Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Semester Thesis

Simulations on Minimal Surfaces

Spring Term 2020

Supervised by:
Prof. Dr. Ralf Hiptmaier

Author:
Lea Fritschi

Contents

Chapter 1

Background

1.1 ADMS

Recently, the company Spherene AG [1] has developed so called adaptive density minimal surfaces, ADMS for short. An example ADMS can be seen in Figure 1.1. ADMS are thin, printable structures that take the outer form of normal everyday objects, such as cylinders. However, in comparison to these everyday objects, ADMS are easily printable without support nodes, they need less material and also have lower printing times. Nevertheless, they achieve high stability and other desirable mechanical properties. These properties and the cell-like structure of ADMS allows for the integration of ADMS in many different scientific fields. Examples include the possibility of ADMS bone implants or creating light, crash resistant fuel tanks that can be sent into space.
The goal of this thesis is to create heat flow and stiffness simulations for these thin ADMS.

1.2 Heat Computations

The heat equation, also known as the diffusion equation, is one of the most studied PDEs in mathematics. It describes the temperature field of a given object given some boundary conditions. The temperature field can either be varying over time (non stationary) or independent of time (stationary). While analytical solutions exist for special objects, such as the annulus, they are usually not available for more complex structures such as the ADMS surface described above or even a simple object such as a square. Therefore, the heat equation is often solved using numerical methods such as the finite element method. In this thesis, we will implement FEM to solve the heat equation on the 2D ADMS surface in 3D space.

1.3 Elasticity Computations

An interesting property of every physical body is its reaction to external forces. Usually, objects either react with elasticity or plasticity, meaning that they deform and go back to their initial state once the external force is removed, or that they break. In this thesis we will only study the behavior of elastic materials.
Elasticity is often measured in terms of the displacement, stress and strain a body experiences in response to external forces. In the case of simple 3D structures, these properties can be computed by solving the linear or finite elasticity problem. However, this is not a feasible approach when working with thin, elastic structures

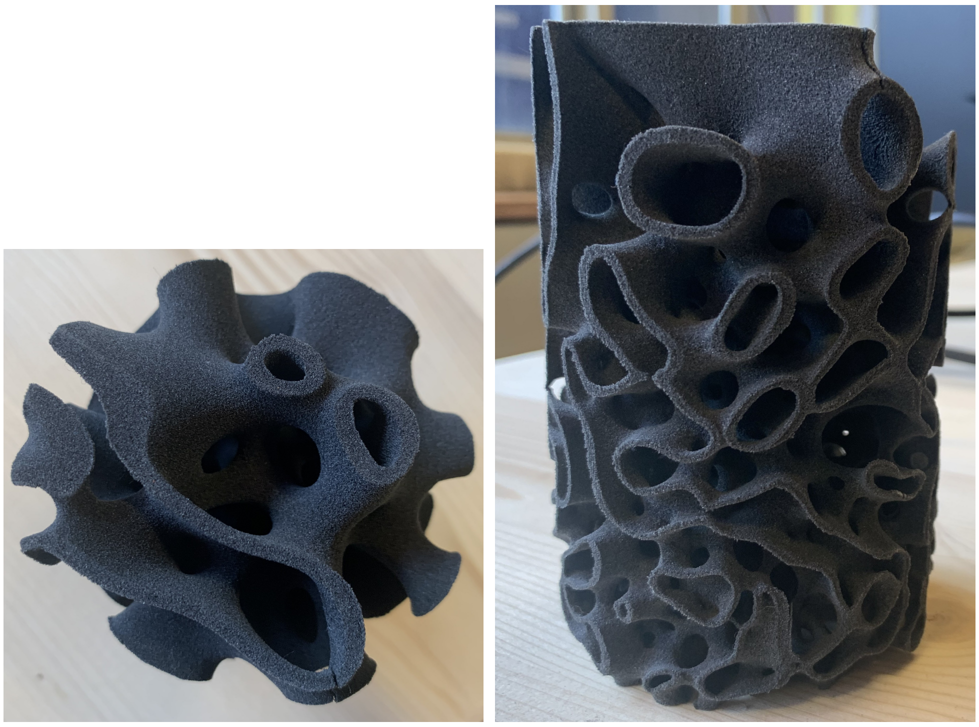

Figure 1.1: Example of a printed, cylindrical ADMS

such as thin metal plates, clothing or the ADMS surfaces described above.
In the next sections, we will shortly describe the linear elasticity assumptions and give sources for further reading, before detailing the models used with thin, elastic structures, such as thin plates or thin shells.

1.3.1 Linear and Finite Elasticity

The most simple model used to describe the behavior of 3D structures is the model of linear elasticity [2]. In this model, one assumes that a structure only experiences small displacements and rotations. One furthermore expects the stress to be proportional to the strain of a model, which is generally known as the Young's modulus. Under these assumptions, the elasticity problem breaks down to a set of a tensor PDEs that can be solved using the finite element method. An extensive treatment of FEM for Linear Elasticity can be found in many standard textbooks, for example in a book published by Falk in 2008 [3].
However, for soft materials such as rubber, these assumptions usually do not hold. One therefore has to solve the more complex finite elasticity problem. Many models for the treatment of such materials have been developed, some of which are discussed in a book on the treatment of polymers by Bergström [2].

1.3.2 Thin Plates

Thin plates are a phenomena often encountered in continuum mechanics and engineering. They describe flat structural elements whose thickness is negligible in comparison to their surface area, such as thin metal plates used in airplanes. Due to their thinness and resulting bendability, treating them as normal elastic 3D structures is often not accurate enough. Instead, different models have been developed that capture their behavior more accurately, the most famous one being the Kirchhoff-Love theory of plates [4]. The theory is built around the concept of the so called middle plane of a plate, a plane with infinitesimal small thickness that lays exactly in the middle of the plate. One assumes that all deformations of the plate are exactly reflected on the middle plane, meaning that straight lines normal to the mid-surface remain straight and normal to the mid-surface after deformation. Furthermore, one assumes that the plate does not change thickness, meaning that the normal stress and strain in direction of the plate thickness are negligible [4]. Since the exact semantics of this theory are not important for this thesis, the interested reader is referred to the cited source for a more detailed description.

1.3.3 Thin Shells

In contrast to thin plates, thin shells are thin, elastic structures that are already bent in their initial configuration. Typical examples include fingernails, car bodies, pans or clothing. It is important to note that thin shell models are consistent with thin plate models whenever their initial configuration is flat.
In simulations, thin shell models are usually solved by computing the energies prevailing in a system and using those to find the vertex positions minimizing the energy under some boundary conditions [5]. The energy minimization is often done using implicit time stepping methods, such as Newmark-timestepping [6], or Newton based energy minimization [7].
Early shell models were based on the Kirchoff-Love constitutive equations mentioned above. They were extremely complex and computationally expensive [8] [9]. In these models, energies and forces over a structure were computed using its smooth surface representation. In 2003, Grinspun et al. showed that these models can be simplified drastically without any loss of information[10]. Their simplification was based on the idea to compute energies using geometric operators and a piecewise linear representation of the surface. Their idea was widely accepted and many newer papers make use of their simplification. A few years later, Grinspun and his colleague Tamstorf published a paper illustrating how the gradient and the hessian of flexural energies over a triangulated surface can be computed easily and efficiently [11].
More recent work usually focuses on more complex scenarios, such as dynamically changing environmental conditions or antiisotropic materials. For example, Chen et al. modeled environmental stimuli through dynamic changes in the rest metrics of a material [12]. However, such complex models are out of the scope of this thesis.

Chapter 2

Math Background

2.1 Barycentric Coordinates

The most common coordinate systems in mathematics, such as the cartesian coordinate system, describe the position of a vertex or an object with respect to the coordinate systems point of origin. One exception are the barycentric coordinate functions. These coordinate functions describe the position of a point respective to three other existing points, e.g. the position of three triangle vertex nodes [13]. They are often used in computer graphics, computer simulations and geometry.
Assuming that a triangle K is defined through the three vertex positions $\boldsymbol{v_0}, \boldsymbol{v_1}$ and $\boldsymbol{v_3}$, the triangle normal is given by

$$\boldsymbol{n} = \frac{(\boldsymbol{v_2} - \boldsymbol{v_1}) \times (\boldsymbol{v_1} - \boldsymbol{v_0})}{||(\boldsymbol{v_2} - \boldsymbol{v_1}) \times (\boldsymbol{v_1} - \boldsymbol{v_0})||} \tag{2.1}$$

and the three barycentric coordinate functions are defined as

$$\lambda_1(\boldsymbol{x}) = \boldsymbol{n} \cdot \frac{(\boldsymbol{v_2} - \boldsymbol{v_1}) \times (\boldsymbol{x} - \boldsymbol{v_1})}{2|K|} \tag{2.2}$$

$$\lambda_2(\boldsymbol{x}) = \boldsymbol{n} \cdot \frac{(\boldsymbol{v_0} - \boldsymbol{v_2}) \times (\boldsymbol{x} - \boldsymbol{v_2})}{2|K|} \tag{2.3}$$

$$\lambda_3(\boldsymbol{x}) = \boldsymbol{n} \cdot \frac{(\boldsymbol{v_1} - \boldsymbol{v_0}) \times (\boldsymbol{x} - \boldsymbol{v_0})}{2|K|} \tag{2.4}$$

$$\tag{2.5}$$

where $\boldsymbol{x} \in \mathbb{R}^3$ is a point in the cartesian coordinate system.

In this thesis, the barycentric coordinate functions and their gradients are used in the context of the FEM. The derivation of the gradient is now shown on the example of λ_1. Using the definition of the gradient and Equation 2.2, we find that

$$\nabla_\Gamma \lambda_1(\boldsymbol{p}) = \frac{1}{2|K|}\nabla_\Gamma\left[\boldsymbol{n}\cdot((\boldsymbol{v_2}-\boldsymbol{v_1})\times(\boldsymbol{x}-\boldsymbol{v_1}))\right] \tag{2.6}$$

$$= \frac{1}{2|K|}\nabla_\Gamma\left[\boldsymbol{n}\cdot\begin{pmatrix}(v_{2y}-v_{1y})(p_z-v_{1z})-(v_{2z}-v_{1z})(p_y-v_{1y})\\(v_{2z}-v_{1z})(p_x-v_{1x})-(v_{2x}-v_{1x})(p_z-v_{1z})\\(v_{2x}-v_{1x})(p_y-v_{1y})-(v_{2y}-v_{1y})(p_x-v_{1x})\end{pmatrix}\right] \tag{2.7}$$

$$= \frac{1}{2|K|}\begin{pmatrix}n_y(v_{2z}-v_{1z})-n_z(v_{2y}-v_{1y})\\n_z(v_{2x}-v_{1x})-n_x(v_{2z}-v_{1z})\\n_x(v_{2y}-v_{1y})-n_y(v_{2x}-v_{1x})\end{pmatrix} \tag{2.8}$$

$$\tag{2.9}$$

which corresponds to

$$\nabla_\Gamma \lambda_1(\boldsymbol{p}) = \frac{\boldsymbol{n}\times(\boldsymbol{v_2}-\boldsymbol{v_1})}{2|K|} \tag{2.10}$$

Similarly, $\nabla_\Gamma \lambda_2$ and $\nabla_\Gamma \lambda_3$ evaluate to

$$\nabla_\Gamma \lambda_2(\boldsymbol{x}) = \frac{\boldsymbol{n}\times(\boldsymbol{v_0}-\boldsymbol{v_2})}{2|K|} \tag{2.11}$$

$$\nabla_\Gamma \lambda_2(\boldsymbol{x}) = \frac{\boldsymbol{n}\times(\boldsymbol{v_1}-\boldsymbol{v_0})}{2|K|} \tag{2.12}$$

$$\tag{2.13}$$

In literature one can often only find the gradient of the barycentric coordinate functions for the 2D case. For evaluation, we set

$$\boldsymbol{n} = (0,0,1)^T \tag{2.14}$$

Plugging this into equation 2.10 we get that in 2D,

$$\nabla_\Gamma \lambda_1(\boldsymbol{p}) = \frac{1}{2|K|}\begin{pmatrix}-v_{2y}+v_{1y}\\v_{2x}-v_{1x}\\0\end{pmatrix} \tag{2.15}$$

$$\tag{2.16}$$

which is equal to the result described in the literature [14].

Chapter 3

Heat Flow Computations

In this chapter we first explain the steps necessary to numerically simulate the heat flow on a thin 3D structure. Next, we detail how functionals can be used to extract the mean temperature and the bulk heat conductivity from the computed heat flow. In a next step, we show the correctness of our implementation by comparing the achieved results to the known analytical solution on an annulus and by illustrating that the solution converges towards a solution when the structure is refined. In a last step we simulate the heat flow through a set of different ADMS, before evaluating the respective functionals and comparing the obtained results.

3.1 Simulation of Heat Flow

3.1.1 Assumptions

When simulating the heat flow through a thin structure, certain simplifications and assumptions have to be made regarding the structure and its representation. In the following, we assume that

1. The structure is thin enough to justify a representation as a 2D surface, meaning that the thickness at any point is significantly smaller than the total surface area. To account for changes in thickness in the simulation, we will assume that the heat conductivity at a vertex is approximately proportional to the thickness at that point.

2. The structure is surrounded by air. Due to the low heat conductivity coefficient of air, this allows for neglecting possible heat exchange of the structure with its surroundings.

3. The structure is available as a triangulated mesh. For simplicity, we further require that the mesh has no non-manifold edges or vertices.

3.1.2 Problem Formulation

The heat flow in a medium can be computed using the steady-state heat equation:

$$-\mathbf{div}(\alpha(\boldsymbol{x})\nabla_{\Gamma}\, u(\boldsymbol{x})) = f(\boldsymbol{x}) \quad \text{on } \Omega$$

where $\boldsymbol{x} \in \mathbb{R}^3$ is a point on the ADMS Ω, $u(\boldsymbol{x}) : \mathbb{R}^3 \to \mathbb{R}$ is the unknown heat flow, $\alpha(\boldsymbol{x}) : \mathbb{R}^3 \to \mathbb{R}$ the heat conductivity and $f(\boldsymbol{x}) : \mathbb{R}^3 \to \mathbb{R}$ the heat flux density of an internal heat source. As was mentioned earlier, we assume that the heat conductivity approximately corresponds to the wall thickness. Furthermore, we always use

$$f(\boldsymbol{x}) \equiv 0 \quad \forall \boldsymbol{x} \in \mathbb{R}^3 \tag{3.1}$$

which implies that no internal heat source is used.
In the case of $\alpha(\boldsymbol{x}) \equiv 1$, this problem is also known as the Laplace-Beltrami Equation

$$\triangle u = 0 \tag{3.2}$$

with $\triangle := \nabla_{\Gamma} \cdot \nabla_{\Gamma}$ being the Laplace Operator.

To solve the steady-state Equation 3.1.2, we have to specify boundary conditions. In this thesis, we restrict the set of possible boundary conditions to homogeneous Neumann boundary conditions

$$(\alpha(\boldsymbol{x})\nabla_{\Gamma}\, u(\boldsymbol{x})) \cdot \boldsymbol{n} = 0 \quad \text{on } \partial\Omega \tag{3.3}$$

and arbitrary Dirichlet boundary conditions

$$u = g \quad \text{on } \partial\Omega$$

where g is a function defined on $\delta\Omega$. In reality, the boundary is often partitioned into multiple regions with varying boundary conditions. For example, a cylinder might have an isolating mantle area (homogeneous Neumann boundary condition), a top area that is heated with $100°C$ and a bottom area that is kept at $0°C$ (constant

Dirichlet boundary conditions). We therefore assume that the boundary $\delta\Omega$ can be partitioned into a set of mutually disjoint boundaries Γ_i, meaning that

$$\delta\Omega = \Gamma_1 \cup \Gamma_2 \cup ... \cup \Gamma_k$$

with

$$\Gamma_1 \cap \Gamma_2 \cap ... \cap \Gamma_k = \emptyset$$

3.1.3 Discretized Linear Variational Problem

The PDE mentioned above can be written as linear variational problem:

$$u \in V: \quad a(u,v) = l(v) \quad \forall v \in V \tag{3.4}$$

$$a(u,v) := \int_{\Omega} \nabla_{\Gamma} u \cdot \nabla_{\Gamma} v \, d\boldsymbol{x} \tag{3.5}$$

as described in Section **??**. Since computers can not solve a problem in an infinite dimensional function space V, we replace V with a finite-dimensional subspace $V_h \subset V$. In practice, this means that we create a basis B_h such that

$$B_h = \{b_h^1, ..., b_h^N\} \quad N := \dim V_h \quad V_h = \text{Span}\{B_h\}$$

B_h can then be used to discretize u_h:

$$u_h = \mu_1 b_h^1 + ... + \mu_N b_h^N \quad \mu_i \in \mathbb{R}, u_h \in V_h$$

This leads to the discrete formulation of the linear variational problem

$$u_h \in V_h: \quad a(u_h, v_h) = l(v_h) \quad \forall v_h \in V_h$$

It is important to note that the choice of basis B_h has no influence on the solution u_h. In this thesis, we choose the barycentric coordinate functions described in Section 2.1 as local basis functions.

3.1.4 Solving the Discretized Linear Variational Problem

The linear system of equations corresponding to the discretized linear variational problem is

$$A\boldsymbol{\mu} = \boldsymbol{\phi}$$

where A is the so called stiffness matrix (Galerkin matrix) $A = [a(b_h^k, b_h^j)]_{j,k=1}^N \in \mathbb{R}^{N \times N}$, $\boldsymbol{\mu}$ the coefficent vector and $\boldsymbol{\phi} = [l(b_h^j)]_{j=1}^N \in \mathbb{R}^N$ the load vector (right hand side vector). This Section is concerned with the assembly of A and $\boldsymbol{\phi}$ as well as with solving the system of equations under consideration of the boundary conditions $\delta\Omega$.

Galerkin Matrix

First, let us consider the the Galerkin matrix assembly. Clearly, $(A)_{ij} = 0$ if there is no connection between the nodes $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Since most nodes are not connected with each other, the Galerkin matrix fulfills the criteria of a sparse matrix.

For $i \neq j$ and $\boldsymbol{x}_i$ connected to $\boldsymbol{x}_j$, we can write

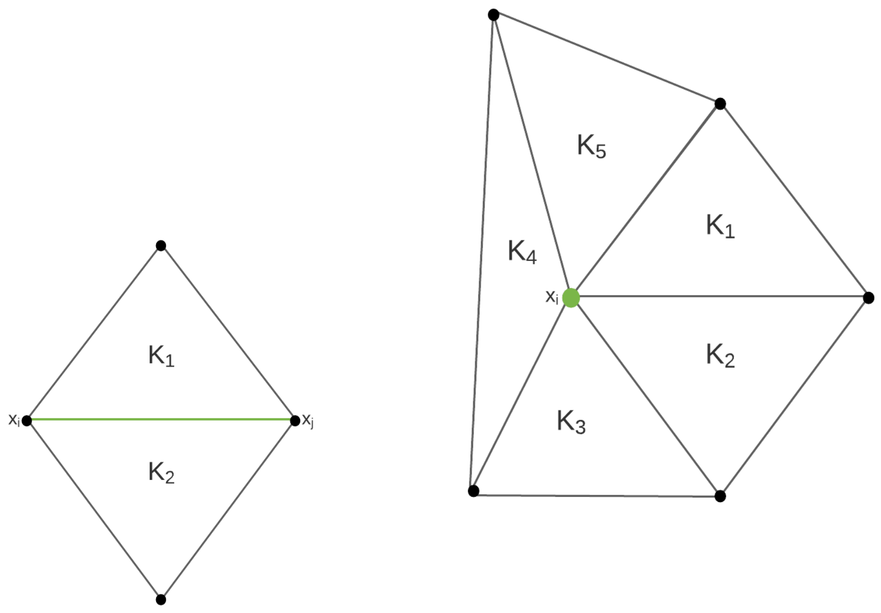

Figure 3.1: Two very simple triangle meshes. Figure 3.1.4 shows the triangles that contribute to $(A)_{ij}$, Figure 3.1.4 shows the triangles that contribute to $(A)_{ii}$.

$$(A)_{ij} = \int_{K_1} \alpha(\boldsymbol{x}) \cdot \nabla_\Gamma\, b^j_{h|K_1} \cdot \nabla_\Gamma\, b^i_{h|K_1}\, d\boldsymbol{x} + \int_{K_2} \alpha(\boldsymbol{x}) \cdot \nabla_\Gamma\, b^j_{h|K_2} \cdot \nabla_\Gamma\, b^i_{h|K_2}\, d\boldsymbol{x} \tag{3.6}$$

where K_1 and K_2 are the triangles adjacent to the edge between the vertices $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ as depicted in Figure 3.1.4

If $i = j$, we sum up the contributions of the triangles associated with node $\boldsymbol{x}_i$

$$(A)_{ii} = \sum_k \int_{K_k} \alpha(\boldsymbol{x}) \cdot \nabla_\Gamma\, b^i_{h|K_k} \cdot \nabla_\Gamma\, b^i_{h|K_k}\, d\boldsymbol{x} \tag{3.7}$$

The most efficient way to construct the Galerkin matrix is by first assembling the local Galerkin matrix for each triangle. The local Galerkin matrix $A_K \in \mathbb{R}^{3\times3}$ can be computed with the formula:

$$A_K(b^j_h, b^i_h) = \int_K \alpha(\boldsymbol{x}) \cdot \nabla_\Gamma\, b^j_{h|K} \cdot \nabla_\Gamma\, b^i_{h|K}\, d\boldsymbol{x} \tag{3.8}$$

where $b^i_{h|K}$ is the i-th basis function of basis B_h for triangle K. Using the derivation for the barycentric coordination functions described in Section 2.1 and using them as local basis functions, we get that

$$\nabla_\Gamma \, b^0_{h|K} = \frac{\boldsymbol{n} \times (\boldsymbol{v_2} - \boldsymbol{v_1})}{2|K|} \tag{3.9}$$

$$\nabla_\Gamma \, b^1_{h|K} = \frac{\boldsymbol{n} \times (\boldsymbol{v_0} - \boldsymbol{v_2})}{2|K|} \tag{3.10}$$

$$\nabla_\Gamma \, b^2_{h|K} = \frac{\boldsymbol{n} \times (\boldsymbol{v_1} - \boldsymbol{v_0})}{2|K|} \tag{3.11}$$

$$\tag{3.12}$$

where n is the normal to the triangle K, $|K|$ is the triangles area and $\boldsymbol{v}_i \in \mathbb{R}^3$, $i, l \in \{0, 1, 2\}$ are the triangle nodes. Using the quadrature rule [15] [14] on Equation 3.8 we find that

$$(A_K)_{i,l} = \frac{1}{3} \cdot |K| \sum_j^3 \alpha(\hat{\boldsymbol{m}}^{\boldsymbol{j}}) \cdot \nabla_\Gamma \, b^i_{h|K} \cdot \nabla_\Gamma \, b^l_{h|K} \tag{3.13}$$

with $\hat{\boldsymbol{m}}^{\boldsymbol{j}}$ being the midpoints of triangle K.

In a next step, the full Galerkin matrix is assembled. In the case of $i \neq j$ we use Equation 3.6 to find that

$$(A)_{ij} = (A_{K_1})_{a,b} + (A_{K_2})_{c,d} \tag{3.14}$$

where $a, b, c, d \in \{0, 1, 2\}$ and $a \neq b$ and $c \neq d$ are the edges vertex indices of triangle K_1 and K_2 respectively. If $i = j$, we get

$$(A)_{ii} = \sum_k (A_K)_{a,a} \tag{3.15}$$

where a is the vertex index of triangle K_k that is adjacent to the vertex node $\boldsymbol{x}_i$. Figure 3.2 visualizes which entries are supposed to be summed up.

Load Vector

Since we always have $\triangle u = 0$, it follows that $\boldsymbol{\phi} \equiv 0$.

Boundary Conditions

As mentioned earlier, we only deal with homogeneous Neumann boundary conditions or Dirichlet boundary conditions. Homogeneous Neumann boundary conditions do not require any changes in either the Galerkin matrix or the load vector. In comparison, Dirichlet boundary conditions have to be enforced, meaning that we have to guarantee that when solving the linear system of equation, the value for $\boldsymbol{x}_i \in \Gamma_l$ has to be equal to $g(\boldsymbol{x}_i)$, the solution on the boundary. This can easily be done by setting $\phi_i = g(\boldsymbol{x}_i)$, $(A)_{ii} = 1$ and every other value in the row $(A)_i$ to zero.

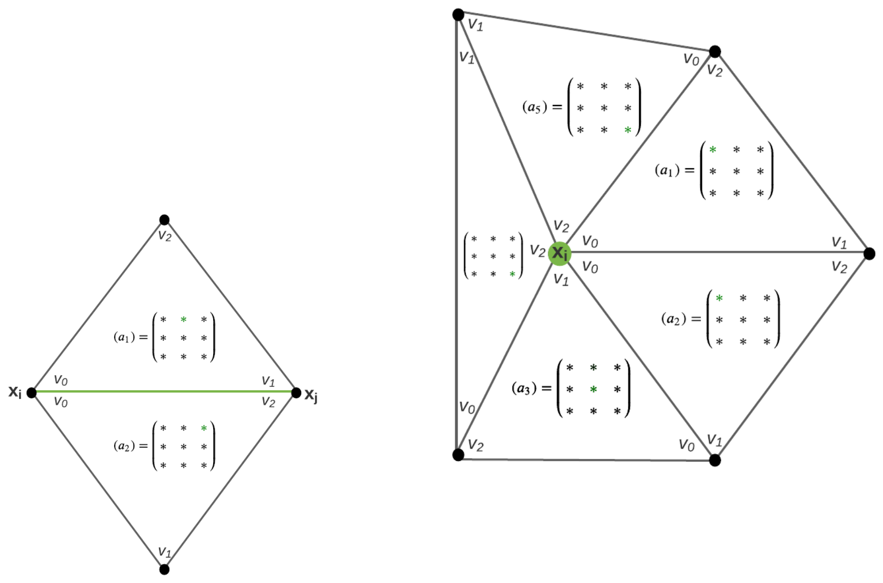

Figure 3.2: Which entries to sum up shown in green

3.2 Functionals

Once the temperature u_h is found or computed, we might want to use it to extract further properties related to it. For example, one might want to know about the mean temperature or the total heat flux in the considered object. More technically, this means that we consider a mapping from our function space $u \in V$ into $\mathbb{R}$. In the following sections we will give the mathematical description and numerical approaches to evaluating the heat flux and the mean temperature functional.

3.2.1 Heat Flux

The heat flux through a boundary $\mathit{\Gamma}_l$ can be computed using the formula:

$$J(u) := \int_{\mathit{\Gamma}_l} \alpha(\boldsymbol{x}) \cdot \nabla_{\mathit{\Gamma}}\, u \cdot \boldsymbol{n}\, dS \tag{3.16}$$

where $\boldsymbol{n}$ is the normal with respect to $\mathit{\Gamma}_l$.
First, let us consider how to numerically solve a line integral. Clearly, we can discretize the integral to the level of triangles:

$$J(u) = \sum_k J_k(u) = \sum_k \int_{\mathit{\Gamma}_l} a(\boldsymbol{x}) \cdot \nabla_{\mathit{\Gamma}}\, u_k \cdot \boldsymbol{n}\, dS \tag{3.17}$$

Since we are working with 2D triangles in a 3D environment, we have to differentiate between four different cases:

1. If no triangle vertex lies on the boundary, then that triangle simply does not contribute to the bulk heat flux. The integral evaluates to zero.

$$J_k(u) = 0 \tag{3.18}$$

2. If only one triangle vertex lies on the boundary, it can be ignored.

3. If two vertices lie on the boundary, we have to evaluate the integral as a line integral. Without loss of generality, assume that the vertices v_i and v_j lay on $\mathit{\Gamma}_l$. Then, we can parametrize the line as

$$f(t) = (1-t) \cdot v_i + t \cdot v_j \qquad t \in (0,1) \tag{3.19}$$
$$f'(t) = -v_i + v_j \tag{3.20}$$

and the integral $J_k(u)$ can be rewritten as

$$\int_0^1 \alpha(f(t)) \cdot \nabla_{\mathit{\Gamma}}\, u_k(f(t)) \cdot \boldsymbol{n} \cdot ||f'(t)||\, dt \tag{3.21}$$

Clearly, $||f'(t)||$ is simply the length of the triangle edge that lie on the boundary. Using simple one point quadrature, we can now rewrite the integral as

$$J_k(u) = \alpha(\boldsymbol{m^{ij}}) \cdot \nabla_{\mathit{\Gamma}}\, u_k(\boldsymbol{m^{ij}}) \cdot \boldsymbol{n} \cdot ||\boldsymbol{v_j} - \boldsymbol{v_i}|| \tag{3.22}$$

where $\boldsymbol{m^{ij}}$ denotes the midpoint between $\boldsymbol{v_i}$ and $\boldsymbol{v_j}$.

4. Since we are working in 3D space with a 2D surface, it can theoretically happen that all three vertices lie on a boundary. This is however mathematically invalid and an indicator that something went wrong.

Next, we have to evaluate $\nabla_\Gamma u$. Recall that by Equation 3.1.3, we can reconstruct the original solution by summing over the value of u at each vertex multiplied by the respective basis function

$$u_h(x) = \sum_i^N \mu_i \cdot b_h^i(\boldsymbol{x}) \tag{3.23}$$

Since $b_i^h(\boldsymbol{x}) \neq 0$ only if $\boldsymbol{x}$ associated with the triangle the basis function belongs to, we can express $\nabla_\Gamma u_K$ of triangle K in terms of the local triangle basis functions:

$$\nabla_\Gamma u_K = \sum_j^3 u_K^j \cdot \nabla_\Gamma b_{h|k}^j(\boldsymbol{x}) \tag{3.24}$$

where j is the local vertex index.

Note that since the gradient of the barycentric functions are constant, $\nabla_\Gamma u_K$ over a triangle is also constant.
While this solution converges, experiments show that [14] it does so only slowly with rate $O(h_M^1)$.

3.2.2 Heat Flux with cutoff function

Alternatively, we can define a cutoff function φ that fulfills the following conditions

$$\varphi(\boldsymbol{x}) \equiv 1 \qquad \boldsymbol{x} \in \Gamma_l \tag{3.25}$$

$$\varphi(\boldsymbol{x}) \equiv 0 \qquad \boldsymbol{x} \in \partial\Omega_{dir} \setminus \Gamma_l \tag{3.26}$$

where Ω_{dir} is the set of all Dirichlet boundary conditions. Since such a cutoff function will not change the value of an integral over the boundary Γ_l, we can use it to extend Equation 3.16 to obtain

$$\int_{\Gamma_l} \alpha(\boldsymbol{x}) \cdot \nabla_\Gamma u \cdot \boldsymbol{n}\, dS = \int_{\Gamma_l} (\alpha(\boldsymbol{x}) \cdot \nabla_\Gamma u \cdot \boldsymbol{n}) \cdot \varphi\, dS \tag{3.27}$$

$$= \int_\Omega \mathbf{div}(\alpha(\boldsymbol{x}) \cdot \nabla_\Gamma u) \cdot \varphi + \alpha(\boldsymbol{x}) \cdot \nabla_\Gamma u \cdot \nabla\varphi\, d\boldsymbol{x} \tag{3.28}$$

According to our problem formulation in Section 3.1.2, the term

$$\mathbf{div}(\alpha(x) \cdot \nabla_\Gamma u) \equiv 0$$

always evaluates to zero. Therefore, the modified Equation 3.27 can be written as

$$J^*(u) := \int_\Omega \alpha(\boldsymbol{x}) \cdot \nabla_\Gamma u \cdot \nabla\varphi\, d\boldsymbol{x} \tag{3.29}$$

Experiments have shown that this modified heat flux converges with $O(h_M^2)$, which leads to a remarkable reduction in output error [14].
Using the quadrature rule and Equation 3.24, we find that $J^*(u)$ can be evaluated as follows:

$$J_K^*(u) := \frac{1}{3} \cdot |K| \cdot \sum_i^3 \alpha(\boldsymbol{x}) \cdot \nabla_\Gamma u(\hat{m}^i) \cdot \nabla\varphi(\hat{m}^i) \tag{3.30}$$

$$J^*(u) = \sum_k^M J_k^*(u) \tag{3.31}$$

3.2.3 Mean Temperature

The mean temperature in an object can be computed using the following functional:

$$F(u) := \frac{1}{\int_{\Omega} \alpha(\boldsymbol{x})\, d\boldsymbol{x} \cdot |\Omega|} \int_{\Omega} \alpha(\boldsymbol{x}) u(\boldsymbol{x})\, d\boldsymbol{x} \tag{3.32}$$

Since $u(\boldsymbol{x})$ is known once the diffusion equation is solved, this integral can easily be computed using the quadrature rule [15].

3.3 Implementation Details

This section details the algorithm and methods used for solving the heat diffusion equation.

3.3.1 Geometry Processing Library IGL

The C++ geometry processing library IGL [16] is a widly used library that offers a variety of mesh processing functions. Amongst other things, it offers standard functions for reading and writing meshes, functions that offer information about the mesh and neighbourhood connectivity, as well as most other functions found in stand alone mesh editors such as mesh lab.
IGL relays heavily on the well known matrix operation library Eigen [17]. Meshes are fully represented as matrices. Assuming one is working in a 3D environment, then all mesh vertices are saved in a matrix $V \in \mathbb{R}^{N\times 3}$ and the triangle connectivity is saved in yet another matrix $F \in \mathbb{N}^{M\times 3}$ where M is the number of triangles.

3.3.2 Solving the Heat Diffusion Equation

For simplicity, the algorithm used for solving the heat diffusion equation is summarized as pseudo code in Algorithm 1. Some of the steps are explained in more detail in the following sections.

Algorithm 1: Overview of the algorithm for solving the heat diffusion equation

```
   Data: Mesh represented as matrices $V$ and $F$
   Result: Solution $u$ of heat diffusion equation
 1 Detect boundaries;
 2 Assemble local Galerkin matrices;
 3 Assemble global Galerkin matrix $A$;
 4 Set $\boldsymbol{\phi}$ to zero vector;
 5 foreach Vertex $i$ on dirchlet boundary do
 6     Set row $i$ in $A$ to zero;
 7     Set $(A)_{ii} = 1$;
 8     Set $\boldsymbol{\phi}_i$ to value on boundary of $i$;
 9 end
10 Solve $A\boldsymbol{u} = \boldsymbol{\phi}$;
11 return $u$
```

Boundary Detection

The boundary detection method implemented is equal to the boundary detection method used in the black box that generates the ADMS, thereby simplifying the interface between black box and the implemented heat flow computation. In this use case, the boundaries are defined as separate meshes. A point is then assumed to lie on the boundary defined by the separate mesh if it lies on one of the meshes faces.
If a point that is technically a boundary point is not detected by the used boundary detection method, it is simply assumed to be on a homogeneous Neumann Boundary. If a point is detected to be on more than one boundary, an error is thrown.

To define the value of a point on its boundary, a boundary function can be defined. The boundary function can either be constant or dependent on the boundary points position.

Galerkin Matrix Assembly

First, the local Galerkin matrices are assembled. To that end, we iterate over all triangle faces and assemble the local matrix (A_K) according to the aforementioned Equation 3.8. For simplicity, the different local Galerkin matrices are saved in a Matrix $L \in \mathbb{R}^{M \times 9}$ where $(A_K)_{i,j}$ corresponds to the entry $L_{K,3i+j}$. Since (A_K) is symmetric, it might be worth reducing the dimensions of L to $L \in \mathbb{R}^{M \times 6}$ or $L \in \mathbb{R}^{M \times 8}$ from a computational point of view. We however decided against this in order to not further complicate the code and since the total computation time is not dominated by the matrix assembly.
Next, the local Galerkin matrices are used to assemble the sparse global Galerkin matrix according to Equation 3.6. Rather than adding the local matrix contributions directly to the global matrix, the construct of Eigen Triplets is used.

Solving the Linear System of Equation

Since the matrix A is necessarily sparse and quadratic, solving the linear system of equation is done using Eigens SparseLU solver.

3.4 Convergence Tests

3.4.1 Annulus

In mathematics, an annulus is defined through two circles sharing a center but with differing radii. For simplicity, we will assume that the annulus lays solely in the (x, y) plane, meaning that $z_i = 0\ \forall(\mathrm{x}_i, y_i, z_i) \in \Omega$.

Let R_{in} be the radius of the small inner circle and R_{out} be the radius of the larger circle. The boundaries can then be defined as

$$\Gamma_{in} := \{(x, y)|\, x^2 + y^2 = R_{in}^2\} \tag{3.33}$$

$$\Gamma_{out} := \{(x, y)|\, x^2 + y^2 = R_{out}^2\} \tag{3.34}$$

let u_{in} and u_{out} denote the temperature in Celsius on Γ_{in} and Γ_{out} respectively. The stationary heat equation problem can then be written as:

$$\triangle u(r, \phi) = 0 \qquad \text{on } \Omega \tag{3.35}$$

$$u(R_{in}, \phi) = u_{in}(R_{in}, \phi) \qquad \text{on } \Gamma_{in} \tag{3.36}$$

$$u(R_{out}, \phi) = u_{out}(R_{in}, \phi) \qquad \text{on } \Gamma_{out} \tag{3.37}$$

$$\tag{3.38}$$

To simplify the search for an appropriate cutoff function, we further set $R_{in} = 0.5 \cdot R_{out}$. The cutoff function

$$\varphi(\boldsymbol{x}) = \frac{2||\boldsymbol{x}||}{R_{out}} - 1 \tag{3.39}$$

$$\nabla\varphi(\boldsymbol{x}) = \frac{2}{R_{out} \cdot ||\boldsymbol{x}||} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.40}$$

then fulfills the criteria for a cutoff function as stated in Equation 3.25.

Real Solution for Boundary Conditions Independent of ϕ

Let us assume that the boundary conditions are constant, meaning that they do not depend on the angle ϕ. Furthermore, let $a(\boldsymbol{x}) \equiv 1$. The solution to Equation 3.35 is then

$$u(r, \phi) = C_1 \cdot \ln(r) + C_2 \tag{3.41}$$

Plugging in the boundary conditions above, we find that

$$C_2 := \frac{u_{out} \cdot \ln(R_{in}) - u_{in} \cdot \ln(R_{out})}{\ln(R_{in}) - \ln(R_{out})} \tag{3.42}$$

$$C_1 := \frac{u_{in}}{\ln(R_{in})} - C_2 \cdot \ln(R_{in}) \tag{3.43}$$

$$\tag{3.44}$$

Assuming $\alpha(x) \equiv 1$, the real heat flux over the boundary Γ_{out} is given by

$$
\begin{aligned}
J_{flux}(u) &= \int_{\Gamma_{out}} \nabla u \cdot n \, dS && (3.45)\\
&= \frac{1}{R_{out}} \int_{\Gamma_{out}} \begin{pmatrix} \frac{C_1}{r} \\ 0 \end{pmatrix} \cdot \begin{pmatrix} r \\ \phi \end{pmatrix} dS && (3.46)\\
&= \frac{C_1}{R_{out}} \int_0^{2\pi} || \begin{pmatrix} -R_{out} \cdot sin(t) \\ R_{out} \cdot cos(t) \end{pmatrix} || \, dt && (3.47)\\
&= C_1 \int_0^{2\pi} 1 \, dt && (3.48)\\
&= 2 \cdot \pi \cdot C_1 && (3.49)
\end{aligned}
$$

and the mean temperature given in Equation 3.32 evaluates to

$$
J_{\bar{T}}(u) = \frac{1}{\pi(R_{out}^2 - R_{in}^2)} \qquad (3.50)
$$

Convergence for Boundary Conditions Independent of ϕ

We can now compare the analytical results to the result computed by the finite element method. We start with the relatively coarse annulus mesh shown in Figure 3.4.1 with $R_{in} = 12mm$ and $R_{out} = 24mm$. We calculate the heat distribution, the two different heat fluxes and the mean temperature and compare it to the real solutions derived in the previous section. This step is then repeated multiple times on increasingly refined annulus meshes. The finest mesh can be seen in Figure 3.4.1.

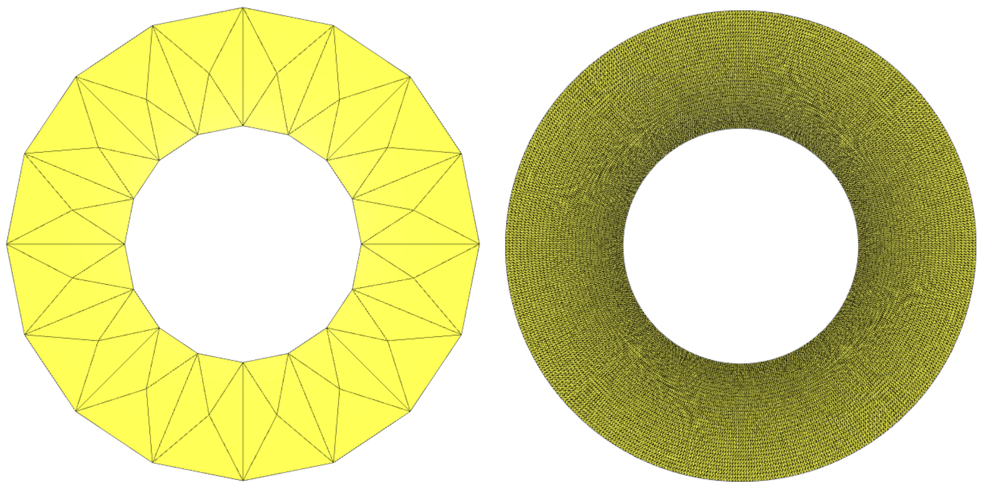

Figure 3.3: Coarse and highly refined annulus annulus mesh with an average edge length of 6.8mm and 0.2mm respectively

Our boundary conditions are given with

$$
\begin{aligned}
u_{in} &= 60 \, C^{\circ} && (3.51)\\
u_{out} &= 10 \, C^{\circ} && (3.52)
\end{aligned}
$$

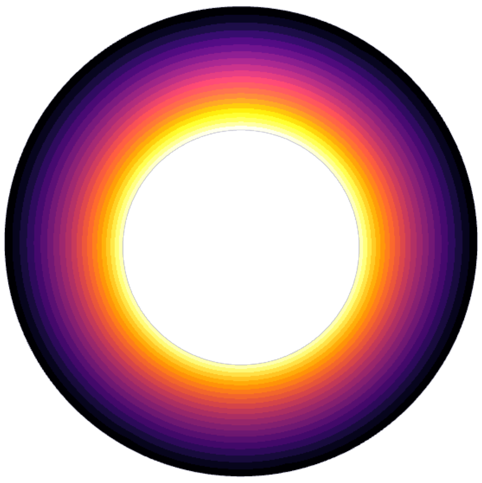

Figure 3.4: Visualization of the heatflux for $u_{in} = 60\,C^{\circ}$ and $u_{out} = 10\,C^{\circ}$

Applying the formulas for the analytical solutions of the annulus, we find

$$u(r,\phi) = -72.1348 \cdot \ln(r) + 238.701 \tag{3.53}$$

$$J_{flux} = -453.236 \tag{3.54}$$

$$J_{\bar{T}} = 29.4007 \tag{3.55}$$

The computed solution is visualized in Figure 3.4.1

In Figure 4.5, we plot the RMSE between the real solution and the computed solution at every point of the mesh for different mesh resolutions. We can clearly see that the result converges towards the real solution.

Similarly, the absolute error between the different functionals and their respective real solution are plotted in 3.4.1. As expected, the computed solution of the heat flux functional is far more exact when using the cutoff function. Interestingly, the computed mean temperature seems to be even more precise. This can be explained by the fact that the mean temperature is already continuous on $H^1(\Omega)$ without introducing an artificial cutoff function.

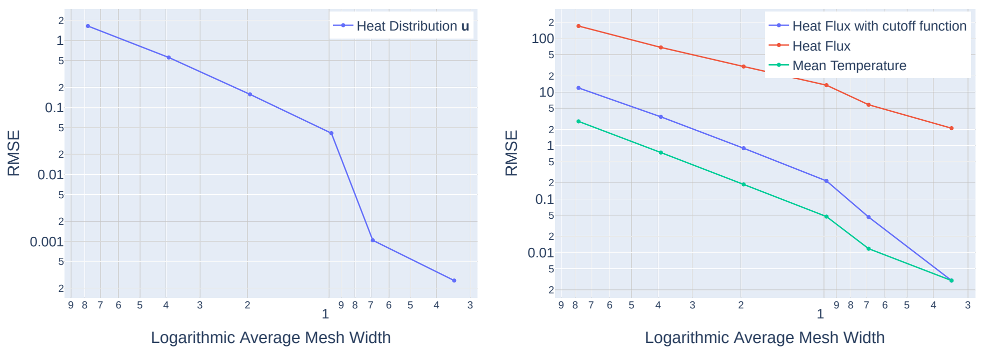

Figure 3.5: Error convergence on annulus

Convergence for Boundary Conditions Dependent of ϕ

In this section, we show convergence on the annulus for a set of more complex boundary conditions. For this test case, we only show that the solution converges towards a fixed solution. Using the meshes, functionals and cut off function as above, but this time modify the problem statement to be

$$\triangle u(r,\phi) = 0 \quad \text{on } \Omega \tag{3.56}$$
$$u(R_{in},\phi) = 0 \quad \text{on } \Gamma_{in} \tag{3.57}$$
$$u(R_{out},\phi) = 4\sin(5\phi) \quad \text{on } \Gamma_{out} \tag{3.58}$$
$$\tag{3.59}$$

where $\phi \in [-\pi,\pi]$ and therefore defined as

$$\phi = \arctan 2(\boldsymbol{x}) \tag{3.60}$$

A visualization of the heat equation on the coarsest and finest grid can be seen in Figure 3.6.

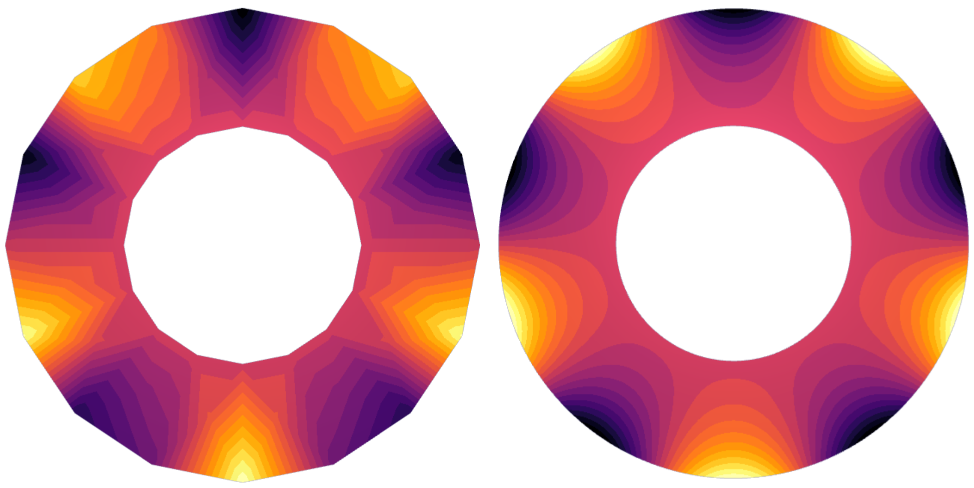

Figure 3.6: Solution of the heat equation for $u(R_{in},\phi) = 0$ on Γ_{in} and $u(R_{out},\phi) = 4\cdot\sin(5\phi)$ on Γ_{out} on a coarse and a refined mesh. It is easy to see that both meshes show the same solution.

The value of the functional at each refinement step as well as the absolute change in value in comparison to the previous step is plotted in Figure 3.7. The heat flux and the mean temperature both converge towards zero. This seems reasonable considering that the temperature on the inner circle is zero and is uniformly distributed between $-4C^\circ$ and $+4C^\circ$. As before, it becomes obvious that computing the heat flux with a cutoff gives far better results than computing the heat flux normally.

3.4.2 Mesh Independence of Result on Cylindrical ADMS

We now show that the computed heat flow on a cylindrical ADMS is independent of the chosen mesh. To that end, we compare the computed heat flux and mean temperature of a cylindrical ADMS on two different mesh representations, shown in Figure 4.2.

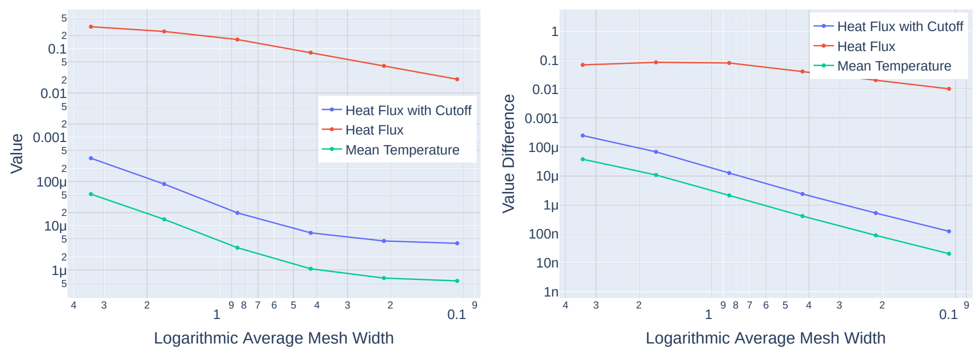

Figure 3.7: Change in value and value difference between respective mesh refinement steps on an annulus.

Let R be the cylinder radius and H its height. A natural boundary region is defined through the cylinder surface $\mathit{\Omega}$, split into the mantle area $_m$, the cylinder top $\mathit{\Gamma}_{top}$ and the cylinder bottom $\mathit{\Gamma}_{bot}$. For this test, we keep the top area at $60C^\circ$, the bottom area at $10C^\circ$ and assume that the mantle area is isolated. The problem statement can then be written as

$$\triangle u(r,\phi,h) = 0 \qquad \text{on } \mathit{\Omega} \tag{3.61}$$
$$u(r,\phi,0) = 10C^\circ \qquad \text{on } \mathit{\Gamma}_{bot} \tag{3.62}$$
$$u(r,\phi,H) = 60C^\circ \qquad \text{on } \mathit{\Gamma}_{top} \tag{3.63}$$
$$\nabla u(R,\phi,h) = 0 \qquad \text{on } \mathit{\Gamma}_m \tag{3.64}$$
$$\tag{3.65}$$

In Figure 3.9 the vertices of the used cylindrical ADMS lie on $\mathit{\Gamma}_{top}$, $\mathit{\Gamma}_{bot}$, $\mathit{\Gamma}_m$ are colored in red, green and blue respectively.
As a cutoff function, we define

$$\varphi(\boldsymbol{x}) = \frac{h^2}{R^2} \tag{3.66}$$
$$\nabla\varphi(\boldsymbol{x}) = \frac{2h}{R^2}\begin{pmatrix}0\\0\\1\end{pmatrix} \tag{3.67}$$

which fulfills the criteria of a cutoff function when the the boundary flux over $\mathit{\Gamma}_{top}$ is computed and if $\mathit{\Gamma}_m$ is isolating.

The results of the heat flux for the given problem on the coarsest mesh can be seen in Figure 3.10. Table 3.1 lists the properties and the computed functionals for the two different meshes. It is easy to see that the heat flux and the mean temperature is almost identical, thereby showing that the results can be assumed to be mesh independent.

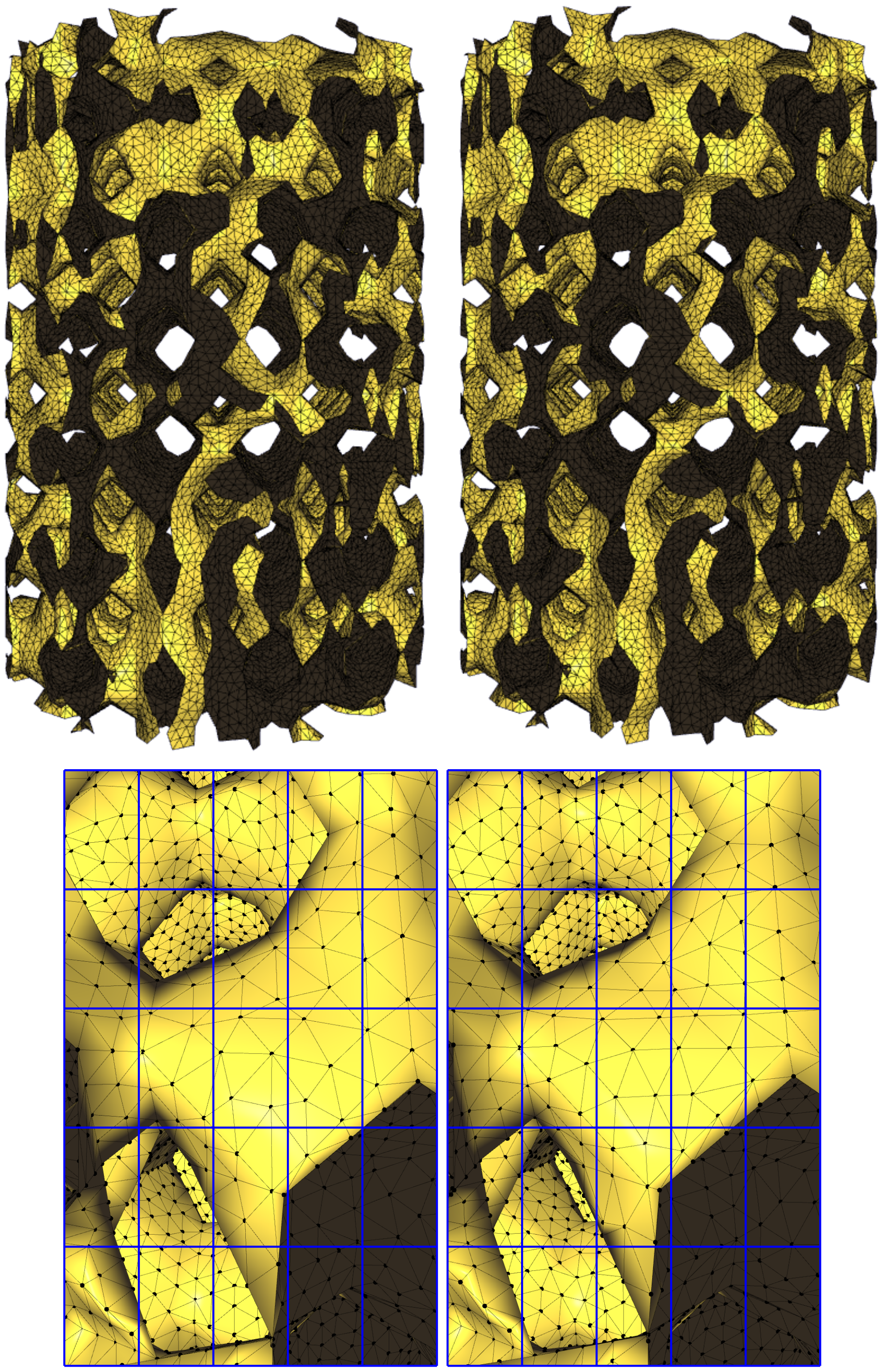

Figure 3.8: The two different cylindrical meshes used to proof that the results are mesh independet. The Figure shows the two different meshes (top) as well as a zoom from the top of the cylinders (left). The blue lines help see the differences between the two meshes.

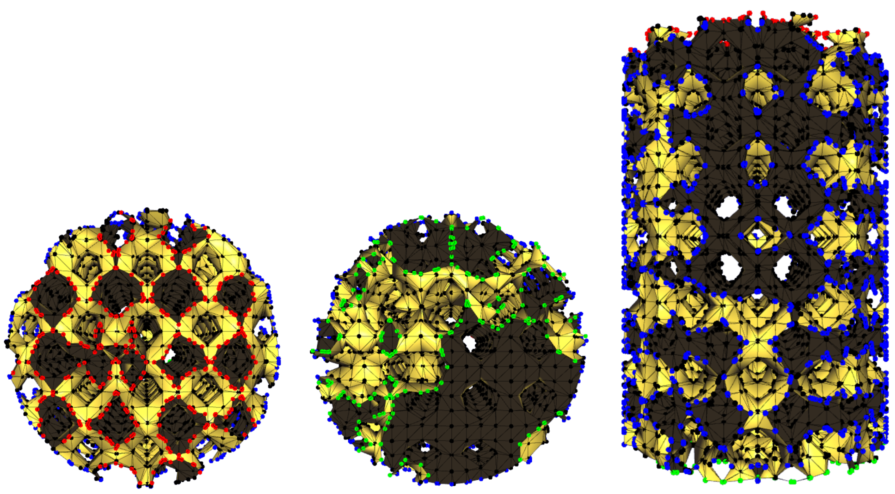

Figure 3.9: Coarse ADMS mesh with vertices colored according to the boundary they belong to. Red for the top area, green for the bottom area, blue for the mantel area and black if they do not belong to any boundary.

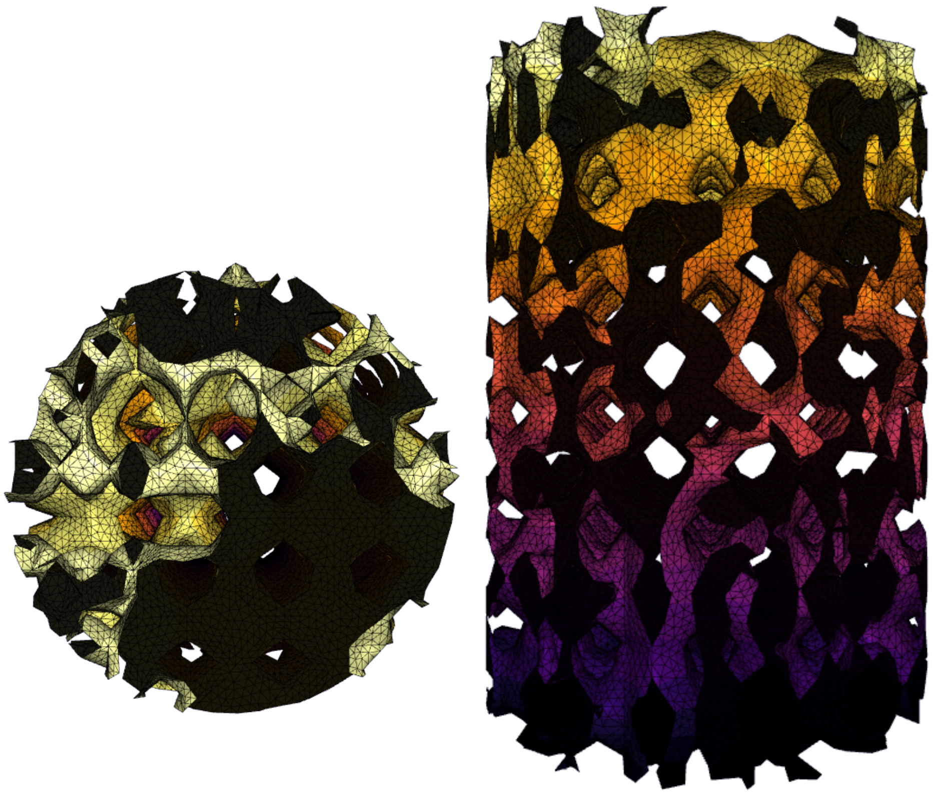

Figure 3.10: Result on one of the two cylindrical meshes.

Cylinder	**Faces**	**Mean Temperature**	**Heat Flux**	**Heat Flux w.** φ
Cylinder 1	81’397	40.1045	250.954	220.405
Cylinder 2	86’452	40.1502	249.952	221.364

Table 3.1: Properties of two different meshes of the same ADMS. It is shown that the results are independent of the chosen mesh.

Cylinder	**Faces**	**Mean Temperature**	**Heat Flux**	**Heat Flux w. ϕ**
Cylinder 1	624’220	31.294	269.209	277.351
Cylinder 2	607’269	29.1906	251.612	259.728
Cylinder 3	591’515	27.1429	235.508	241.24

Table 3.2: Properties of three different cylindrical ADMS simulated with increasing density on the top of the cylinder

3.5 Results on ADMS

In this section we present the results of the heat flux equation and the respective functionals on different cylindrical and cubic ADMS. The results are then analysed and compared with each other.

3.5.1 Cylinder Structure

The exact geometry of an ADMS is determined by many different factors. One of the most important factors is a density field that describes how large the tubes and channels are supposed to be in a certain region. The higher the density, the larger the tubes and channels. In this section we now study the effect of differently sized channels and tubes. To this end, we simulate the heat flow on three different cylindrical ADMS with increasingly larger tubes and channels towards the top of the ADMS. The top of each cylinder is set to be $60C^{\circ}$, the bottom to be $10C^{\circ}$ and the mantel area to be isolating.

The results are shown in Figure 3.11 and their respective functional values and geometric properties are listed in Table 3.2. When looking at the Figures and the reported data, it becomes clear that the heat applied on the top diffuses faster when the top tubes become larger. This is probably due to the increased surface area present in that area. In reverse, the lower temperature at the bottom stays until further into the cylindrical structure. These facts are also reflected in the decreasing mean temperature and the decrease in heat flux over the upper boundary.

3.5.2 Wall Width

Usually the wall width of an ADMS is not constant over the whole surface. To study the effects the wall width has on the heat distribution, we first show the difference an increase in constant wall width has before experimenting with wall widths that vary over the surface.

Increase in Constant Wall Width

To study the effect of a change in constant wall width, we simulate the same three cylinders as above but with a five times thicker surface. Considering the formula used to compute the heat flux, we would expect the mean temperature to stay the same and a increase in heat flux proportional to the increase in constant wall width.
Table 3.3 lists the computed properties. It is easy to see that the results are as expected: While the mean temperature is unchanged, the heat flux increases proportionally.
By simulating the heat flow on the same cylinders as in the previous section, we can furthermore compute the absolute change in temperature at each vertex. As would

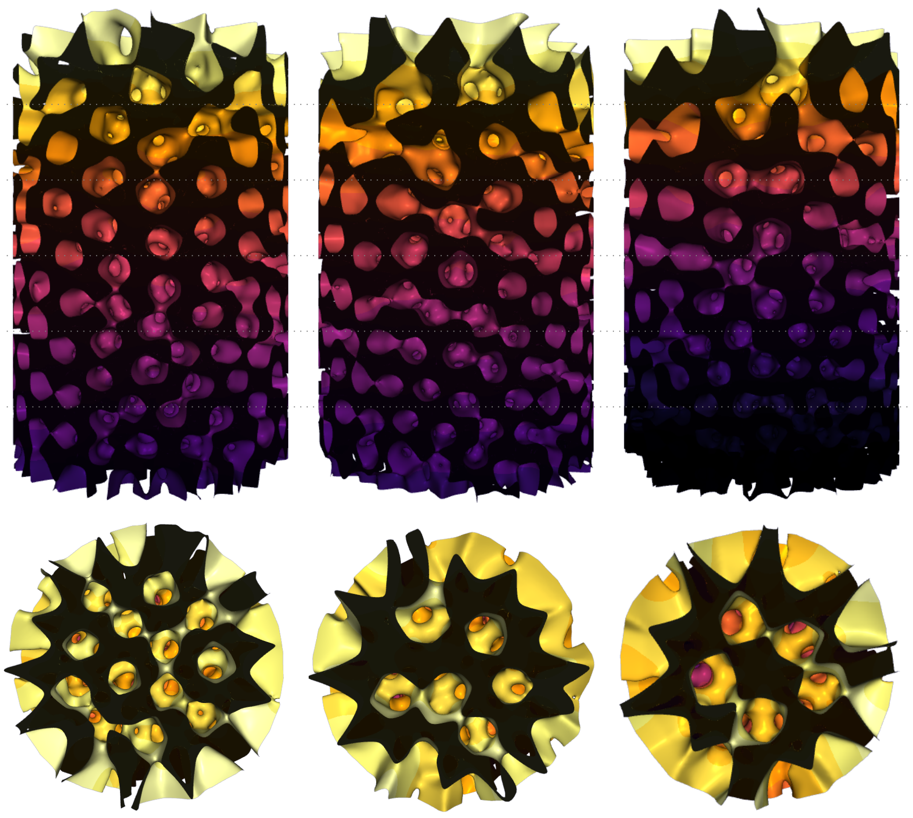

Figure 3.11: Heat Flow in three different cylindrical ADMS simulated with increasingly larger density on the top of the cylinder. The cylinders are numbered as cylinder 1, cylinder 2 and cylinder 3 from left to right.

be expected, the temperature difference at every vertex is so small that it can be attributed to floating point rounding errors.

Linearly Increasing Wall Width

Using the same cylinderical ADMS as before, we now assume the wall width to be a function of the cylinders height.

$$w(\boldsymbol{x}) = 1 + \frac{h}{H} \tag{3.68}$$

where h is the height of point $\boldsymbol{x}$ and H is the total height of the cylinder. Using this function w, we get a large wall width for the large tubes on top of the cylinder and a small wall width for the tiny tubes on the bottom of the cylinder. The results can be seen in Figure 3.12, the results of the functionals are listed in Table 3.4. Clearly and as expected, the mean temperature and the heat flux over the upper boundary increase.

Cylinder	Faces	Mean Temperature	Heat Flux	Heat Flux w. ϕ
Cylinder 1	624'220	31.294	1346.05	1386.75
Cylinder 2	607'269	29.1906	1258.06	1298.64
Cylinder 3	591'515	27.1429	1177.54	1206.2

Table 3.3: Properties of three different cylindrical ADMS simulated with increasingly larger density on the top of the cylinder. While the cylinders are the same as the ones used in Table 3.2, the wall width was increased by a factor of five.

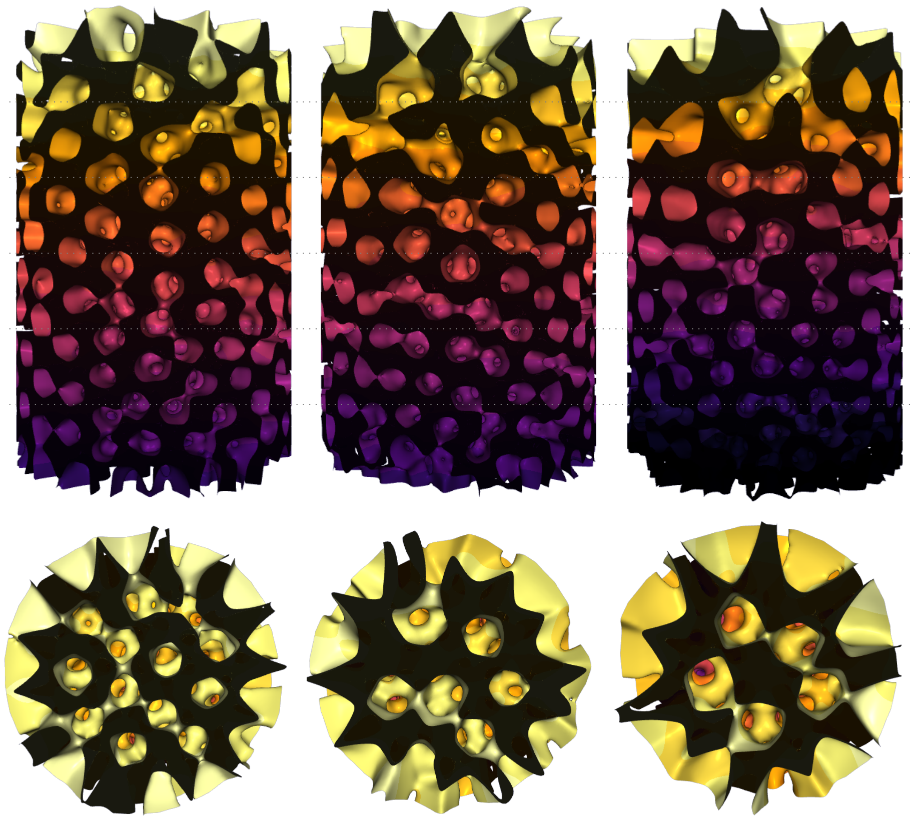

Figure 3.12: Heat Flow in three different cylindrical ADMS simulated with increasingly larger density on the top of the cylinder. The cylinders are numbered as cylinder 1, cylinder 2 and cylinder 3 from left to right. In comparison to the cylinders in Figure 3.11, the cylinders in this Figure have a linearly increasing wall width towards the top of the cylinders.

Cylinder	Faces	Mean Temperature	Heat Flux	Heat Flux w. φ
Cylinder 1	624'220	36.8127	400.684	411.724
Cylinder 2	607'269	34.6243	380.188	391.04
Cylinder 3	591'515	32.3979	360.022	369.263

Table 3.4: Properties of three different cylindrical ADMS simulated with increasingly larger density on the top of the cylinder. While the cylinders are the same as the ones used in Table 3.2, the wall width was increased linearly towards the top of the cylinders.

Chapter 4

Elasticity Computations

This chapter describes the mathematics and physics needed to compute the stiffness of thin shelled structures. First, we identify a set of assumptions needed for a reasonably simple simulation. Next, we explain the different simulation steps needed to find the displacements of internal vertices as a reaction to the displacement of some boundary vertices. These displacements and resulting forces are then used to compute the stiffness.

4.1 Assumptions

Just as when simulating the heat flow through a thin structure, certain simplifications and assumptions have to be made when computing the vertex displacement in a thin structure. In the following, we assume that

1. The structure is thin enough to justify a representation as a 2D surface, meaning that the thickness at any point is significantly smaller than the total surface area. The theoretical material thickness is used to determine a stiffness parameter and therefore still has an influence on the simulation.
2. When no external forces or displacements are enforced, the energy at any point in the structure is zero.
3. The structure is elastic, meaning that once external forces or displacements are not enforced anymore, the material deforms back to its initial configuration.
4. The material is isotropic, meaning that the material properties are independent of their direction. Isotropic materials are for example glass and metals, non-isotropic materials include layered stone and wood.
5. There are no rigid body transformations but only deformations.
6. The deformation is small enough to not make in-object collision detection necessary.
7. The structure is available as a triangulated mesh. For simplicity, we further require that the mesh has no non-manifold edges or vertices.

4.2 Deformation

Whenever force is applied to a body or structure, it transforms in some way. It might rotate, translate or deform. While rotations and translations are rigid body transformations that do not change the shape and thereby the potential energy in the body, deformations trigger an increase in potential energy. According to the principle of minimal work, the body will deform in a way that minimizes this increase in potential energy. When simulating deformations, we make use of this behavior: We first define a discretized version of the elastic energy known from mechanics, compute its derivative and Hessians before using Newtons Method to minimize the energy present in a body.
In the following, let $\boldsymbol{e_{ij}}$ denote the edge between two vertices $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, K denote a triangle and $|K|$ its area. The hinge angle between two triangles K_l and K_r is denoted as θ_{lr}. A visualization of this can be seen in Figure 4.1. Bars relate to the quantity in its undeformed configuration.

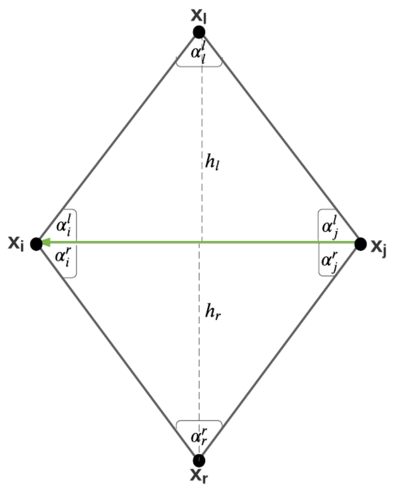

Figure 4.1: Visualization with of naming convention in triangle

4.2.1 Energy

The energy used in this simulation needs to be a measure for the amount of deformation a structure has experienced. When working with solids, it is usually enough to consider the well known elastic energy that is used when working with springs. For thin, easily bendable structures such as the ADMS, one also needs to take bending energy into account.
A typically used energy is the shell energy [18] defined as

$$E_{shell} = \int_{\Omega} k_{stiff}||I(u,v) - \bar{I}||_F^2 + k_{bend}||II - \bar{II}||_F \, dudv \tag{4.1}$$

where I and II are the fundamental forms of a surface and measure the amount of stretching and bending in a body respectively [19].
While this model is physically accurate, it is computationally expensive to compute. Instead, we use the discretized version of this energy presented by Grinspun et al. [10]. The energy becomes

$$E = \sum_{\theta_{lr}} E_{bend}(\theta_{lr}) + \sum_{\boldsymbol{e_{ij}}} E_{membrane}(\boldsymbol{e_{ij}}) \tag{4.2}$$

where E_{bend} accounts for the bending energy over all hinges and $E_{membrane}$ accounts for the material stretching. Both these energies will be explained in more detail in the two following sections.

4.2.2 Bending Energy

The bending energy describes the amount of change in local curvature over a hinge. It has been defined in several different ways in the literature [10] [20] [21]. In this thesis, we will use the bending energy as defined in [11]:

$$E_{bend}(\theta_{lr}) = k_{bend} \cdot a_{lr} \cdot (\varphi(\theta_{lr}) - \varphi(\bar{\theta}_{lr}))^2 \tag{4.3}$$

$$a_{lr} = 3 \cdot \frac{|\bar{\boldsymbol{e}}_{\boldsymbol{ij}}|^2}{|\bar{K}_l| + |\bar{K}_r|} \tag{4.4}$$

$$\varphi = 2 \cdot \tan\left(\frac{\theta_{lr}}{2}\right) \tag{4.5}$$

where k_{bend} is the bending stiffness. The chosen φ is especially useful since it prevents faces from collapsing by increasing the energy drastically when θ_{lr} becomes too large.
To ensure that the model agrees with the plate model when in resting state, the bending stiffness should be set to

$$k_{bend} = \frac{D}{2} = \frac{Yh^3}{24(1-\nu^2)} \tag{4.6}$$

where Y is the Young's modulus $[\frac{N}{m^2}]$, ν is the Poisson's ratio and h is the shell thickness $[m]$ [11].

4.2.3 Membrane Energy

Membrane Energy is the energy that results from the stretching and shearing of a surface.

$$E_{membrane} = E_{shear} + E_{stretch} \tag{4.7}$$

where E_{shear} is the energy related to local changes in edge length and $E_{stretch}$ the energy related to changes in area.
We define the shearing energy as

$$E_{shear}(\boldsymbol{e_{ij}}) = k_{shear} \cdot (|\boldsymbol{e_{ij}}| - |\bar{\boldsymbol{e}}_{\boldsymbol{ij}}|)^2 \tag{4.8}$$

and the stretching energy as

$$E_{stretch}(\boldsymbol{e_{ij}}) = k_{stretch} \cdot (|K| - |\bar{K}|)^2 \tag{4.9}$$

While materials like rubber are prone to stretching and shearing, other materials such as stone barely stretch or shear at all. In simulations this can easily be achieved by setting large enough stretching $k_{stretch}$ and shearing coefficients k_{shear}.
Since ADMS are mostly printed with stiff materials, this restriction applies to us. For simplicity and since the stretching can not increase without an increase in shearing energy, we will use $E_{stretch} = 0$.

4.2.4 Derivatives and Hessian

Computing the energy minima using Newton's method, the energy gradient and the energy Hessian are required. This section lists the gradient and Hessian for the bending- and membrane energy.
It is interesting to note that many papers get around the error prone derivation of energy Hessians by either applying gradient descent to find the energy minima or by computing the Hessian using auto differentiation. However, both these methods can cause a drastic increase in runtime.

Bending Energy

First, let us consider the gradient of the angle θ_{lr}. Clearly, θ_{lr} depends on the four vertices related with the two triangles left and right of the hinge. The respective gradients are:

$$\frac{\partial\theta}{\partial\boldsymbol{x}_i} = \frac{\cos(\alpha_j^l)}{h_i^l}\boldsymbol{n_l}^T + \frac{\cos(\alpha_j^r)}{h_i^r}\boldsymbol{n_r}^T \tag{4.10}$$

$$\frac{\partial\theta}{\partial\boldsymbol{x}_j} = \frac{\cos(\alpha_i^l)}{h_j^l}\boldsymbol{n_l}^T + \frac{\cos(\alpha_i^r)}{h_j^r}\boldsymbol{n_r}^T \tag{4.11}$$

$$\frac{\partial\theta}{\partial\boldsymbol{x}_l} = -\frac{1}{h_l}\boldsymbol{n_l}^T \tag{4.12}$$

$$\frac{\partial\theta}{\partial\boldsymbol{x}_r} = -\frac{1}{h_r}\boldsymbol{n_r}^T \tag{4.13}$$

$$\tag{4.14}$$

Furthermore, we have to take the derivative of φ. Remember that $\varphi(\theta) = 2\tan(\frac{\theta}{2})$. Assuming for a second that θ is constant we get

$$\varphi'(\theta) = \sec^2\left(\frac{\theta}{2}\right) \tag{4.15}$$

$$= 1 + \tan^2\left(\frac{\theta}{2}\right) \tag{4.16}$$

$$\tag{4.17}$$

and

$$\varphi''(\theta) = \tan\left(\frac{\theta}{2}\right)\cdot\sec^2\left(\frac{\theta}{2}\right) \tag{4.18}$$

$$= \varphi'(\theta)\cdot\tan\left(\frac{\theta}{2}\right) \tag{4.19}$$

$$\tag{4.20}$$

However, θ is not constant. We therefore have to apply the chain rule, leaving us with

$$\frac{\partial\varphi(\theta)}{\partial\boldsymbol{x}} = (1 + \tan^2\left(\frac{\theta}{2}\right))\cdot\frac{\partial\theta}{\partial\boldsymbol{x}} \tag{4.21}$$

and the total bending energy derivative

$$\frac{\partial E_{bend}(\theta_{lr}}{\partial\boldsymbol{x}} = 2\cdot k_{bend}\cdot a_{lr}\cdot(\varphi(\theta_{lr}) - \varphi(\bar{\theta}_{lr}))\cdot\frac{\partial\varphi(\theta)}{\partial\boldsymbol{x}} \tag{4.22}$$

respectively. The bending energy Hessian can directly be taken from the paper and respective technical report by Tamstorf and Grinspun [11].
Since writing down the formula for the Hessian is extensive and error prone, the reader is referred to the paper by [11] and the accompanying technical report for the exact formulas needed.

Membrane Energy

Clearly, the membrane energy only depends on the two vertices describing the respective edge $\boldsymbol{e_{ij}} = \boldsymbol{x}_i - \boldsymbol{x}_j$. Therefore, all derivatives other than $\frac{\partial E_{shear}(\boldsymbol{e_{ij}})}{\partial \boldsymbol{x}_i}$ and $\frac{\partial E_{shear}(\boldsymbol{e_{ij}})}{\partial \boldsymbol{x}_j}$ will be zero. Using the chain rule and that $\frac{\partial |\boldsymbol{e_{ij}}|}{\partial \boldsymbol{x}_i} = \frac{\boldsymbol{e_{ij}}}{|\boldsymbol{e_{ij}}|}$, we find that the gradients are given by:

$$\frac{\partial E_{shear}(\boldsymbol{e_{ij}})}{\partial \boldsymbol{x}_i} = 2 \cdot k_{shear} \cdot \frac{\boldsymbol{e_{ij}}}{|\boldsymbol{e_{ij}}|}(|\boldsymbol{e_{ij}}| - |\bar{\boldsymbol{e}}_{\boldsymbol{ij}}|) \tag{4.23}$$

$$\frac{\partial E_{shear}(\boldsymbol{e_{ij}})}{\partial \boldsymbol{x}_j} = -2 \cdot k_{shear} \cdot \frac{\boldsymbol{e_{ij}}}{|\boldsymbol{e_{ij}}|}(|\boldsymbol{e_{ij}}| - |\bar{\boldsymbol{e}}_{\boldsymbol{ij}}|) \tag{4.24}$$

$$\tag{4.25}$$

Computing the energy Hessian results in:

$$\frac{\partial^2 E_{shear}}{\partial \boldsymbol{x}_i^2} = 2 \cdot k_{shear} \cdot \left(I \cdot \left(1 - \frac{|\bar{\boldsymbol{e}}_{\boldsymbol{ij}}|}{|\boldsymbol{e_{ij}}|} \right) + \boldsymbol{e_{ij}} \boldsymbol{e_{ij}}^T \left(- \frac{|\boldsymbol{e_{ij}}| - |\bar{\boldsymbol{e}}_{\boldsymbol{ij}}|}{|\boldsymbol{e_{ij}}|^3} + \frac{1}{|\boldsymbol{e_{ij}}|^2} \right) \right) \tag{4.26}$$

$$\frac{\partial^2 E_{shear}}{\partial \boldsymbol{x}_j^2} = 2 \cdot k_{shear} \cdot \left(I \cdot \left(1 - \frac{|\bar{\boldsymbol{e}}_{\boldsymbol{ij}}|}{|\boldsymbol{e_{ij}}|} \right) + \boldsymbol{e_{ij}} \boldsymbol{e_{ij}}^T \left(- \frac{|\boldsymbol{e_{ij}}| - |\bar{\boldsymbol{e}}_{\boldsymbol{ij}}|}{|\boldsymbol{e_{ij}}|^3} + \frac{1}{|\boldsymbol{e_{ij}}|^2} \right) \right) \tag{4.27}$$

$$\frac{\partial^2 E_{shear}}{\partial \boldsymbol{x}_i \boldsymbol{x}_j} = 2 \cdot k_{shear} \cdot \left(-I \cdot \left(1 - \frac{|\bar{\boldsymbol{e}}_{\boldsymbol{ij}}|}{|\boldsymbol{e_{ij}}|} \right) + \boldsymbol{e_{ij}} \boldsymbol{e_{ij}}^T \left(\frac{|\boldsymbol{e_{ij}}| - |\bar{\boldsymbol{e}}_{\boldsymbol{ij}}|}{|\boldsymbol{e_{ij}}|^3} - \frac{1}{|\boldsymbol{e_{ij}}|^2} \right) \right) \tag{4.28}$$

and clearly $\frac{\partial^2 E_{shear}}{\partial \boldsymbol{x}_i \boldsymbol{x}_j} = (\frac{\partial^2 E_{shear}}{\partial \boldsymbol{x}_j \boldsymbol{x}_i})$

4.2.5 Energy Minima

The energy gradient and hessian computed in the previous section can now be used to find the energy minima with the damped Newton's method. The general idea is to find the energy minima by computing

$$x_{k+1} = x_k - \gamma H(E)^{-1} \cdot \nabla E \tag{4.29}$$

where $\in (0, 1]$ is a small step size. However, computing the inverse of a matrix is expensive and furthermore unlikely to preserve the sparsity of H. Using the energy gradients derived in the previous sections, we can instead compute

$$Hu = \nabla E \tag{4.30}$$

and then

$$x_{k+1} = x_k - \gamma u \tag{4.31}$$

In the case of the complex ADMS, it is vital that a small γ is chosen.

4.3 Stiffness

Since ADMS are not guaranteed to deform regularly, computing their stiffness is tricky. For simplicity, we only compute the stiffness in one degree of freedom, namely for deformations happening in the direction of the z-axis. The average stiffness is then defined as

$$\kappa_{avg} = \frac{1}{N} \sum_i \frac{F_{z,i}}{\delta} \tag{4.32}$$

where δ is the deformation in the z axis and $F_{z,i}$ is the force applied to the vertices on the cylinder top.

4.4 Implementation Details

This section details the algorithm and methods used for finding the displacement resulting from a small external force. This information is then used to compute the stiffness.
Just as for the heat flux computations, the geometry library IGL [16] as described in Section 3.3.1 is used. A very broad overview of the algorithm used is shown in Algorithm 2. Most of the different steps needed are explained in more detail in the following sections. The detection of boundaries works as described in Section 11.

Algorithm 2: Overview of the algorithm for computing the displacement and stiffness in a thin shell structure

```
   Data: Mesh represented as matrices $V$ and $F$
   Result: Displacement and stiffness
 1 Mesh cleaning;
 2 Compute initial hinge angles and edge lengths;
 3 Detect boundaries;
 4 Enforce small displacement;
 5 while Energy has not converged do
 6     Compute energy, energy gradient and energy Hessian in mesh;
 7     Enforce boundaries;
 8     Do one Newton Step;
 9 end
10 Compute stiffness;
```

4.4.1 Mesh Cleaning

During our experiments, we found that the runtime of our algorithm increases drastically whenever obtuse triangles are present. Unfortunately, these triangles are known to occur quite often in ADMS meshes. We therefore used a special mesh cleaning method that was implemented outside the scope of this thesis to successfully remove these obtuse triangles.

4.4.2 Computing Energy Gradients and Hessians

The energy Hessians and gradients are computed by iterating over all possible edges and computing the energy gradient and Hessians using the respective formulas. To ensure correct results, it is important to guarantee that face normals are oriented consistently and do not flip between different Newton-Iterations, that edge normals always point outward and that the hinge angle θ_{lr} is treated as a signed quantity, allowing a differentiation between concave and convex hinges. Ways to ensure this are detailed in the following sections.
To avoid unnecessary computational load, all properties needed are computed only once after each Newton step.

Face Normal

For simplicity, we ensure a consistent face normal orientation by using the respective algorithm from the IGL Library.

Edge Normals

The simplest way to ensure that the edge normals always point outward is by guaranteeing that the edge vectors always point counter-clockwise. The edge normals can then be computed using

$$\boldsymbol{m_l} = \boldsymbol{e_{ij}} \times \boldsymbol{n} \tag{4.33}$$

and similarly for $\boldsymbol{m_i}$ and $\boldsymbol{m_j}$ Considering the rather complex structure of ADMS and to avoid having to relabel the vertices and triangle faces of an already existing mesh, we decided on a different approach: First, we compute the different edge normals with the previous Equation as if we could guarantee a consistent edge direction. Next, for each vertex, we compute the vector pointing from the vertex to the triangle centroid:

$$\boldsymbol{s} = \frac{1}{3}(\boldsymbol{x}_i + \boldsymbol{x}_j + \boldsymbol{x}_l) - \boldsymbol{x}_l \tag{4.34}$$

Last, we check the dot product between the vertex-centroid vector and the previously computed edge normal. If the dot product is smaller than zero, the previously computed edge normal $\boldsymbol{m}$ is pointing inward and has to be flipped.

Hinge Angle

As was shown in Section 4.2.2, the bending energy depends on the term $\varphi(\theta) = 2\tan\left(\frac{\theta}{2}\right)$. Rather than computing θ and then $\tan\left(\frac{\theta}{2}\right)$, we can directly use the trigonometric functions

$$\sin\left(\frac{\theta}{2}\right) = \frac{|\boldsymbol{n}_l - \boldsymbol{n}_r|}{2} \qquad \cos\left(\frac{\theta}{2}\right) = \frac{|\boldsymbol{n}_l + \boldsymbol{n}_r|}{2} \tag{4.35}$$

and therefore

$$\tan\left(\frac{\theta}{2}\right) = \frac{|\boldsymbol{n}_l - \boldsymbol{n}_r|}{|\boldsymbol{n}_l + \boldsymbol{n}_r|} \tag{4.36}$$

To determine which way a shell is bending, $t\theta$ and therefore $\varphi(\theta)$ has to be a signed quantity. Without loss of generality, we define that $\varphi(\theta)$ is positive whenever two triangles are concave, meaning that their normals point away from each other, and negative whenever the two triangles are convex, meaning that their normals point towards each other. To differentiate between convex and concave triangles, we employ a similar approach as with the edge normals: We compute a vector $\boldsymbol{s}$ between the midpoints of the two triangles adjacent to the hinge. Assuming $\boldsymbol{s}$ points from K_l to K_r and the dot product $\boldsymbol{s} \cdot \boldsymbol{n_r}$ is larger than zero, the two triangles are concave. Otherwise, they are convex.

4.4.3 Newton Step

For each Newton step, the linear system of equation shown in Equation 4.30 has to be solved. In our implementation, the system is solved using the SparseLU solver from Eigen.
As long as no boundary conditions are enforced, the energy Hessian is symmetric, real-valued and positive definite. Therefore, we can alternatively solve the system approximatly using a conjugate gradient solver. Conjugate gradient solvers are significantly faster for well conditioned matrices, while still resulting in reasonably correct results. Since the energy Hessian is not necessarily diagonal dominant, we use the conjugate gradient solver with an incomplete cholesky preconditioner. In our experiments, the solver proofed to be up to 7 times faster, a number that might have been improved further by enabling multi threading.
Conjugate gradient solvers require the sparse matrix to be symmetric, meaning

that we can not employ the same method for boundary enforcement as for the heat flow computations in Section 3.1.4. However, since we defined the force on the boundary to always be zero, this reduces to simply setting both the respective rows and columns to zero.

4.4.4 Initial Boundary Displacement

It is important to note again that only small displacements are allowed to avoid instabilities and problems with face self intersection. A good indicator is the energy measure: If the energy increases between two timesteps, the displacement or the Newton step size should be decreased. Larger displacements can be done by iteratively enforcing a small displacement and minimizing the respective energy.

4.5 Mesh Independence

To verify the correctness of our implementation, we now show that the solution is mesh independent. To that end, we will use the same meshes as for showing mesh independence in the heat flow chapter, shown in Figure 4.2. The vertices at the top of the cylindrical ADMS are moved by $\Delta = (0.01, 0.01, -0.01)^T$ and fixed. The vertices at the bottom of the cylindrical ADMS are fixed as well, thereby enforcing a deformation of the structure and avoiding that the structure simply experiences a translation.

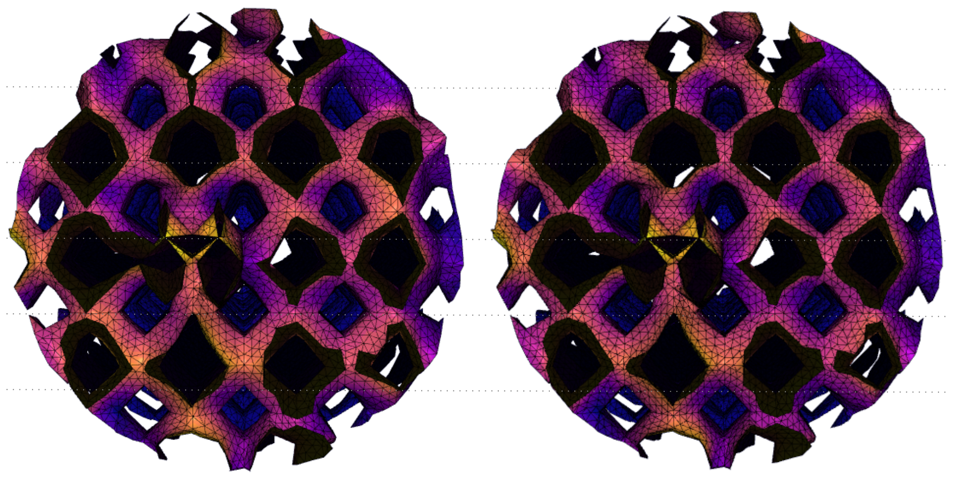

Figure 4.2: The nodal displacement from the top of the cylindrical ADMS on two different meshes that represent the same surface.

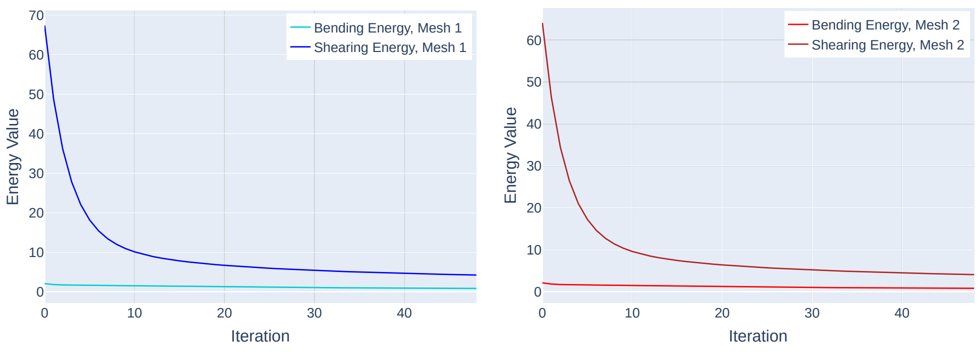

Figure 4.3: Change in bending and shearing energy of the two meshes. Clearly, the behavior of the energy is almost identical.

Cylinder	**Stiffness**
Cylinder 1	120.729
Cylinder 3	107.024

Table 4.1: Computed average properties of two different cylindrical ADMS with a wall width of $h = 3mm$

Cylinder	**Stiffness**
Cylinder 1	247.79
Cylinder 3	210.755

Table 4.2: Computed average properties of two different cylindrical ADMS with a wall width of $h = 6mm$

4.6 Results on ADMS

4.6.1 Chosen Paramters

To simulate the displacement resulting from a small external force in a structure, certain aforementioned parameters have to be defined. Namely the Young's Modulus and the Poissons Ratio, the wall thickness and a shearing and stiffness constant. Furthermore, the damped Newton method requires the choice of a step size γ and a convergence criteria.
According to the manufacturer of the printer used to print the ADMS, the ADMS should have a Young's Modulus of $1.470 \cdot 10^9 Pa$. This is in accordance with the Young's Modulus and Poisson-Ratio for the respective material found by Amado-Becker et al. [22]. For the following experiments, we will therefore assume a Young's Modulus of $E = 1.470 \cdot 10^9 Pa =$ and a Poisson's Ratio of 0.4. Together with the wall thickness, these values can then be used in Equation 4.6 to compute the bending stiffness k_{bend}.
Through different simulations we found that γ can be chosen according to mesh quality. The better the mesh, the less problems occur with choice of a larger γ. We stop the simulation once the energy delta between the current and previous Newton step is smaller than 0.01

4.6.2 Cylinder Structure

In this section, we explore the effects of a displacement of $\mathit{\Delta} = (0, 0, 0.02)$ on two of the three different ADMS already introduced in the heat flow computation chapter. The wall width is assumed to be constant, $h = 3mm$.The resulting displacement of the different nodes is visualized in Figure 4.4, the resulting stiffness is listed in Table 4.1. As one would expect, the cylinder with the smaller cells and tunnels proofs to be more stiff than the ADMS with larger cells towards the top of the structure.

4.6.3 Increase in Wall Width

As a next experiment, we increase the wall width from $h = 3mm$ to $h = 6mm$ while keeping the deformation constant at $\mathit{\Delta} = (0, 0, 0.02)$. This allows us to examine the influence of wall width on stiffness and the other mechanical properties. As expected, the increase in wall width cause the structures to be more stiff. The respective mechanical properties are listed in Table 4.2.

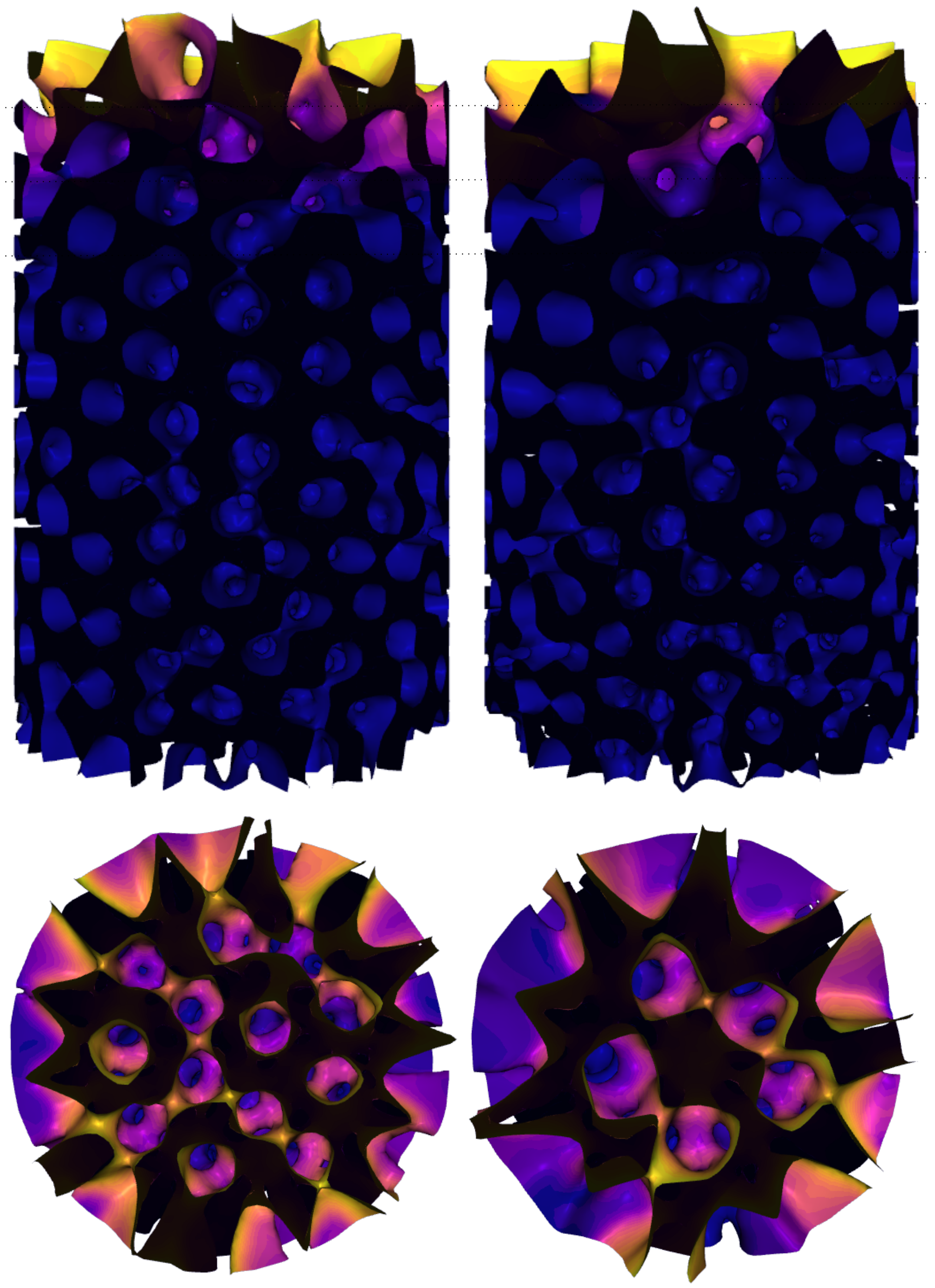

Figure 4.4: The nodal displacement from the top of the cylindrical ADMS on two different meshes.

Chapter 5

Conclusion and Outlook

In this semester thesis we implemented two different algorithms to simulate physical processes on thin, elastic structures.
The first algorithm allows for the computation of the heat flow through the structure. From the results, the bulk heat conductivity and the mean temperature were computed. To ensure correctness of the implemented algorithm, we conducted a set of experiments: First, we computed the heat flux on an annulus and compared the results to the known, real solution. We also showed that the accuracy of the computed solution increases when the mesh is refined. Next, we demonstrated that the solution is mesh independent. Last, we applied the algorithm to different ADMS and analyzed the influence of the exact ADMS structure on the temperature distribution, the bulk heat conductivity and the mean temperature. We found that large tubes cause temperature to diffuse more quickly. In this thesis, we only simulated the heat flux through the structure, ignoring possible heat exchange with the medium flowing through the structure. This is completely reasonable under the assumption that the other medium is air. For future work, it might be interesting to simulate the heat exchange between the ADMS and another medium, like for example water.
The second simulation aimed at computing the stiffness of of ADMS. To that end, we implemented a discrete shell algorithm that computes the structure deformation under a small boundary displacement. The deformation was then directly used to compute the stiffness of the different ADMS. Future work could focus on allowing for the simulation of larger displacements. To achieve this, one would have to implement collision detection that avoids the self intersection of different mesh faces. Furthermore, one could no longer use the infinitesimal strain assumption, requiring more complex functions for the computation of strain and stress. Another problem with the current implementation is that all printed materials are not technically isotropic and might break or deform easier in certain directions depending on their print orientation.

Bibliography

[1] Spherene. Spherene ag.

[2] J. Bergström, "5 - elasticity/hyperelasticity," in *Mechanics of Solid Polymers*, J. Bergström, Ed. William Andrew Publishing, 2015, pp. 209 – 307.

[3] R. S. Falk, *Finite Element Methods for Linear Elasticity.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 159–194.

[4] E. Oñate, *Thin Plates. Kirchhoff Theory.* Dordrecht: Springer Netherlands, 2013, pp. 233–290.

[5] J. Reddy, *Theory and Analysis of Elastic Plates and Shells, Second Edition*, ser. Series in Systems and Control. Taylor & Francis, 2006.

[6] N. Newmark, "A method of computation for structural dynamics," 1959.

[7] J. Nocedal and S. Wright, *Numerical optimization*, 2nd ed., ser. Springer series in operations research and financial engineering. New York, NY: Springer, 2006.

[8] F. Cirak, M. Ortiz, and P. Schröder, "Subdivision surfaces: a new paradigm for thin-shell finite-element analysis," *International Journal for Numerical Methods in Engineering*, vol. 47, no. 12, pp. 2039–2072, 2000.

[9] S. Green, G. Turkiyyah, and D. Storti, "Subdivision-based multilevel methods for large scale engineering simulation of thin shells," 01 2002, pp. 265–272.

[10] E. Grinspun, A. N. Hirani, M. Desbrun, and P. Schröder, "Discrete Shells," in *Symposium on Computer Animation*, D. Breen and M. Lin, Eds. The Eurographics Association, 2003.

[11] R. Tamstorf and E. Grinspun, "Discrete bending forces and their jacobians," *Graphical Models*, vol. 75, no. 6, pp. 362 – 370, 2013.

[12] H.-Y. Chen, A. Sastry, W. M. van Rees, and E. Vouga, "Physical simulation of environmentally induced thin shell deformation," *ACM Trans. Graph.*, vol. 37, no. 4, Jul. 2018.

[13] *Barycentric Coordinates.* London: Springer London, 2006, pp. 193–221.

[14] R. Hiptmair, "Lecture notes in numerical methods for partial differential equations," September 2020.

[15] L. Zhang, T. Cui, and H. Liu, "A set of symmetric quadrature rules on triangles and tetrahedra," *Journal of Computational Mathematics*, vol. 27, pp. 89–96, 01 2009.

[16] A. Jacobson, D. Panozzo *et al.*, "libigl: A simple C++ geometry processing library," 2018, https://libigl.github.io/.

[17] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[18] M. Botsch and O. Sorkine, "On linear variational surface deformation methods," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 1, pp. 213–230, 2008.

[19] M. P. do Carmo, *Differential geometry of curves and surfaces.* Prentice Hall, 1976.

[20] M. Wardetzky, M. Bergou, D. Harmon, D. Zorin, and E. Grinspun, "Discrete quadratic curvature energies," *Computer Aided Geometric Design*, vol. 24, no. 8, pp. 499 – 518, 2007, discrete Differential Geometry.

[21] R. Bridson, S. Marino, and R. Fedkiw, "Simulation of clothing with folds and wrinkles," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '03. Goslar, DEU: Eurographics Association, 2003, p. 28–36.

[22] A. Amado-Becker, J. Ramos-Grez, M. Yanez, Y. Vargas, and L. Gaete-Garretón, "Elastic tensor stiffness coefficients for sls nylon 12 under different degrees of densification as measured by ultrasonic technique," *Rapid Prototyping Journal*, vol. 14, pp. 260–270, 09 2008.

Appendix A

Code Structure

This chapter shortly details the used folder structure, describes how to build and execute the code and explains what the different executables do.

A.1 Folder Structure

The following folders can be found in the project folder:

- `executables` contains program code for the starting points and GUI of the implemented algorithms. The different starting points are explained in more detail in Section A.2.3.
- `include` containing the header files for the implemented algorithms.
- `output` is the folder that results are written into.
- `input` containing the meshes defining the boundaries, the meshes used for the convergence tests and for the ADMS experiments.
- `plotting` containing several jupyter notebooks that can be used to analyze and plot the results generated by the implemented algorithms.
- `src` containing the source files for the implemented algorithms.
- `tests` containing some basic unit tests.

A.2 Building and Executing the Code

A.2.1 Preconditions

To build and run the code accompanying this semester thesis, several software packages and external libraries are needed.
The following software should be installed beforehand:

- Git
- A `C++` Compiler capable of compiling `C++` standard 17
- CMake, at least version 3.10

The following external libraries are needed:

- libigl [16], the graphics library. For installation instructions and information about the needed third party software, the reader is referred to the libigl website. For CMake to find the installation, the installation folder should be put directly inside the project folder or at any other location listed in the `FindLIBIGL.cmake` file. For this semester thesis, libigl should be built with the dependencies `LIBIGL_WITH_OPENGL`, `LIBIGL_WITH_OPENGL_GLFW`, `LIBIGL_WITH_OPENGL_GLFW_IMGUI` and `LIBIGL_WITH_PNG` set to `on`.

- Eigen [17], the matrix operation library. This library gets installed through libigl.

- The unit test library Google-Test gets installed automatically when building the semester project.

- A working python installation together with the packages `jupyter`, `pandas`, `numpy` and `plotly` are needed to plot statistical results generated by the algorithms.

A.2.2 Build using CMake

Once all preconditions are satisfied, the code can be built using CMake. Let `$PROJECT_PATH` be the path pointing to the project folder. On Linux, the code can be built using the following commands in the terminal:

Listing A.1: Building the Code

```
$> cd $PROJECT_PATH
$> mkdir build
$> cd build
$> cmake ..
$> make
```

A.2.3 Running the Code

Compiling the code results in a total of 4 executables located in the build folder. The following list describes what these executables do.

`Semesterproject_main_heatequation`: This executable can be used to run the algorithm implemented in this thesis for computing the heat flow and several related functionals on a given mesh. One of four different test cases can easily be chosen my passing a number between `0` and `3` to the executable. Each test case comes with an already predefined mesh and boundary conditions. If no number is passed, the first test case is chosen automatically. Upon execution, `semesterproject_main_heatequation` starts the libigl viewer, showing a small toolbar on the left hand side and the chosen mesh on the right hand side. An example can be seen in Figure TODO. The heat equation is solved with the boundary conditions accompanying the boundary conditions whenever the button "solve" is pressed. The resulting heat diffusion is directly visualized on the displayed mesh. Mean temperature, heat flux with and without cutoff function as well as some mesh statistics are outputted on the command line.

`Semesterproject_main_deformation` : This executable can be used to run the algorithm implemented in this thesis for computing the deformation of a mesh. Similarly as for `Semesterproject_main_heatequation`, one can choose between three different meshes by passing a number between `0` and `2` to the executable. Upon execution, the libigl viewer is started. The deformation can be started by pressing the "solve" button. The resulting displacement of the different nodes is visualized directly on the mesh. The stiffness is outputted on the command line.

`Semesterproject_test_real_solution`: Executable that computes the heat diffusion as well as several functionals on six different annulus meshes of varying resolution. The computed solution is compared to the real solution. The results are written to a `csv` file and can be evaluated using the jupyter notebooks saved in the `plotting` folder. This is the executable that was used for generating the results described in Section 3.4.1.

`Semesterproject_convergence_test_vs_itself`: Executable that computes the heat diffusion as well as several other functionals on given test meshes, for example of two meshes representing the same surface. The results are written to a `csv` file and can be evaluated using the jupyter notebooks saved in the `plotting` folder. This is the executable that was used for generating the results described in Sections 3.4.2 and 4.5.

A.3 Modules and Important Functions

The code accompanying this semester thesis consists of three modules: `utils`, `heat_flow` and `discrete_shell`. The modules `heat_flow` and `discrete_shell` are mutually independent of each other, although both require a reference to the `utils` module. The following section give an overview over these modules.

A.3.1 Utils

The `utils` module contains a set of helper functions. The following list shortly describes the most important files and the type of functions they contain.

- `view_utils` contains functions that can be called from the libigl viewer. For example, one of the functions can be used to take normed screenshots of a currently displayed mesh. Others are there to help with visualizing the boundary points of a mesh.
- `matrix_utils`, `str_utils` and `math_utils` contain everyday helper functions that can be useful during programming.
- `geometry_utils` contains functions around cleaning and analyzing a mesh.
- `boundary_condition` contain classes around selecting and managing boundary points. The `boundary_condition_factory` returns premade and initialized boundary conditions for various ADMS shapes.
- `triangle` contains triangle face specific functions, such as computing the barycentric coordinates, face normals, edge lengths and more.

A.3.2 Heat Flow

The `heat_flow` module contains functions around solving the heat diffusion equation.

- `fem` is the main file in this module. It contains the function `solveStationaryHeatEquation` that solves the heat diffusion equation for a set of vertices, faces and boundary conditions. Other functions in this file are helper functions for assembling the local and global Galerkin matrices.
- `heat_functionals` contains functions for computing the different functionals described and used in this thesis

A.3.3 Discrete Shell

The `heat_flow` module contains functions around computing the deformation of a structure.

- `discrete_shell` is the main file in this module. It contains the fucntions for computing the different kind of energy gradients, as well as for executing the necessary Newton steps.
- `displacement_functional` contains functions for computing the different functionals described and used in this thesis

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Simulations on Minimal Surfaces

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Fritschi	Lea

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Tagelswangen, 10.10.2020	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.