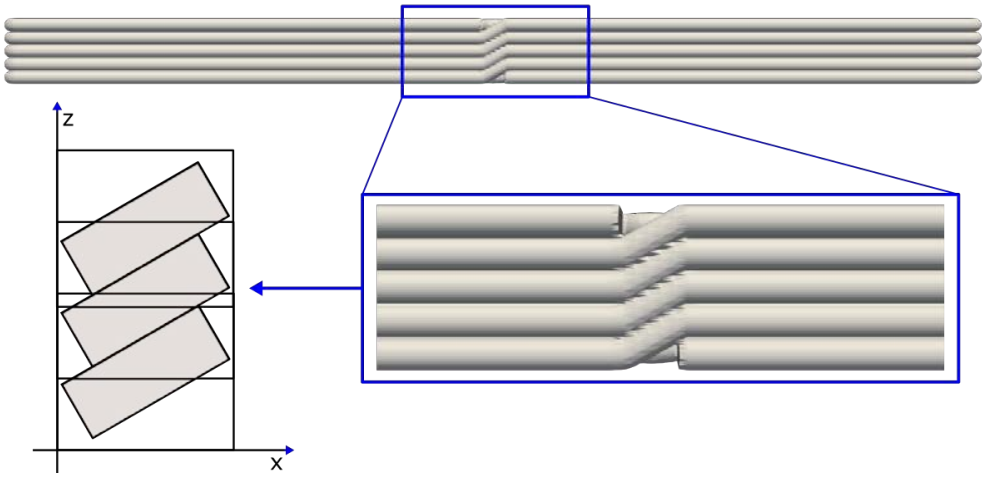


Figure 3.16: Overlapping unit cell domains of layer switch curves

3.4.4 Vertical periodic domain with overlapping cells

The periodic domain approach introduced in this chapter gives a possible answer to the problem mentioned at the end of the last section (section 3.4.3). As it is schematically depicted on fig. 3.17, if the cell domains have to overlap in order to get their contained geometries touch each other, there will be points, which will be contained by the bounding boxes of two (or in extreme cases more) sweeps (like it is the case for point A on the drawing).

For the ”pull-back” operation of a periodic domain approach, though, it must be clearly decided to which layer the given point corresponds to, so that the translation vector $\mathbf{v}$ (in red on the right drawing of fig. 3.17) can be subtracted from it the necessary amount of times to bring the point back for the final PMC test into the base unit cell layer.

In this implementation it was decided, that the upper bounds of a cell's bounding box will count as the boundary between two layers. The numbering of layers according to this convention is given in green in fig. 3.17, with k_x, k_y and k_z corresponding to each coordinate direction. Since these three numbers can differ for some tested points, to comply with the previously decided rule about the upper bounds being considered as boundaries between layer-correspondence, the largest of the three is always taken as the layer the tested point belongs to.

Therefore, on figure fig. 3.17, the point A belongs to the layer 0 while the point B to layer 2. So while the ”pull-back” operation with the translation vector $\mathbf{v}$ (in red) will be carried out 2 times on B, for the point A no ”pull-back” operation is necessary at all.

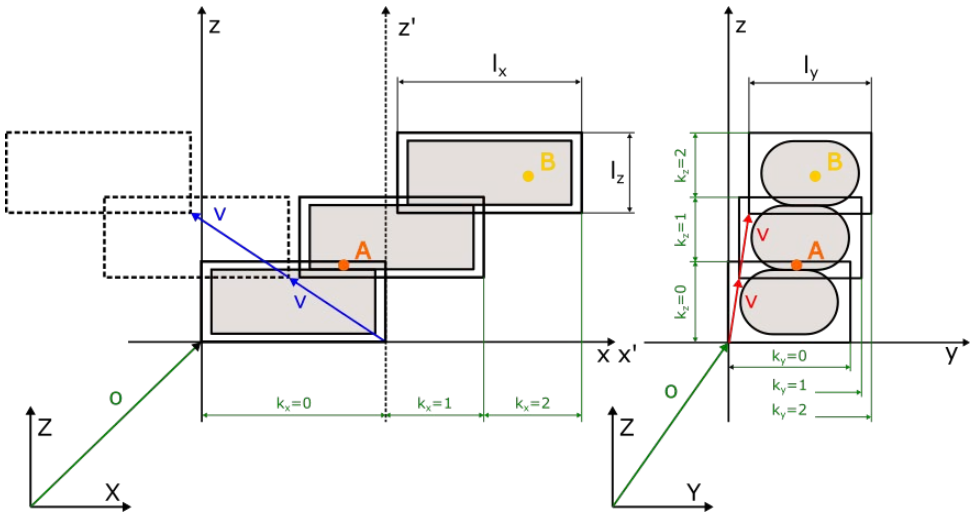


Figure 3.17: Vertical periodic domain with overlap

In case of overlapping unit cell bounding boxes, handling periodically translated cells in negative directions becomes also more complicated than just performing an extra ”pull-back”, if the z coordinate of the translation vector v is negative. The previously described approach can be used for these cases as well, if the local coordinate system is shifted with the dimension of the unit cell's bounding box, corresponding to which coordinate of the translation vector $\mathbf{v}$ (in red on fig. 3.16) is negative. If all components of it are negative, then the local coordinate system should be shifted in all coordinate directions with the dimension of the unit cell in the corresponding direction. In fig. 3.17 e.g., the domains outlined only with dashed lines on the left of the figure showcase such a situation. There the x coordinate is negative; therefore, the coordinate system is translated to the right (x'-y' system) with the x dimension of the unit cell l_x. Then the same translation vector, but now referring to a different origin ($\mathbf{v}$ in blue on fig. 3.17), can be used to perform the ”pull-back” operation.

If one would implement the PMC test as described so far, the problem depicted on fig. 3.18 would show up. While the algorithm would work fine with only the rectangular domain of the bounding boxes, it does not in case the contained sweeps have different subdomains of the overlapping region as part of their geometry. This can be best seen on fig. 3.18b, where it is clearly visible that the bounding boxes of the previous cell cut out the bottom part of the geometry of the next one. This is due to the fact, that the ”pull-back” operation - as introduced in the preceding paragraphs - happens based on the lower of the overlapping cells. And since the upper part of the lower cell's geometry contain different points within the overlapping region than the lower part of the sweep of the upper cell, the latter gets removed from the combined geometry, since the corresponding domain in the lower cell is empty. This shows the necessity to carry out the ”pull-back” operation and PMC test according to all of the overlapping cells and not only one of them.

Considering this last addition to the implementation, the full procedure of the PMC for periodic domains with overlapping bounding boxes is described with the help of Algorithm 3.2. The notation remained the same as in Algorithm 3.1

Algorithm 3.2: PMC for a vertical periodic domain with overlapping cells

1 **procedure** isInsideDomain(Vector $\mathbf{X}$)

2 **if** (!myBoundingBox $\longrightarrow$ contains($\mathbf{X}$)) **return** false

3 $\mathbf{x} = \mathbf{X} - \mathbf{o}$

4 $\mathbf{x}[0] = \mathbf{x}[0] - \text{int}(\mathbf{v}[0] < 0) \cdot l_x$

5 $\mathbf{x}[1] = \mathbf{x}[1] - \text{int}(\mathbf{v}[1] < 0) \cdot l_y$

6 $\mathbf{x}[2] = \mathbf{x}[2] - \text{int}(\mathbf{v}[2] < 0) \cdot l_z$

7 $i_x = \text{floor}\left(\frac{||\mathbf{x}[0]|-l_x|}{|\mathbf{v}[0]|}\right)$ $i_y = \text{floor}\left(\frac{||\mathbf{x}[1]|-l_y|}{|\mathbf{v}[1]|}\right)$ $i_z = \text{floor}\left(\frac{||\mathbf{x}[2]|-l_z|}{|\mathbf{v}[2]|}\right)$

8 $k_x = \text{int}\left(\frac{\mathbf{x}[0]}{l_x} > 1\right) + \text{int}\left(\frac{\mathbf{x}[0]}{l_x} > 1\right) \cdot i_x$

9 $k_y = \text{int}\left(\frac{\mathbf{x}[1]}{l_y} > 1\right) + \text{int}\left(\frac{\mathbf{x}[1]}{l_y} > 1\right) \cdot i_y$

10 $k_z = \text{int}\left(\frac{\mathbf{x}[2]}{l_z} > 1\right) + \text{int}\left(\frac{\mathbf{x}[2]}{l_z} > 1\right) \cdot i_z$

11 $k = \max(k_x, k_y, k_z)$

12 isInside = false

13 overlaps = 0

14 **if** (($|\mathbf{v}[0]| < l_x$ **or** $|\mathbf{v}[1]| < l_y$) **or** $|\mathbf{v}[2]| < l_z$)

15 overlaps = $\min\left(\text{ceil}\left(\min\left(\frac{l_x}{|\mathbf{v}[0]|}, \frac{l_y}{|\mathbf{v}[1]|}, \frac{l_z}{|\mathbf{v}[2]|}\right)\right), n\right)$

16 **end**

17 checks = $\min(\text{overlaps}, n - k)$

18 **for** $j = 0$ to checks

19 $\mathbf{x}' = \mathbf{x} - (k + j) \cdot \mathbf{v}$

20 $\mathbf{X}' = \mathbf{x}' + \mathbf{o}$

21 $\mathbf{x}[0] = \mathbf{x}[0] + \text{int}(\mathbf{v}[0] < 0) \cdot l_x$

22 $\mathbf{x}[1] = \mathbf{x}[1] + \text{int}(\mathbf{v}[1] < 0) \cdot l_y$

23 $\mathbf{x}[2] = \mathbf{x}[2] + \text{int}(\mathbf{v}[2] < 0) \cdot l_z$

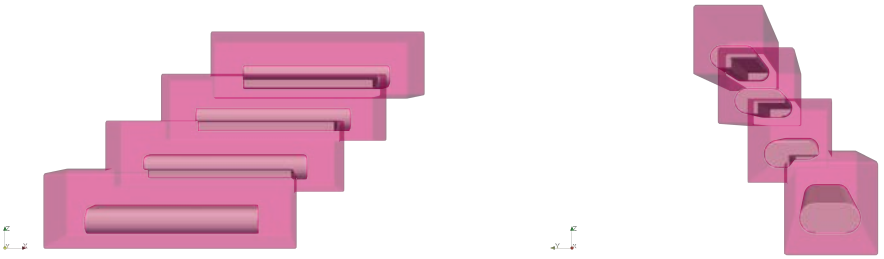
24 isInside = isInside **or** myUnitCell $\longrightarrow$ isInsideDomain($\mathbf{X}'$)

25 **end**

26 **return** isInside

27 **end**

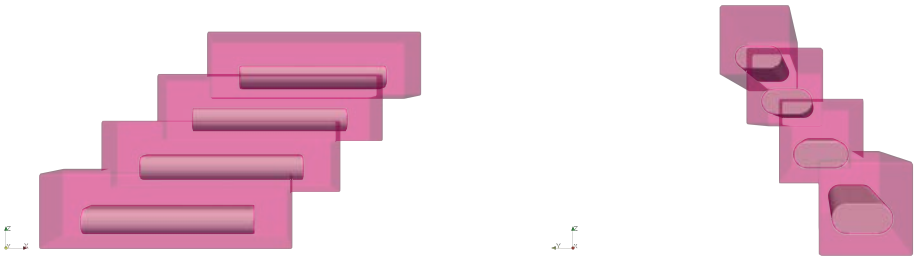
The algorithm starts by first testing the input point against the bounding box of the periodic domain, and then translating it into the local coordinate system of the unit cell, if it was found inside. In line 4 to line 6 additional transformation of the coordinates is carried out, if the corresponding coordinate of the translation vector $\mathbf{v}$ is negative, as described in the previous example. Afterwards, in line 7, it is determined in each directions how many extra "pull-backs" have to be made above the one, which already must be done, if the point is outside of the bounding box of the unit cell (see depiction of k_x, k_y and k_z values in green in fig. 3.17). Then, the number of necessary "pull-back" steps has to be determined in each direction by deciding, whether the point is within the unit cell domain, and if it is already outside, the extra steps determined previously have to be added. In the end, the maximum of these values must be taken to get the correct number of "pull-backs" overall. In line 14 to line 16, based on the relation of unit cell dimensions and the coordinates of the translation vector $\mathbf{v}$, it is decided, whether the cells overlap. If they do, the number of overlapping cells is determined, which is maximum the total number of cells (n). Afterwards, the amount of point inclusion checks that has to be carried out due to the overlaps can be reduced, if the current layer's distance to the upper most cell is smaller than the total number of overlaps. From line 18 to line 25 the determined number of point inclusion tests are carried out with the different number of "pull-backs" and translations back to the original frame, according each of the overlapping layers. Finally, if any of the points called with the PMC algorithm of the unit cell evaluates to true, the entire subroutine will return true. Using this last, final version of the algorithm, the faulty example case from fig. 3.18 already gives the desired diagonally repeating, overlapping domain depicted in fig. 3.19.



(a) Faulty overlapping example - X-Z view

(b) Faulty overlapping example - Y-Z view

Figure 3.18: Problems with performing the point inclusion test only on one of the overlapping domains



(a) Correct overlapping example - X-Z view (b) Correct overlapping example - Y-Z view

Figure 3.19: Results for the correctly implemented PMC algorithm for overlapping periodic domains

3.4.5 Updated path data structure for creating periodic domains

Storing the parsed curves in a single array like it was described in section 3.3.1 and depicted on fig. 3.6, would be impractical for the generation of the periodic doamin primitives described in the previous sections (section 3.4.3 and section 3.4.4). As it can be seen from the algorithms described there, it must be decided beforehand, based on which path curves the generation of a periodic domain is possible, and whether there are overlaps expected between the bounding boxes of the generated cells. Writing a subroutine, which decides this curve-by-curve read from an array would be complicated and inefficient, compared to parsing the curve data from the input file into a more suitable data structure.

The new data structure chosen for the storage of the parsed curves is depicted on fig. 3.20. As it can be seen, it contains two arrays of curve arrays. One of them corresponding to the layer curves, which are completely on a single plane, while the other stores the curves, which connect the separate layers. The arrays, contained by the two main arrays, hold curves layer-by-layer. So the first array in the ”In-plane Layer Curves” array would correspond to the bottom layer, followed by the curves connecting the first with the second layer, stored in the first array of the main ”Layer Switcher Curves” array. Then the second array of the ”In-plane Layer Curves” follows, with the curves in the second layer, then the second array of the ”Layer Switcher Curves” to connect to the next layer, and so on, until the end of the print path is reached.

One can see, that keeping track of the print path information in this format still makes the parsing of the data manageable in a single loop over the data file lines, deciding based on the z coordinate of the two end points of each curve, to which array it should belong to. Since the curves in the data file are stored following the path the printing robot’s nozzle should take, and in this path no arbitrary jumps are expected, one can keep checking for the points where the z coordinates of the curves differ, indicating that a switch between layers is occurring.

This means that from the starting point of the geometry (in fig. 3.20, indicated in red at both the bottom geometry and in the upper left part of the data structure depiction) the parser algorithm keeps comparing the z coordinates of each curve's end points. As long as they agree, the parsed curves get stored in the first array of "In-plane Layer Curves", representing the first layer. At the point, where the end z coordinates first differ, the place of storage is changed to the first array of "Layer Switcher Curves", representing the curves connecting the first layer with the second (in fig. 3.20 they are built up from two arcs and a single line). New curves get added to this array, until the z coordinates of the ends of the curve parsed equal again. Then the second array of "In-plane Layer Curves" is started, and continued until the z coordinates of the end points differ again. This process is then repeated like this, until the end of the input file is reached and all curves get stored in the "Print Path Data Structure".

Keeping track of the path information in this way still allows for recovery of the path curve in a single row of curve pointers, if needed, concatenating the arrays from the main arrays in an alternating manner as shown by the path in red in the upper half of fig. 3.20. However, it also makes it easier to check for possibilities for generating periodic domains. The decision about whether to use the computationally more expensive version with overlaps allowed or the faster one without overlaps can be made based on the two main arrays. If curves are part of "In-plane Layer Curves", the version without overlaps can be used, since the curves there are in planes parallel to the x-y plane, therefore aligned with the boundaries of their bounding boxes. The version allowing for overlaps of the individual cells has to be used only for the curves in "Layer Switcher Curves".

For creating the periodic domains, then it has to be decided, which curves from the different layers are equivalent up to translation (and therefore having equivalent sweeps generated from them as well). This can be done in each of the main arrays in a column-wise manner as indicated by the green and blue brackets in fig. 3.20.

When checking for the possibility to create periodic primitives, each column of the main arrays is looped through from the bottom layer to the top and checked, which of the consecutive curves are translatable into each other. For those which are, the translation vector $\mathbf{v}$ for the domain generation as described in section 3.4.3 and section 3.4.4 is determined as the difference of the vectors of the start points of the curves. Curves which found not to be equivalent with any of its neighbours get swept individually, as it was already done in section 3.4.1. If such curves were found in the column, the generated corresponding sweeps get combined with the periodic domain(s), built from the column into a CSG tree, ending up with an array of such subdomains for each column of curves.

In the end, these subdomains get combined into a single CSG tree with the help of union operations, providing the final geometry. As it was found in section 4.2, building up the geometry in this manner results in much faster solution times for simulations.

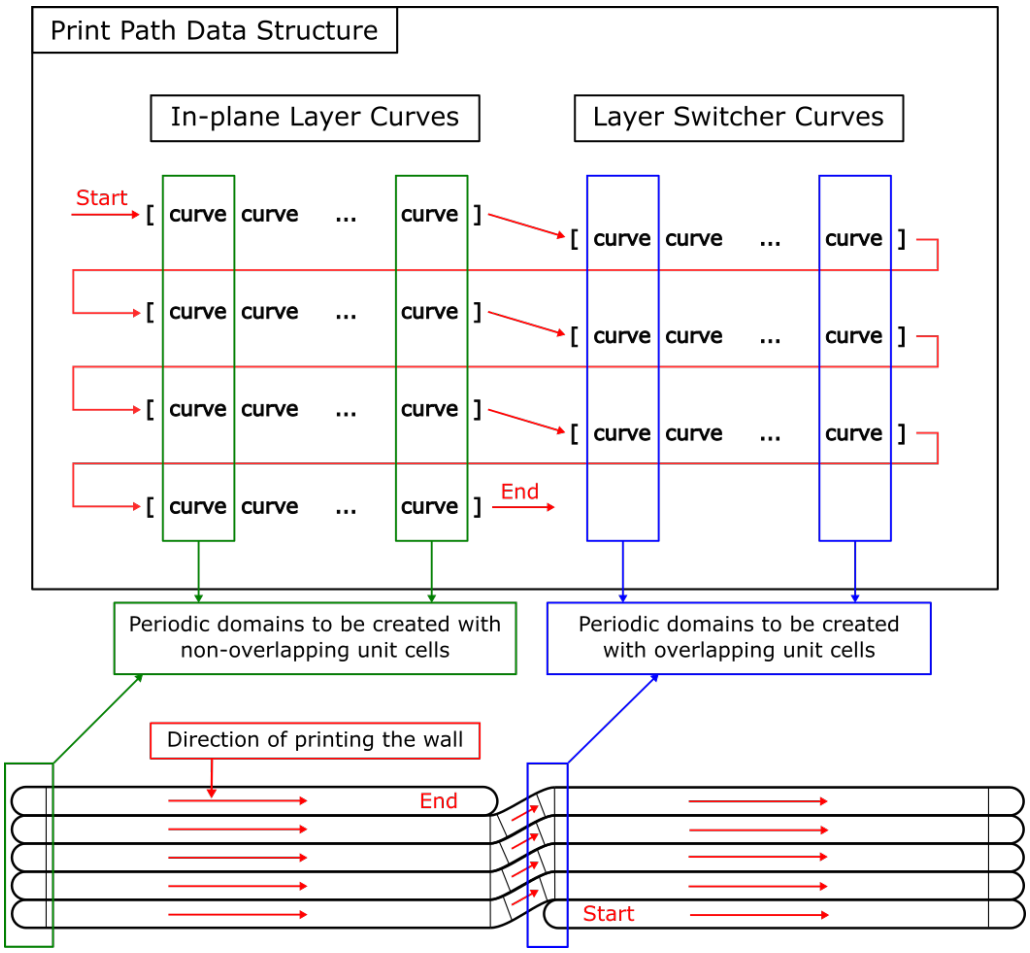


Figure 3.20: The print path data structure of curves for periodic domains

3.5 The way of defining boundary conditions

Defining the surfaces on which the desired boundary conditions can be applied with the help of the penalty approach, as mentioned in section 2.4.4, and addressed in more detail by WASSERMANN (2020) and FELIPPA (2004), is the most time consuming part of the entire process of setting up a simulation with the current tool.

Keep in mind that for defining boundary conditions in the current application, the surface description is needed in the STL format. The process of generating these surface descriptions is visualized in fig. 3.21. First, the boundary of the implicit geometry must be generated with the help of, e.g., the marching cubes algorithm (LORENSEN & CLINE, 1987). This algorithm has been already implemented in the AdhoC++ (CMS, 2022) framework and the corresponding subroutine processes the results into a .vtu file, which is a native format for the visualisation tool, ParaView (PARAVIEW DEVELOPERS, 2022) .

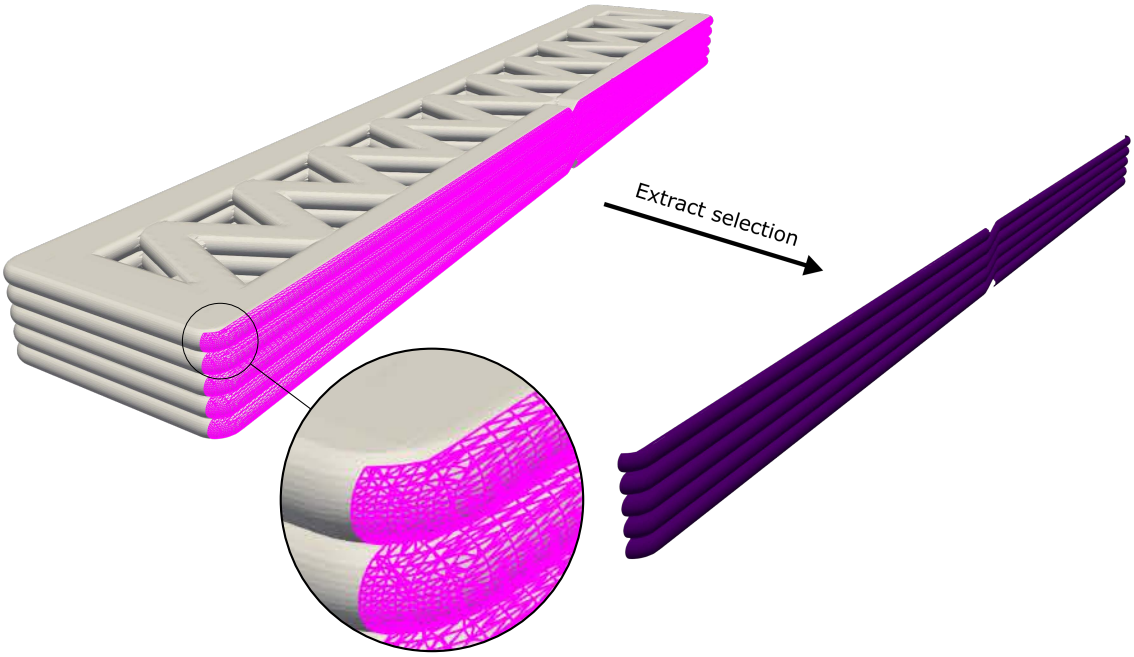


Figure 3.21: Extracting surfaces for boundary condition application

After the surface geometry of the wall is available, it has to be imported into ParaView (PARAVIEW DEVELOPERS, 2022) . The desired boundary surface has to be then defined by selecting the corresponding triangles on the wall surface (highlighted in pink in fig. 3.21). After this step, the selected triangles can be extracted into a separate surface (displayed in purple in fig. 3.21), which can be saved in the end in STL format. These STL surface files then can be called by the simulation upon applying the prescribed boundary conditions in the model.

3.6 Output data

For outputting results of the simulations, two main approaches are available. On the one hand, the physical quantities linked to the computational geometry, as detailed more in section 3.6.1, can be inspected. On the other hand, for thermal and linear elastic problems, two respective scalar values were defined (see section 3.6.2) and calculated with each simulation, providing a quantity to describe the overall performance of the wall design in question.

3.6.1 Result files for detailed inspections

In the AdhoC++ (CMS, 2022) framework, there are several post processing options already implemented, of which the current application makes also good use of. Most of these are related to providing information to the user about the physical quantities that were determined with the help of the simulation. These results usually get post processed on a mesh linked to the geometry of the model, and can be inspected then with the help

of the software ParaView (PARAVIEW DEVELOPERS, 2022) . Furthermore, data about the element and integration mesh and further technical details of the computation can be written out and visualized in the same format. The post processors relevant for this application will be listed in the following.

Thermal analysis

For a thermal analysis of the wall, information about the temperature distribution (primary solution field of the heat equation) and heat fluxes were post processed into a point-wise data set, and displayed on the wall geometry. For example, a visualization of the temperature field can be seen on fig. 3.22.

Furthermore, total fluxes as reactions to the applied temperature boundary conditions at the predefined surfaces also can be determined, which will be relevant for calculating the U-Value performance measure for the wall as introduced in section 3.6.2.

Figure 3.22: Temperature field visualized on the geometry

Linear elastic analysis

For linear elastic problems, displacements (3 components per point) and stresses (6 components per point) can be determined as point-wise data sets. The calculated reactions on the boundary surfaces in this case are the total forces.

Discretization details

Along the physical quantities, information about the discretization of the problem can be also post processed and visualized. The most relevant options available are the active computational mesh (fig. 3.23) and the integration mesh (see section 2.4.3) visualized as cell data and the integration points as point data.

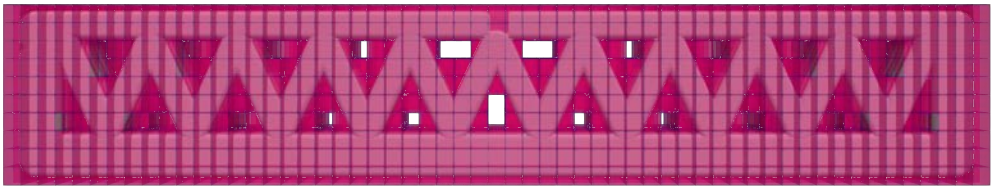


Figure 3.23: Finite cell mesh of the simulation visualized

3.6.2 Performance measures

As the main goal of this thesis was to link physical simulations into the design loop (see fig. 3.24 repeated here for convenience from section 1.3) of Fabrication Information Modeling, some overall quantities needed to be defined as feedback, which can express the performance of a wall design in a single number.

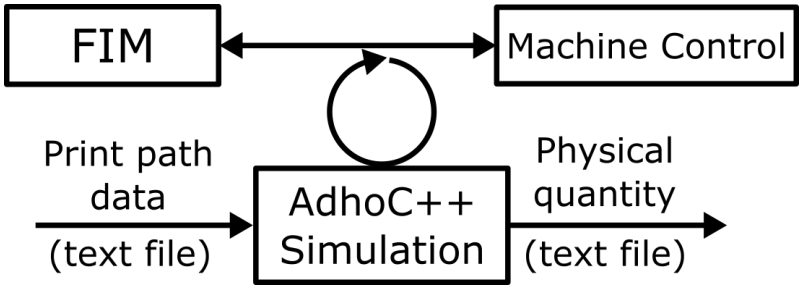


Figure 3.24: The place of AdhoC++ simulations in the design loop

There are of course other possibilities as well, but in this implementation for thermal simulations the so called U-Value, while for linear elastic problems the average vertical displacement was taken for this purpose.

U-Value

The U-Value ($U\left[\frac{W}{m^2K}\right]$) for simulation models as described, e.g., by the HTFLUX DEVELOPERS (2022), can be determined from the total flux through the surface ($\Phi[W]$), the area of the surface ($A[m^2]$) and the temperature difference ($\Delta T[K]$) between the two sides of the wall. The formula for this reads as follows:

$$U = \frac{\Phi}{A\Delta T} \tag{3.5}$$

The total heat flux through the respective surface can be determined by the reaction post processor mentioned already in section 3.6.1, using the same STL surface as for the boundary condition application to calculate the total flux reaction on. The area of this same surface for the calculation can then be easily determined by adding up the areas of the individual STL triangles. Smaller U-Values indicate lower conductive properties of the wall; therefore, the one with the smallest U-Value is considered to be the best from the point of heat insulation.

Average vertical displacement

As performance measure for linear elastic simulations on the 3D printed wall geometries, the average vertical displacement ($\bar{u}_z[m]$) over the top surface of the geometry was taken. This can be simply calculated as:

$$\bar{u}_z = \sum_{i=1}^{n} u_{iz} \tag{3.6}$$

This value is taken as a characteristic for the load bearing capacity of the geometric design, by geometries with smaller average vertical displacements taken as better performant.

Chapter 4

Performance evaluation of the implemented simulation tool

In the following, the capabilities of the implementation from the previous chapter (chapter 3) will be examined. For the test simulations, the simple, straight wall geometry depicted on fig. 4.1 will be used. In section 4.1, first, the appropriate resolution for the discretization of the model is determined, i.e., a convergence study is conducted. After finding a good enough resolution of the discretized model, the performance of the two CSG geometry generation approaches from chapter 3 will be compared to an STL and a voxel representation of the same geometry. Finally, the computational cost of the two methods of numerical integration described in section 2.4.3 is compared in section 4.3.

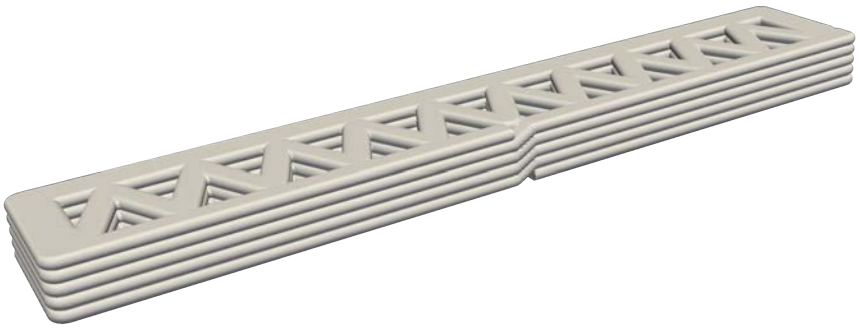


Figure 4.1: The test geometry

The numerical experiments were carried out on a workstation provided by the Chair of Computational Modeling and Simulation (CMS). The main characteristics of the computer are listed in table 4.1. For benchmarking the different simulations and calculating the run time statistics, the command-line tool *hyperfine* (HYPERFINE DEVELOPERS, 2022) was used.

Table 4.1: The test computer specifications

Computer specifications	
Core number	8
CPU Type	Intel ® Core ™ i7-7700 3.60 GHz
Register width	64 bits
RAM	31 GiB

4.1 Necessary model resolution for achieving reliable results

Before running any other simulations with the implemented tool, it had to be determined, at which refinement level of the descretized model can the obtained solution assumed to be close enough to the true solution of the continuous mathematical model of the problem.

General information about the overall dimensions of the geometry, the boundary conditions and the chosen ansatz space can be read in table 4.2. The boundary conditions can be better understood with the help of fig. 4.1. The surface, where the Dirichlet boundary condition T_{BC1} was applied, is the one where the switching between the layers happen (so the frontal surface that can be seen on fig. 4.1). Here, $1°C$ was prescribed, while on the opposing side of the wall, the temperature was set to $0°C$. The area of the respective surfaces differ form each other because of the different geometry due to the layer connecting section, and to the inaccuracies caused inevitably, when extracting the STL surfaces manually (as described in section 3.5). The polynomial degree for the ansatz space was taken as $p = 3$, with an integration partitioning depth of 4 for the adaptive octree method (described in section 2.4.3), since previous experience has showed, that these parameters are usually enough to obtain converged discrete solutions with a reasonably fine mesh resolution.

Table 4.2: General data for the convergence study. With p denoting the ansatz order and d the integration depth for the octree partitioning as described in section 2.4.3

p	d	Temperature BC $[°C]$		Surf. Area $[m^2]$		Dimensions $[m]$			Material
		T_{BC1}	T_{BC2}	A_{BC1}	A_{BC2}	x	y	z	$\kappa\left[\frac{W}{mK}\right]$
3	4	1	0	0.313158	0.312735	1.88	0.327	0.125	0.38

The *h-refinement* steps of the study are given in table 4.3. The number of elements in each direction was chosen in a manner that the discretization has roughly the same amount of elements per unit length in every dimension. Since due to the different surface areas the calculated U-Values (as defined in section 3.6.2) at the to ends would never agree, to observe the convergence behaviour of the model, the total flux was calculated at both sides. The change of the total fluxes with the increasing number of elements is depicted on fig. 4.2, together with the temperature field displayed on the wall geometry. As it can be seen on the graph and more clearly in table 4.3, from the 4th refinement step onward, the total fluxes at the opposing surfaces equal. This already lines up with one's physical expectations. From that point on, a steady convergence of the results can be observed. Looking at the last columns of table 4.3, it can be seen, that the change from step 6 to 7 is already less then 1 %; therefore, it is reasonable to assume that the true solution of the analytical model is very close to the one obtained with the resolution in step 6. Therefore, from here onward, all the simulations, where it was possible, will have at least 100 elements per unit length, so that the same level of refinement is ensured. This includes the numerical experiments done in the following sections, section 4.2 and section 4.3.

Table 4.3: Refinement steps with the resulting fluxes.

Ref. step	Elements					Total Fluxes $[W/m^2]$		Change [%]	
	x	y	z	1 / m	DoFs	T_{BC1}	T_{BC2}	T_{BC1}	T_{BC2}
0	57	10	4	30	20306	0.183389	0.183363	-	-
1	70	12	5	37	29529	0.180446	0.180445	1.60	1.59
2	85	15	6	45	52744	0.174751	0.174756	3.16	3.15
3	104	18	7	55	89664	0.176268	0.176267	0.87	0.86
4	126	22	9	67	164533	0.176971	0.176971	0.40	0.40
5	157	27	11	83	274371	0.172833	0.172833	2.34	2.34
6	188	33	13	100	465599	0.169598	0.169598	1.87	1.87
7	288	50	20	153	1348796	0.16805	0.16805	0.91	0.91
8	376	65	25	200	2801609	0.167119	0.167119	0.55	0.55

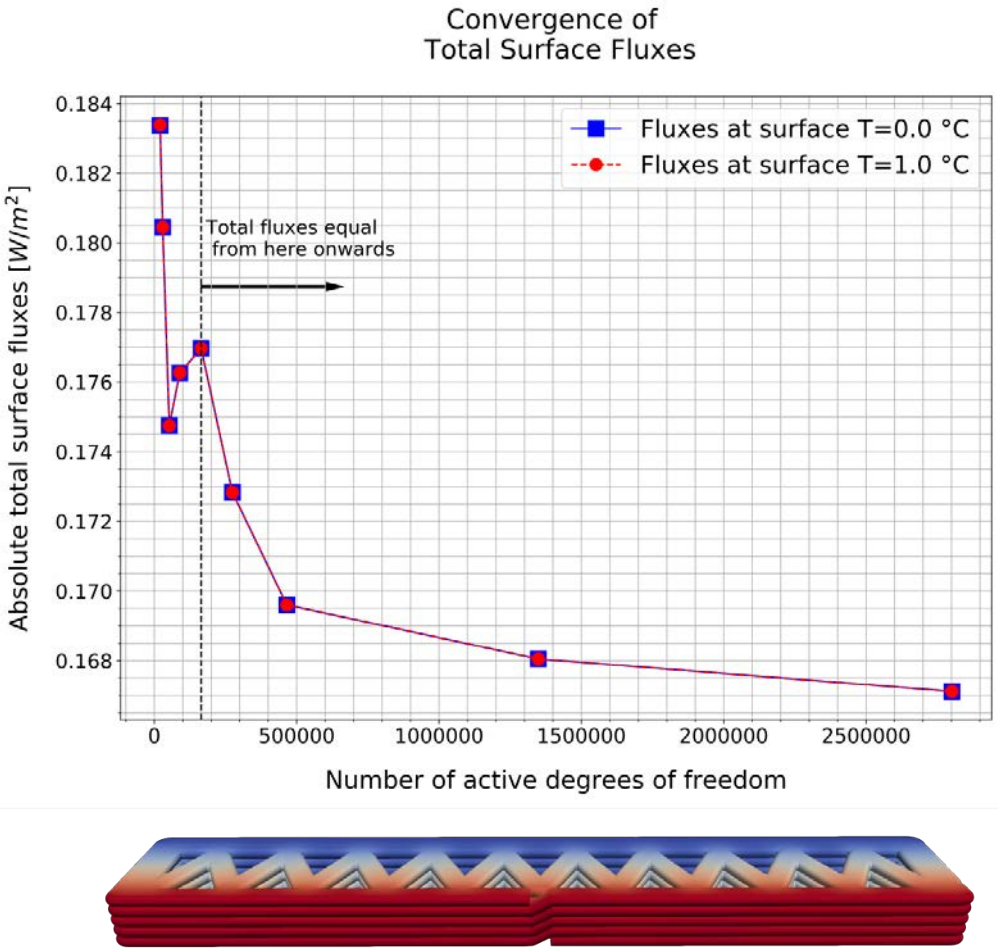


Figure 4.2: Convergence of total surface heat fluxes

4.2 Comparison to other geometric representations

After a sufficient resolution for the discretization was found in section 4.1, in this section the performance of the two way of creating CSG geometries (described in chapter 3) will be compared to the other two representation options mentioned in section 2.2, using the same wall geometry as in the previous section.

Since so far no implementation for generating STL or voxel wall geometries straight from the provided print paths exist, the ones used here are created from the available CSG representation. In case of the STL files this happened by generating the surface of the CSG geometry with the help of the marching cubes algorithm (LORENSEN & CLINE, 1987), and then the STL file was obtained from this with the help of ParaView (PARAVIEW DEVELOPERS, 2022) . For the voxel file, the data was generated with the help of the tool *binvox* (MIN, 2021), using the available maximum resolution (512 x 512 x 512) in the free version of it.

As mentioned in the introduction, to execute the repeated runs in a controlled manner and to obtain statistical evaluation of the results, the tool called *hyperfine* (HYPERFINE DEVELOPERS, 2022) was utilized. To not start the benchmarking simulations with an empty cache, a single ”warm-up” run was carried out in each geometric represantation’s case, followed by three regular executions. A high number of runs to gain reliable statistical data about the execution times was not feasible, since simulations with an appropriate resolution took long (couple of hours). Nevertheless, the three consecutive executions (without any other program running on the computer) can give enough confidence about the repeatability of the measured simulation times.

The input data and the measured execution times for the different geometries can be read in table 4.4. For the CSG geometries and the STL representation, the resolution of step 6 from table 4.3 was used, since it already showed sufficient convergence, as discussed in section 4.1. For the voxel geometry, this was not possible, since the elements have to be aligned with the voxel mesh (in this case each element being 4 voxels wide). A higher number of elements than this could also not be set, due to memory restrictions. Therefore, the polynomial degree was increased to get the closest number of degrees of freedom possible, but the obtained model still differed significantly from the previous cases. Nonetheless, this resolution was still good for seeing the overall tendencies, when comparing the execution times on fig. 4.3.

Even with the significantly higher number of degrees of freedom, the voxel simulation executed the fastest. But the accuracy of the geometric description achievable with the given memory restrictions was significantly lower compared to the other options, apparent from the depictions at the bottom of fig. 4.3. The second best performing option was the CSG representation using the periodic primitives, closely followed by the STL description. The CSG tree, built solely from individual sweep primitives, performed very poorly in comparison. I took almost double the time to execute than the STL geometry, next in the performance ranking. A further, more detailed assessment of the different options examined here will be given individually in sections section 4.2.1, section 4.2.2 and section 4.2.3.

Table 4.4: Run times with different representations of the geometry

Geometry	t $[s]$	$\sigma_t[s]$	p	d	Elements				Active DoFs
CSG Sweep	16813	55			x	188	Total	Active	
CSG Periodic	7330	5	3	4	y	33	80652	57638	465525
STL	8728	24			z	13			
VOX	2509	58	5	4	x	130	Total	Active	
					y	130	2197000	19592	578559
					z	130			

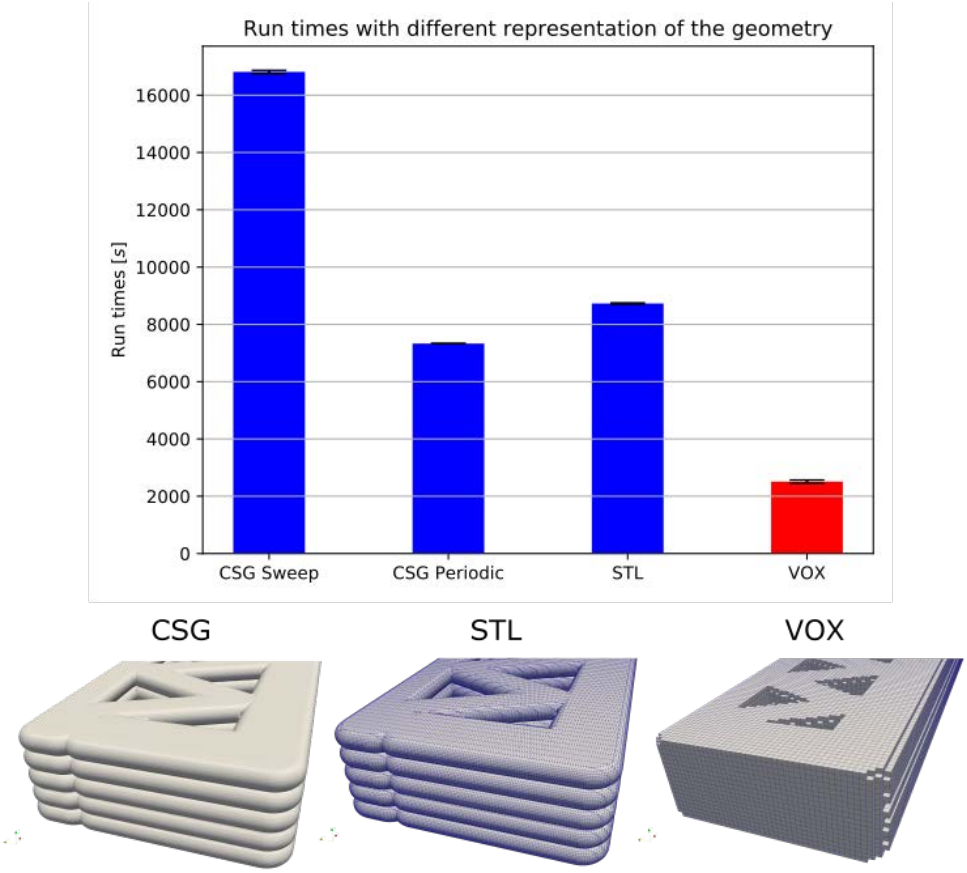


Figure 4.3: Run times for different representations of the geometry

4.2.1 CSG representation

In general, it can be stated for both of the CSG approaches, that they can be created without the need of storing large data files, like it is the case for the ASCII STL files used, but specially compared to voxels. Furthermore, since they only need the path input data file for geometry generation, as described in section 3.3, regardless of the required accuracy of the geometry, a better representation of the model does not come at the expense of file size and memory usage.

While the geometry built up from sweeps only, performed rather poorly, the one created from the periodic primitives achieved the second best execution times in this comparison study. Concerning the significantly higher accuracy of the geometric representation compared to the voxel case, makes it a viable alternative to the voxel representation, especially for personal computers, where the larger capacity in terms of memory required by a finer voxel model is usually not present. Finally, the CSG approach implemented in the scope of this thesis still shows decent potential for improvements (see suggestions in section 6.2), which could decrease execution times further.

4.2.2 STL representation

Looking at fig. 4.3, the STL representation still provides comparatively good execution times, close to the ones achieved with the periodic CSG domain. But unlike in the CSG case, representing the geometry as an STL surface comes at the cost of more storage required. Specially with the ASCII version of STL used in AdhoC++ (CMS, 2022) , and a sufficiently refined geometry, the file sizes can get large. This also comes with an increased time of parsing it at the start of the simulation, but the process was still significantly faster, than in case of the voxel files.

Even though an implementation for obtaining STL wall geometries directly from the print path representation has not been done yet, it could be achieved in a similar manner as with the CSG case. By moving a cross section along the print path, a surface could be directly generated as well. However, special care would have to be taken to only generate the outer surface of the entire geometry and not the surfaces at the layer contacts.

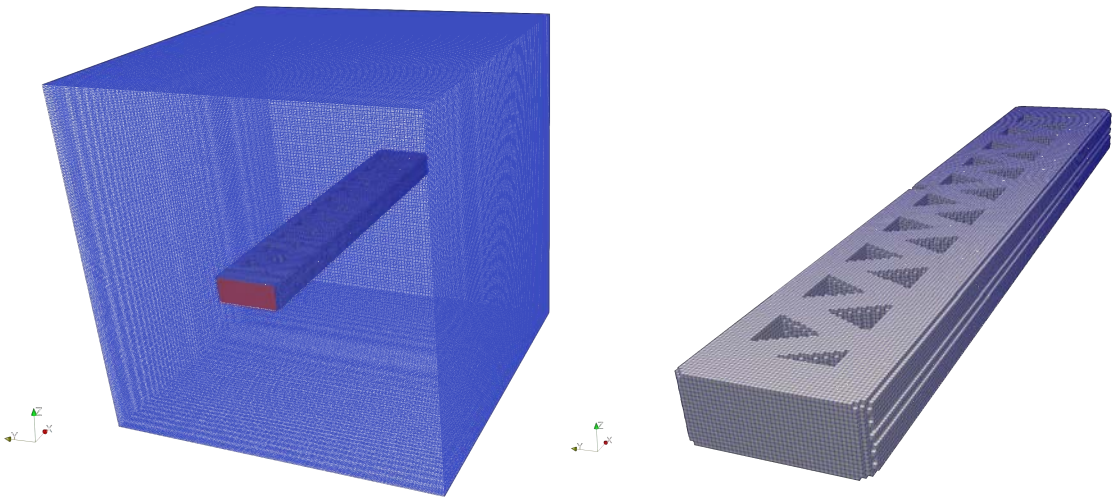
4.2.3 Voxel representation

Compared to the CSG representation and the analogous approach for the STL case with swept surfaces mentioned in the previous section, the generation of a voxel model directly from the FIM provided print path description does not seem to be straight forward.

A further drawback of the approach is the huge file size it comes with, specially in case of a sufficiently refined geometry. With the 512 x 512 x 512 resolution used here, a significant part of the execution was already spent with reading the file in. But file sizes like this can already exhaust the memory capacity of personal computers; therefore, making the complete simulation infeasible in the first place, as it happened in this study as well, when this large file size was combined with a higher finite cell resolution than the one given in table 4.4.

Another shortcoming of the voxel representation is, that the number of finite cells defined for the discretization in each dimension, has to be aligned with the voxel resolution. Meaning, that cells can not be defined arbitrarily, their boundaries must align with the voxel grid. Furthermore, when generating the discretization, the entire voxel domain has to be meshed first, and then the elements not containing anything from the actual geometry get filtered out. Looking at the visualization of the voxel geometry in fig. 4.4, it is very apparent, that this approach will lead to plenty of elements being created, just to be filtered out later on by a significant amount of computational effort. The extent of this issue can be seen in table 4.4, where the finally obtained active elements in the mesh only make up about 1 % of all the finite cells created initially. This severe inefficiency, can be mostly attributed to the voxelizer tool used. Finding another option, which does not have to create the same amount of voxels in every direction, but can adjust it according to the dimensions of the geometry, could significantly improve on this situation.

But the voxel representation offers significant benefits as well. Thanks to the efficient integration technique implemented for these geometries by YANG et al. (2012), simulations with this representation achieved much faster run times than the second best performing periodic CSG approach. Furthermore, the voxel representation would offer a convenient way to model the material of the printed wall more precisely, by the possibility of assigning material constants voxel-wise.



(a) (b)

Figure 4.4: The actual geometry of the wall (b) takes up only a small portion of the entire voxel domain (a)

4.3 Comparison of adaptive octree and moment fitting

In the final numerical experiment, the two integration methods for FCM, introduced in section 2.4.3, were compared. For running the simulations, again, the data from refinement step 6 (table 4.3) in the convergence study was taken. Only the integration methods were set differently, with each of them having the standard number of points per dimensions defined within an integration mesh element. The results can be compared with the help of table 4.5.

Table 4.5: Integration schemes comparison

Integration scheme (points)	Number of integration points	Integration times
Adaptive octree (p+1)	570065482	1h 56min 28s
Moment fitting (2p+1)	23660642	1h 56min 18s

It was found, that even though moment fitting uses significantly less integration point to obtain the same results, the total integration times for both options are practically the same. An explanation for this could be that - as it was described in section 2.4.3 - for defining the integration points and weights for moment fitting, the integrals at the right sight of the moment fitting equations still have to be calculated first. In a general case, this also happens with the help of composed integration, which - depending on the complexity of the integrals - can take significant computational effort, eliminating the gains achieved by reducing the number of integration points at the very end.

However, moment fitting still could offer significant benefits, if the same simulation needs to be re-runned (e.g., with different material parameters). In this case the integration points and weights calculated once, could be reused in the consecutive executions, reducing the integration times significantly, compared to running the adaptive octree method every time.

Chapter 5

Application examples

After determining the refinement requirements (section 4.1) and examining the performance of the geometry generation method in comparison to other available representations (section 4.2), in this chapter a few examples will be given. The examples aim to illustrate the capability of the implemented tool to support design choices in different scenarios. In section 5.1, two simulations with different physical models will be detailed using complex wall geometries. While in section 5.2, the process of evaluating different design iterations based on the defined performance measures (section 3.6.2) will be showcased.

Since the main goal of these examples is to demonstrate the capabilities of the software and are not part of any real life project, doing a convergence analysis on each model will be omitted here. For the generation of the geometries the CSG approach with periodic primitives was taken in all cases.

5.1 Simulation on complex wall geometries

The two simulation types most relevant for the analysis of 3D printed walls are the thermal and linear elastic ones (for a detailed model description see section 2.3). In the following, first, a thermal problem will be demonstrated (section 5.1.1), using a free-form wall geometry. Then a linear elastic analysis will be showcased (section 5.1.2) using a corner wall geometry, where the individual layers are shifted in plane compared to each other. Due to very long execution times (section 5.1.1) and very high memory requirements (section 5.1.2), in both cases lower resolutions for the discretization were used, than it was determined as necessary in section 4.1. Therefore, the examples are here only to demonstrate the capabilities of the implementation, and to showcase the overall trends of the results, but they can not be considered as accurate solutions.

5.1.1 Thermal simulation on a free-form wall

For the free form thermal simulation, the main parameters are summarized in table 5.1. The conductivity of the material remained the same as in section 4.1, while the temperature difference between the two sides of the wall was increased to a level, which is more realistic to occur for a building. The element resolution (table 5.1) had to be set lower, than it would be required according to the convergence study done in section 4.1. The reason for this was the amount of time the simulation required to execute. The PMC test for spline-based primitives, from which significant part of the free-form geometry was built up, proved to be very slow. Even with this coarser mesh (table 5.1), it took more then a day for the

simulation to finish. For finer models, the computation could have taken several days to complete, which was unfeasible and in general not suitable for the aim of the application, i.e., running several simulations to optimize the print path design. Therefore, the results here just showcase the possibility of simulations on free-form wall designs, but due to the low resolution, they are by now means an accurate solution of the problem.

Table 5.1: Parameters for the free form wall simulation

Geom.	Dims. $[m]$	p	d	Elements			BCs $[°C]$		Material
CSG Periodic	x 4.03 y 0.986 z 0.125	3	4	x 323 y 79 z 10	1/m 80	DoFs 477722	T_{BC1} 20	T_{BC2} 0	$\kappa \left[\frac{W}{mK}\right]$ 0.38

The temperature field, even with this low resolution, properly captures the overall linear change from the warmer wall to the colder, as it can be seen on fig. 5.2. In general, it was observed, that this overall tendency of the primary field can be correctly recovered by any of the geometries, even with very coarse meshes. Less accurate results are expected from the derived field. However, looking at fig. 5.2, the overall trends in the heat flux field are still properly captured. Before elaborating on the heat flux field, though, it is worth noting, that only the heat conduction in the solid material was simulated, but not the conduction through the air or other means of heat transfer. Therefore, it is understandable why the flux magnitudes are close to zero between the support wall connection points (fig. 5.2). With the presence of air, this probably would not be the case, but a stronger flux through the support wall sections connecting the perimeters would still be expected. Finally, it is worth pointing out, that at each point, where the support wall touches the perimeter section, a sudden jump in the heat flux occur (see the magnified part of fig. 5.2). This flux concentration of the connection points can be attributed to the radical change in the conductive properties at those places.

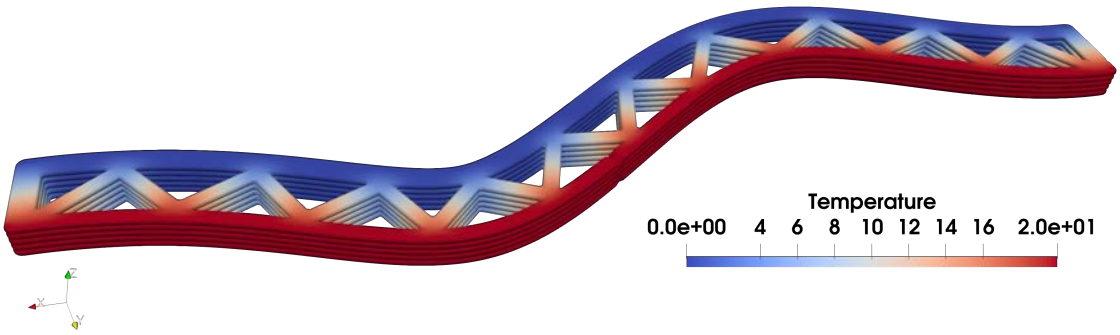


Figure 5.1: Temperature field

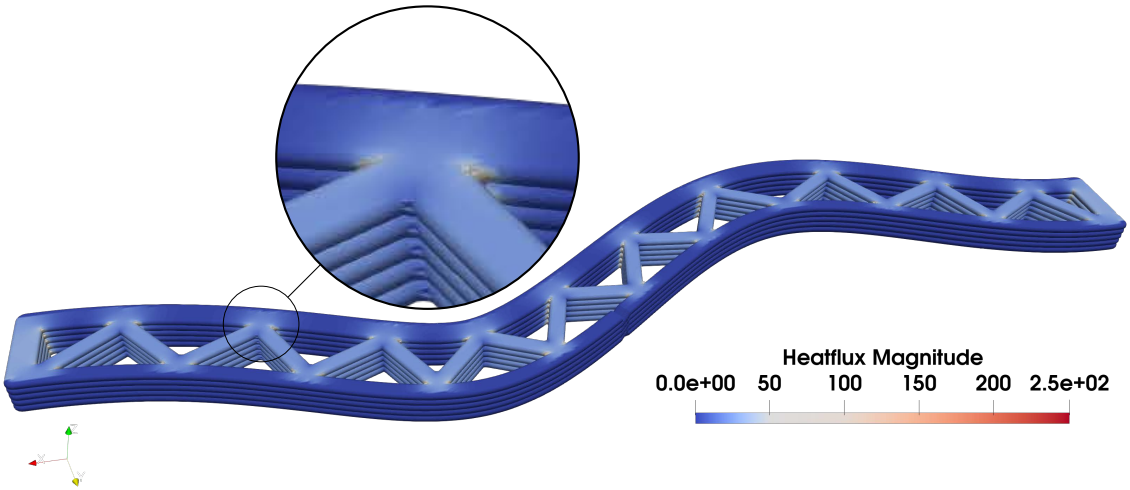


Figure 5.2: Flux magnitudes

5.1.2 Linear elastic static simulation on a slanted corner wall

In case of the slanted corner wall model, the material properties were taken to resemble a behaviour close to concrete, as indicated in table 5.2. At the bottom of the wall, zero displacements were prescribed, while for the top, a total vertical load of 500 $[kN]$ was distributed over the boundary surface, ending up with the distributed load value given in table 5.2. The load value was chosen to be safely below the failure loads measured for 3D printed concrete walls by HAN et al. (2022), where the assumption of linear elasticity (based on the load-vertical displacement curves provided in the paper) still holds. For the finite cell mesh, due to the high memory demand of the model, only a 60 elements per unit length resolution could be taken. Considering, however, that for a linear elastic problem three displacement scalar fields need to be approximated, instead of a single one for temperature, it is understandable, that only a coarser mesh of the three can be stored with the same capacity. Comparing the number of degrees of freedom given for the elastic problem in table 5.2 and the data from the convergence study in table 4.3, one can see, that the elastic case, even with this lower mesh density, already exceeds the number of degrees of freedom present in the thermal model, which already converged (step 6).

Table 5.2: Parameters for the slanted wall simulation

Geom.	Dims. $[m]$	p	d	Elements			BCs		Material	
CSG Periodic	x 2.06 y 2.06 z 0.125	3	4	x 124 y 124 z 8	1/m 60	DoFs 769977	$t[Pa]$ 617897	$u[m]$ 0	$E[Pa]$ $25 \cdot 1e6$	$\nu[-]$ 0.2

Nevertheless, the resolution is significantly below the one necessary, determined by the convergence study (section 4.1). Therefore, only the overall trends in the vertical displacements depicted in fig. 5.3 will be discussed here. Strains and stresses are linked to the derivatives of the primary field, so with a coarse mesh like this, their accuracy is more than questionable. For this reason, a discussion of those results will be omitted here.

With regard the vertical displacement field, there are two main trends to be observed on fig. 5.3. Looking at the side view in the middle, it is apparent that the magnitude of the displacements decreases with the z coordinate. This behaviour is expected from more classical geometries as well, with similar boundary conditions. The more geometry specific behaviour is the difference in vertical displacements in the layer plane. Looking at the side view on fig. 5.3, due to the slantedness of the geometry, the same load is able to push the upper right section of the wall downwards more, since there is just simply less supporting material below it. Therefore, when planning a building with such wall geometries, special care must be taken how the load over the wall is distributed, to avoid significant displacement differences over the structure.

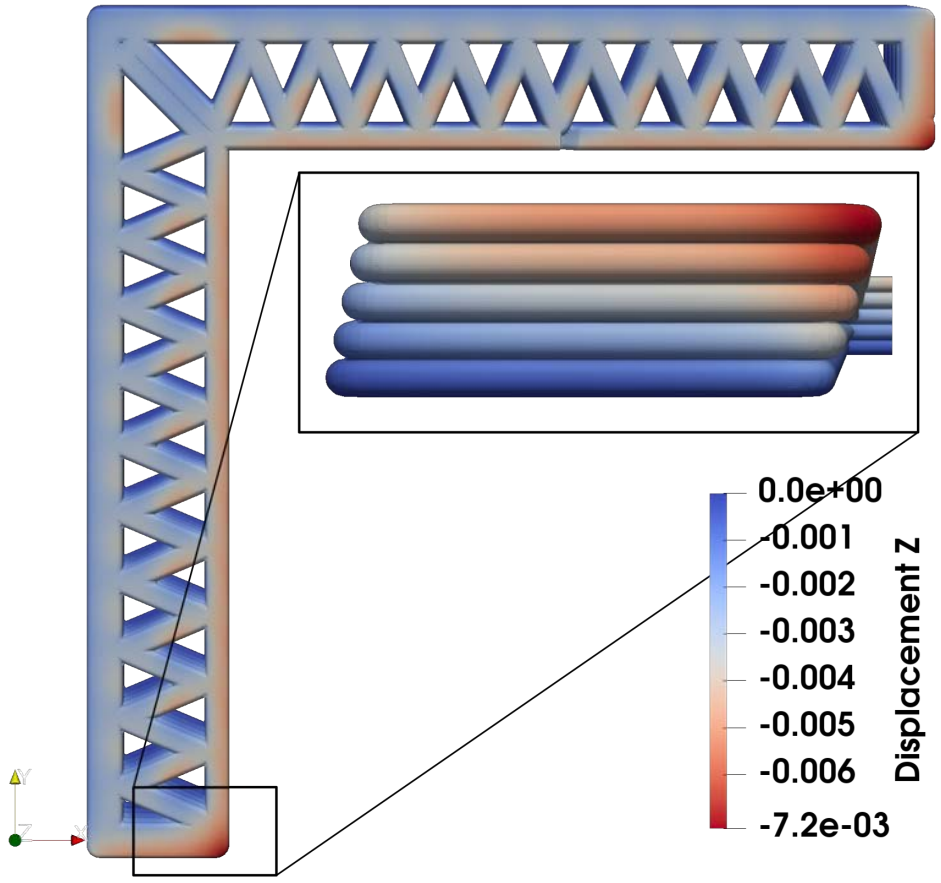


Figure 5.3: Vertical displacements of the wall

5.2 Thermal simulation for finding the best support wall design

Since with the current implementation, extracting the boundary surfaces of a geometry is labour intensive, manual work (see section 3.5), the best use cases for the tool are simulations, where engineers are only interested in the effects of varying the inner support structure of the wall, without changing the perimeter. Then, the surfaces for applying the boundary conditions need to be extracted only once and it can be used to analyze all kind of patterns for the inner support structure to find the most suitable one for the application.

Such a case for studying the heat conductive properties of a wall will be demonstrated here as well. On fig. 5.4, the same wall design with 6 variations of the inner support structure can be seen. Here, for the sake of simplicity, only the density of the pattern was varied, but not the shape. All other parameters for the 6 simulations were the same, which can be read in table 5.3. The number of elements were taken based on the convergence study done in section 4.1, where the 100 elements per unit length resolution has already proved to be converged. The temperature difference between the two sides of the wall was taken again as 20 $^{\circ}C$, and the conductance remained the same as well, as in case in section 5.1.1.

Table 5.3: Parameters for design iteration simulations

Geom.	Dims. $[m]$	p	d	Elements		BCs $[^{\circ}C]$		Material
CSG Periodic	x 1.02 y 0.32 z 0.10	3	4	x 102 y 32 z 10	1/m 100	T_{BC1} 20	T_{BC2} 0	$\kappa \left[\frac{W}{mK}\right]$ 0.38

As it is apparent from fig. 5.4, the change in the inner wall pattern design did not alter the basic linear nature of the temperature distribution. However, looking at table 5.4, it can be seen that it had a strong effect on the total heat flux. The values in the table were determined at the boundary, where the 20 $^{\circ}C$ Dirichlet condition was applied. The results support one's natural expectations, that the larger the total cross section area of the connection between the two sides of the wall, the more heat can flow through. Therefore, it was found that the first design would serve the best from a heat insulation perspective.

One must keep in mind though, that the simulation has only considered the conductive heat flow through the solid material, i.e., the heat flow through the air was neglected, as well as other means of heat transport. These all can have a significant impact on the final heat insulation capabilities of the wall; therefore, a more elaborate simulation could be necessary before making a design decision.

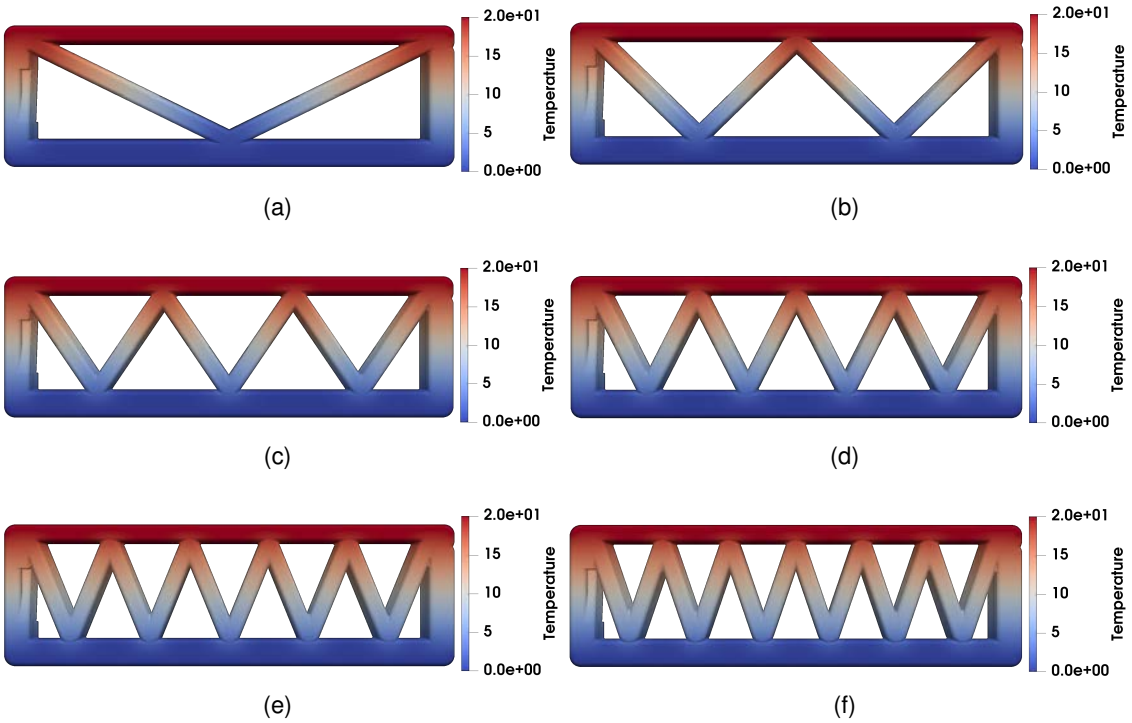


Figure 5.4: Support wall pattern design placeholder

Table 5.4: U-Values determined based on the surface for the first boundary condition for different inner wall patterns

Pattern	1	2	3	4	5	6
Total Flux $[W]$	0.42	0.60	0.78	0.95	1.10	1.24
U-Value $\left[\frac{W}{m^2K}\right]$	0.16	0.23	0.30	0.36	0.42	0.47

Since this example aimed to merely demonstrate an intended use case for the tool, the results themselves were not particularly interesting ones from a design engineer's perspective. A more relevant question could have been to find a good compromise between the heat insulation properties of the wall and its load bearing capacity. But such a multi-objective optimization problem would already lead too far from the main topic of this work.

Chapter 6

Summary and outlook

6.1 Summary

In this final chapter of the thesis, first, in section 6.1.1, it will be summarized what the implemented analysis tool can achieve and how it supports the FIM design workflow addressed in section 1.2. Even though the tool already provides a working option for creating computational models of the 3D printed wall geometry, directly from the print path information, it still has room for improvement at multiple points, which will be addressed in section 6.1.2. Finally, in section 6.2 a couple of suggestions will be made, how some of the shortcomings of the current implementation could be overcome.

6.1.1 The benefits of the implementation for the design workflow

Looking at the results presented in chapter 5, it can be stated that the first significant step was made to integrate the capabilities of AdhoC++ (CMS, 2022) into the 3D printed wall design loop. The implemented tool already makes it possible to perform numerical analysis on complex geometric shapes (section 5.1.1 and section 5.1.2), and to evaluate different design iterations of 3D printed walls, based solely on the print path generated by the FIM model.

The generation of wall geometries with the Constructive Solid Geometry (CSG) approach using periodic primitives, as described in section 3.4.3 and section 3.4.4, proved to be a viable alternative to existing other geometric representations for conducting numerical simulations (section 4.2). In fact, concerning that a fine enough voxel resolution would already exhaust the memory capacity of personal computers, this newly implemented approach seems to be a very promising option for desktop simulation environments with regard the execution time. Furthermore, compared to the overall faster voxel approach, it allows for greater flexibility in choosing the discretization, and geometric information can be obtained with an arbitrary fine level, without the need for the recreation of the model.

Finally, the implementation closes the simulation optimization loop discussed in section 1.2, and depicted on fig. 1.4. It takes a print path generated by the FIM model to be translated for the 3D printer specific machine control program, it creates a volumetric model of the geometry ”as-planned”, on which numerical simulations can be carried out, and then finally, it calculates an overall physical quality measure for the geometry, based on which the design can be updated.

6.1.2 Shortcomings of the application

Even though, as it was elaborated on in the previous section (section 6.1.1), the analysis tool created already managed to bring the AdhoC++ (CMS, 2022) framework's simulation capabilities into the design workflow of 3D printed walls, there were multiple points, where problems or bottlenecks were encountered which could not be solved within the scope of this thesis. In the following, these issues will be mentioned.

One major shortcoming from the input file's side for the current application was, that due to certain compatibility problems within the tool chain that was used in the work of SLEPICKA (2021), end points of curves get often created with different accuracy. Therefore, it happened rather often that end points were found unconnected at a pre-determined level of accuracy for the floating point numbers in the data, despite being created by the tool chain as connected points. For these cases (couple of points usually in some of the files) the accuracy of connection checks had to be lowered. In general, either the compatibility issues need to be resolved (which would be rather hard due to the different code bases of the used software, plenty of them proprietary) or the accuracy of the checks would need to be adjusted dynamically at problematic points (on which one has more influence, but it is not a trivial task to do it in a proper manner). Nevertheless, this was the available way of getting the path information out of the design software and with the knowledge of the shortcomings of it and some extra work and code on healing the provided path, reliable simulation results could be achieved as it was demonstrated in chapter 5.

The process of the wall geometry generation is already a highly automated one, with the notable exception of the definition of surfaces for applying the boundary conditions (section 3.5). The task of extracting the necessary boundaries can take hours for complicated geometries and the process is quite error prone. Accidentally selecting some unwanted small triangles at an other part of the geometry will lead to applying the boundary condition at these as well, which can lead to physically unreasonable results in the simulation and a lot of working hours spent on finding the cause of error. Therefore, in the future it is desired to get the surface extraction process more automated. One suggestion for this is given in section 6.2.3.

Considering the number of elements needed to obtain reliable results for a computational model, the current version of the implementation already runs within a reasonable time frame for geometries based on lines and arcs. However, if robust optimization algorithms were to be utilized for finding best path designs according to certain criteria, even moderately faster execution times could make a big overall difference due to the high number of repeated runs. In the follow up, in section 6.2.1 and section 6.2.2, two suggestions to speed up the PMC tests are given.

In case of spline geometries, as it was experienced in section 5.1.1, execution times proved to be significantly higher. In fact, due to the costly nature of the point inclusion tests of spline-based sweeps, simulations took long enough to state, that in this form, they are not a practical geometric representation for simulations in a desktop environment.

Generally restrictive was on simulations with fine meshes the amount of memory they require. This was especially apparent in the case of the linear elastic simulation on the slanted corner wall geometry (section 5.1.2). There, due to the fact that three displacement component fields had to be approximated, simulations with a sufficiently refined mesh already exhausted the available RAM (31 GiB) on the machine used.

Finally, it must be acknowledged, that the workflow has not been tightly integrated with the FIM model yet and the communication between the two happens in a text file based format. This was necessary, since both tools are still under development; therefore, a rather decoupled approach for connecting the two was preferred to avoid unnecessary work with updating a complex interface (possibly) after each design iteration. However, after both systems matured, a tighter integration would provide the user with a more seamless workflow, excluding steps where mistakes can potentially be made, and cutting on the overall working hours needed for each design iteration loop.

6.2 Possible options to improve on the tool's shortcomings

Overall, there are certainly a plethora of possibilities to improve on every aspect of the implementation, bounded only by one's ingenuity, knowledge and available time. However, in this section three specific ideas will be suggested which carry the potential for significant speed-ups in performing the PMC tests (section 6.2.1 and section 6.2.2), and reducing the required working hours to set up a simulation (section 6.2.3).

6.2.1 Interval tree for affected bounding box identification during PMC

During the implementation of the geometry generator algorithm, it was observed, that there are two important characteristics of the the current application case that could be made use of to speed up the point inclusion tests:

1. The current CSG approach does not make explicit use of the position of the point to find the primitives most likely containing the point.
2. The geometry is built up from a union of primitives only. Therefore, if the point is included by a primitive it can be already marked as part of the geometry, no additional considerations have to be made regarding the effects of other logical operations, e.g., differences of domains. Which means that containment by the geometry comes down to the question, whether any of its primitives contain the point, and no additional logic for the combination of the primitives has to be considered.

Therefore, the goal would be to write an algorithm, which quickly identifies all the bounding boxes the tested point is included in, and then only call the PMC test on those, one after the other, until either one test returns ”true”, or all the tests conclude with a negative result.

After some research done in the field of space partitioning and other tree structures, it was found, that by building a three dimensional interval tree (TAMASSIA, 1992) based on the lower and upper bounds of the bounding boxes of the sub-geometries during the geometry generation phase, such a point inclusion test could be implemented. On fig. 6.1 a two dimensional example for defining such intervals for the bounding boxes is given. In the case of point A for example, when calling the PMC subroutine, it would find all the bounding boxes, which contain the point in question by checking if both of their intervals contain the point. In case of this example, e.g., the algorithm would find the boxes 1 and 2. Then the point inclusion tests of the primitives would be performed from smallest bounding box to largest.

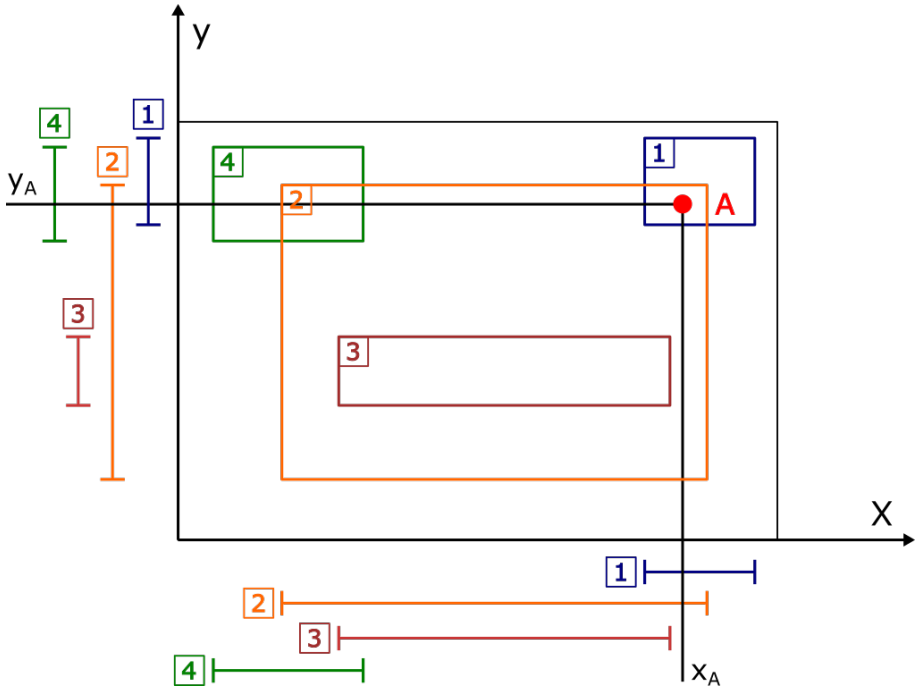


Figure 6.1: Point inclusion with intervals of 2D bounding boxes

The implementation of such a PMC algorithm would offer the following benefits:

1. The maximum number of point inclusion test calls per point would be guaranteed to not exceed the number of bounding boxes the point is included by. In case of the CSG approach, while traversing the tree, several bounding box check can happen in vain. For a huge number of integration points that can add up, even though bounding box checks are not that expensive computationally.

2. Evaluation of bounding boxes from smallest to largest: The smallest bounding box has the smallest likelihood of all to contain a randomly selected point, so once it does, there is a good chance that its contained geometry does as well, compared to large bounding boxes with a lot of ”empty space” around their contained geometry, like the example in fig. 6.2.

3. Combining this approach with the periodic domain CSG tree generator, instead of sweeps, could speed up the tests even more, since the PMC test for them is not significantly more costly then for a basic sweep, but it can significantly reduce the number of primitive geometries present in the model and therefore the size of the interval tree, which would speed up finding the right bounding boxes in return.

6.2.2 Usage of tighter bounding boxes

The definition of a bounding box around a more complex geometric shape is beneficial for PMC tests, since the expensive inclusion test of the contained geometry does not have to be called, if points were already found outside of the bounding box (which is a computationally less expensive check to perform). However, especially in the case of large arcs being present in the geometry (e.g., the one depicted on the left of fig. 6.2), plenty of outside points can be contained by an axis aligned bounding box, making it as a cheap filter much less effective. The situation could be improved by applying slightly more complex, but much tightly containing polytope bounding boxes, as depicted on the right side of fig. 6.2. Such polytopes could filter out significantly more uncontained points, while being just a slightly more expensive computationally. Such Discrete Oriented Polytopes were used in dynamic collision detection problems (DANIEL S. COMING, 2007) with good results, so given that field of application it should be fast enough for this situation as well.

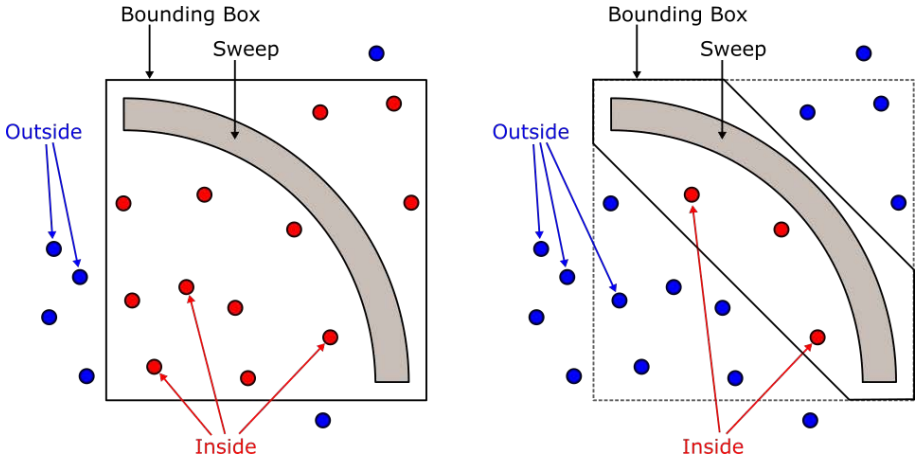


Figure 6.2: The advantage of a tighter bounding box

6.2.3 Automating extraction of surfaces for boundary condition application

As it was mentioned in section 6.1.2, one of the bottlenecks in setting up a simulation with the current tool, and source of possible errors, is the manual nature of extracting surfaces for the application of boundary conditions. Automating this task would result in significant speed-ups in the workflow and would make simulations on design iterations with changing outer perimeter much more feasible.

The following suggestion tries to achieve this by providing extra information in the input data file about curves running along boundary condition surface segments and then use these curve sections and the cross section of the geometry to get the desired surface automatically via a sweeping operation. This idea is depicted on fig. 6.3. With each curve, which runs along the boundary of the wall, where boundary conditions will have to be applied, could have extra information provided about:

1. At which parameter value of the curve the surface starts and at which value it ends. (Curve parameters defined in the interval [-1.0, 1.0].)
2. In its local coordinate system it could be defined, whether the surface to be extracted is in the positive or negative direction according to the normal vector $\mathbf{n}$ (or according to $\mathbf{b}$ if e.g. vertical forces wanted to be applied).

Since the cross section in the current application is build up from 4 curves (2 arcs at the sides and 2 lines at top and bottom), based on the listed information above, it could be already decided, which of the curves is part of the boundary surface (e.g., the curves highlighted in green in fig. 6.3), and then it could be swept along the curve from the defined starting parameter till the ending one (as depicted in the bottom part of fig. 6.3). This swept surfaces, created curve-by-curve, then could be unified into the desired one. In case of slanted wall geometries (see upper right part of fig. 6.3), the situation would be only slightly more complicated, with some additional intersections of the cross sections of different layers needed to be determined before the sweeping operation could take place.

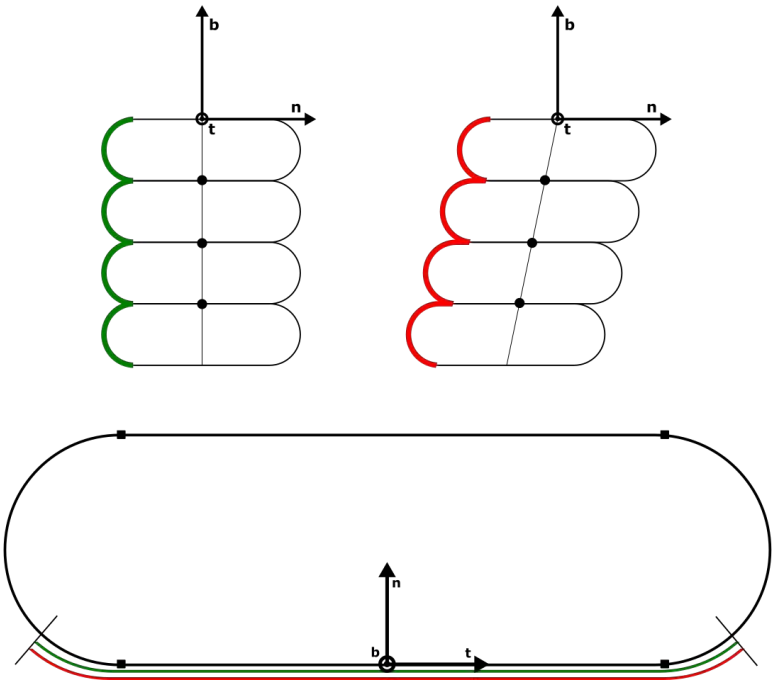


Figure 6.3: Idea for automated boundary extraction

6.3 Final words

These last few suggestions would certainly improve the current implementation significantly, but as it was elaborated on in section 6.1.1, this version of the tool can already achieve several of the initial goals. It can generate the 3D printed wall geometries from the print path data file (chapter 3). Simulations can be run with it in reasonable execution times (section 4.2) and with complex geometries (section 5.1.1 and section 5.1.2). Furthermore, design variations can be examined (section 5.2) and qualified based on simple scalar performance measures (section 3.6.2), which potentially can be fed back into an optimization loop via a text based output file. Therefore, the implemented tool proved to be a decent first step towards a seamless integration of numerical simulations into the Fabrication Information Modeling design workflow, which would be worthy of further development.

Bibliography

CMS. (2022). *Adhoc++*. https://gitlab.lrz.de/cie_sam/adhocpp

ESA. (2013). *Building a lunar base with 3d printing*. Retrieved May 19, 2022, from https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Building_a_lunar_base_with_3D_printing

GE®. (2020). *The future of renewable energy*. Retrieved May 19, 2022, from https://cobod.com/projects-and-partners-ge/#

ALL3DP. (2021). *The stl file format – simply explained*. Retrieved May 18, 2022, from https://all3dp.com/1/stl-file-format-3d-printing/

DANIEL S. COMING, O. G. S. (2007). Velocity-aligned discrete oriented polytopes for dynamic collision detection. *IEEE Transactions on Visualization and Computer Graphics*, *14*(1), 1–12.

FELIPPA, C. A. (2004). *Introduction to finite element methods* (Vol. 885). Boulder CO.

HAN, X., YAN, J., LIU, M., HUO, L., & LI, J. (2022). Experimental study on large-scale 3d printed concrete walls under axial compression. *Automation in Construction*, *133*, 103993. https://doi.org/https://doi.org/10.1016/j.autcon.2021.103993

HOLZAPFEL, G. A. (2000). *Nonlinear solid mechanics: A continuum approach for engineering* (Vol. 455). Wiley.

HTFLUX DEVELOPERS. (2022). *HTflux Documentation*. Computer Software. HTflux Engineering GmbH. Franziskanerplatz 11, 8010 Graz – Austria. https://www.htflux.com/en/documentation/u-value-tool/

HUBRICH, S., STOLFO, P. D., KUDELA, L., KOLLMANNSBERGER, S., RANK, E., SCHRODER, A., & DÜSTER., A. (2017). Numerical integration of discontinuous functions: Moment fitting and smart octree. *Computational Mechanics*, *306*, 1–19. https://doi.org/10.1007/s00466-017-1441-0

HYPERFINE DEVELOPERS. (2022). *Hyperfine*. Retrieved May 22, 2022, from https://github.com/sharkdp/hyperfine

KRUMM, J. (2000). Intersection of two planes. *Microsoft Research, Research note*.

KRYSL, P. (2010). *Thermal and stress analysis with the finite element method*.

KUDELA, L., ZANDER, N., KOLLMANNSBERGER, S., & RANK., E. (2016). Smart octrees: Accurately integrating discontinuous functions in 3d. *Computer Methods in Applied Mechanics and Engineering*, *306*, 406–426. https://doi.org/10.1016/j.cma.2016.04.006

LIOU, F. W. (2007). *Rapid prototyping and engineering applications*. CRC Press.

LORENSEN, W. E., & CLINE, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, *21*(4), 163–169.

MIN, P. (2021). *Binvox*. Retrieved May 22, 2022, from https://www.patrickmin.com/binvox/

PARAVIEW DEVELOPERS. (2022). *ParaView Reference Manual*. Computer Software. Kitware. Clifton Park, New York, USA. https://docs.paraview.org/en/latest/ReferenceManual/index.html

PARVIZIAN, J., DÜSTER, A., & RANK, E. (2007). Finite cell method. *Computational Mechanics*, *41*, 121–133. https://doi.org/10.1007/s00466-007-0173-y

PERI®. (2020). *Peri builds the first 3d-printed residential building in germany*. Retrieved May 19, 2022, from https://www.peri.com/en/media/press-releases/peri-builds-the-first-3d-printed-residential-building-in-germany.html

PERI®. (2021). *Germany´s first printed house officially openend*. Retrieved May 19, 2022, from https://www.peri.com/en/media/press-releases/germanys-first-printed-house-officially-openend.html

PITZL, D. (2018). *Bounded optimization by quadratic optimization* [Github repository]. https://github.com/pitzl/bobyqa

POWELL, M. (2009). The bobyqa algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, *26*.

SLEPICKA, M. (2021). *Fabrication information modelling – bim-basierte modellierung von fertigungsinformationen für additive manufacturing* (Master's thesis). Technische Universität München. München.

TAMASSIA, R. (1992). *Point enclosure and the interval tree*. Retrieved May 22, 2022, from https://cs.brown.edu/courses/cs252/misc/resources/lectures/pdf/notes06.pdf

WASSERMANN, B., KOLLMANNSBERGER, S., BOG, T., & RANK, E. (2017). From geometric design to numerical analysis: A direct approach using the finite cell method on constructive solid geometry. *Computers & Mathematics with Applications*, *74*(7), 1703–1726.

WASSERMANN, B. (2020). *Direct simulation on geometric representations with the finite cell method* (Doctoral dissertation). Technische Universität München. München.

YANG, Z., RUESS, M., KOLLMANNSBERGER, S., DÜSTER, A., & RANK, E. (2012). An efficient integration technique for the voxel-based finite cell method. *International Journal for Numerical Methods in Engineering*, *91*(5), 457–471.

YANG, Z. (2011). *The finite cell method for geometry-based structural simulation* (Doctoral dissertation). Technische Universität München. München.