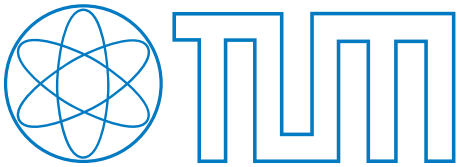

Technische Universität München

TUM School of Natural Sciences

Master Thesis

Computational Models for the Growth of Closed Bacterial Cell Envelopes

Paul Schulze

Abstract

Bacterial shape plays a crucial role in their survival and function. The shape of bacteria is mainly determined by the cell wall, a polymeric envelope providing structural support and protection to the cell. Interestingly, bacteria maintain their distinct shape with outstanding precision and its morphology is highly robust to perturbations of the external conditions. While the proteins involved in the growth of the cell wall have been studied extensively, how the local properties of the cell envelope direct the growth process to achieve such precise morphological control remains a puzzle. In this context, mechanical and geometrical cues have been proposed as potential feedback mechanisms contributing to robust shape preservation.

The central objective of this thesis is to explore bacterial shell growth driven by mechanical and geometrical laws. To this end, a specialized simulation framework is developed. We model the cell wall as a linear elastic material and shell growth as a stochastic process with rates defined in terms on the local mechanical or geometrical properties.

Two models of the bacterial shell are implemented. First, a spring-based model is developed. This model is computationally efficient and provides a simplified framework to understand cell wall growth. This model however shows undesired deviations from linear elasticity. This motivated the development of an alternative model based on finite elements (FEM). The model introduces localized growth to the classical mechanics finite element framework and shows the expected response of linear elastic material. Both models are integrated into a unified simulation package. The computational package provides a practical and insightful tool to explore cell envelope growth, thus contributing to the study and the understanding of bacteria and their remarkable adaptability.

We use the FEM setup to study growth in spherical and rod-shaped vessels. We find that a strain-dependent growth law reduces the roughness of the surface and surface stresses compared to purely random growth. Furthermore, inspired by the observations in rod-shaped bacteria, we find a growth law combining strain and curvature information leading to robust lengthening of the cell with no change in the cell's radius.

This thesis reinforces the role of strain sensing in shape preservation and offers valuable insights into the operation of strain-based growth.

Zusammenfassung

Die Form von Bakterien spielt eine entscheidende Rolle für ihr Überleben und ihre Funktion. Die Form von Bakterien wird hauptsächlich durch die Zellwand bestimmt. Diese Hülle bestehend aus Polymeren, die der Zelle mechanische Unterstützung und Schutz bietet. Interessanterweise behalten Bakterien ihre spezifische Form mit herausragender Präzision und ihre Morphologie ist äußerst robust gegenüber Änderungen in Umweltbedingungen. Die Proteine, die am Wachstum der Zellwand beteiligt sind, sind viel erforscht. Wie die lokalen Eigenschaften der Zellhülle den Wachstumsprozess beinflussen, um eine solch präzise morphologische Kontrolle zu erreichen, bleibt jedoch ein Rätsel. In diesem Zusammenhang wurden mechanische und geometrische Signale als potenzielle Rückkopplungsmechanismen vorgeschlagen.

Das Hauptziel dieser Arbeit ist die Erforschung des bakteriellen Hüllenwachstums, das durch mechanische und geometrische Signale gesteuert wird. Zu diesem Zweck wird ein spezialisiertes Simulationsprogramm entwickelt. Wir modellieren die Zellwand als ein linear elastisches Material und das Wachstum der Hülle als stochastischen Prozess mit Raten, die Abhängig von den lokalen mechanischen und geometrischen Eigenschaften berechnet werden.

Es werden zwei Modelle der bakteriellen Hülle implementiert. Zunächst wird ein federbasiertes Modell entwickelt. Dieses Modell ist rechnerisch effizient und bietet einen vereinfachten Rahmen für das Verständnis des Zellwandwachstums. Es weist jedoch Abweichungen von der linearen Elastizität auf. Dies motivierte die Entwicklung eines alternativen Modells, das auf finiten Elementen basiert. Das Modell führt lokalisiertes Wachstum ein und die Hülle verhält sich wie ein linear elastisches Material. Beide Modelle werden in ein einheitliches Simulationspaket integriert. Die Simulationssoftware bietet ein praktisches und aufschlussreiches Werkzeug zur Erforschung des Zellhüllenwachstums, und trägt damit zum Verständnis von Bakterien und ihrer bemerkenswerten Anpassungsfähigkeit bei.

Wir verwenden die FEM-Simulation, um das Wachstum in kugelförmigen und stäbchenförmigen Gefäßen zu modelliern. Wir stellen fest, dass ein dehnungsabhängiges Wachstumsgesetz die Rauheit der Oberfläche im Vergleich zum rein zufälligen Wachstum reduziert. Außerdem finden wir, inspiriert von den Beobachtungen bei stäbchenförmigen Bakterien, ein Wachstumsgesetz, das Dehnungs- und Krümmungsinformationen kombiniert und zu einer Verlängerung der Zelle führt, ohne dass sich der Radius der Zelle ändert. Diese Arbeit unterstreicht die Rolle der Dehnungserfassung bei der Formerhaltung und bietet wertvolle Einblicke in die Funktionsweise von dehnungsbasiertem Wachstum.

Contents

1 Introduction **3**
- 1.1 Shapes of bacteria . . . 3
- 1.2 Cell wall . . . 4
- 1.3 Elastic deformations . . . 5
- 1.4 Plastic deformations and growth . . . 5
- 1.5 Overview of the thesis . . . 6

2 Spring based model **8**
- 2.1 Theoretical background . . . 9
  - 2.1.1 Modeling the surface as a network of springs . . . 9
  - 2.1.2 Modeling the growth dynamics . . . 10
- 2.2 Computational methods . . . 12
  - 2.2.1 Design principles . . . 13
  - 2.2.2 Notable features . . . 13
  - 2.2.3 Simulation process flow . . . 14
- 2.3 Results and discussion . . . 15
  - 2.3.1 Spherical bacterial shell under turgor pressure . . . 15
  - 2.3.2 Strain-based growth of the spherical bacterial shell . . . 18
  - 2.3.3 Growing the rod-shaped bacterial shell . . . 20
- 2.4 Limitations of the spring based model . . . 22

3 Finite element based model **24**
- 3.1 The finite element method as a computational technique . . . 24
  - 3.1.1 Stress-strain relations . . . 25
  - 3.1.2 Computing the global strain energy . . . 25
  - 3.1.3 Computing the boundary term in the potential energy 26
- 3.2 Computational methods . . . 27
  - 3.2.1 Initial Strains in COMSOL . . . 27
  - 3.2.2 Local growth with initial strains . . . 28
  - 3.2.3 Initial area correction . . . 32
  - 3.2.4 Simulation process flow . . . 33
  - 3.2.5 Technologies . . . 33
- 3.3 Results and discussion . . . 34

3.3.1 Spherical bacterial shell under turgor pressure 35
3.3.2 Strain-based growth in the spherical bacterial shell . . 35
3.3.3 Strain-based growth in the rod-shaped bacterial shell . 39
3.3.4 Directed growth in the rod-shaped bacterial shell . . . 41
3.3.5 Mechanical and geometric cues for directional and positional sensing 42
3.4 Conclusion 48

4 Summary and Outlook **49**

A Structure of the simulation package **51**

B Improving the execution time of the spring based model simulation **57**
B.1 Moving from NumPy to Numba 57
B.2 Moving from Numba to a custom C extension 58

Acknowledgments **63**

Chapter 1

Introduction

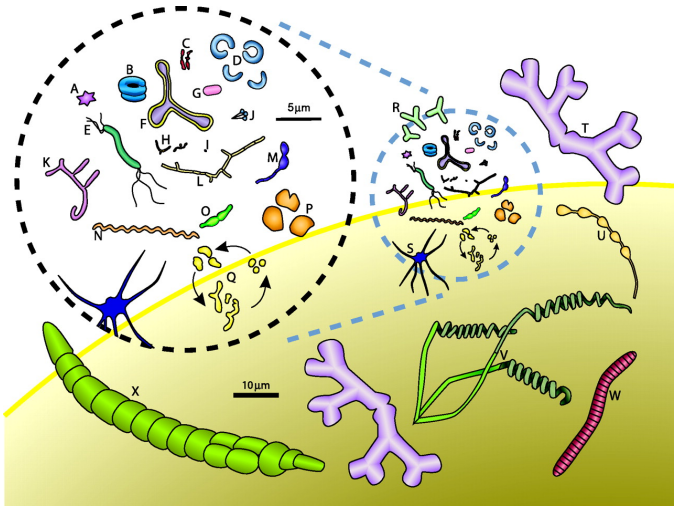

Figure 1.0.1: Shapes and sizes of bacteria. Reproduced from [8]

1.1 Shapes of bacteria

Bacteria can be found in a large variety of shapes and sizes, as shown in Fig.1.0.1. These shapes are not accidental, but of high biological importance for the bacteria. The cell shape impacts a bacteria's ability to survive in a multitude of ways: Nutrient access, cell division and segregation, attachment to surfaces, passive dispersal, active motility, polar differentiation, the need to escape predators, and the advantages of cellular differentiation [8]. Although there are many possibilities, bacteria select their shape consistently from generation to generation, and morphological changes can be traced through evolutionary lines. Some cells even make adjustments to their shape in response to changing conditions. These factors indicate that

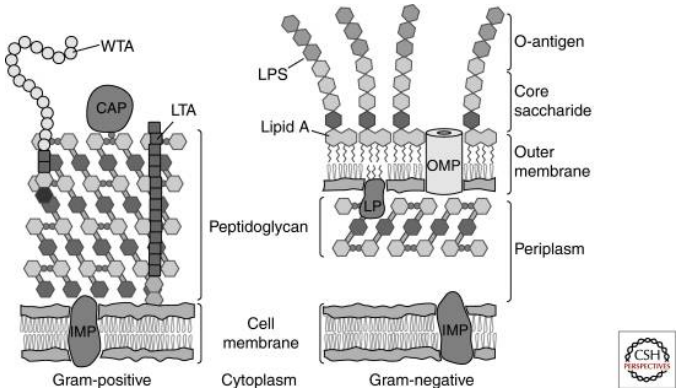

Figure 1.2.1: Gram-positive and gram-negative cell envelopes.
CAP = covalently attached protein; IMP, integral membrane protein; LP, lipoprotein; LPS, lipopolysaccharide; LTA, lipoteichoic acid; OMP, outer membrane protein; WTA, wall teichoic acid. Reproduced from [13]

the geometrical shape of bacteria is a selectable trait that aids in survival [6].

1.2 Cell wall

The bacterial shape is determined by the peptidoglycan (PG) layer, which is composed of glycan chains connected by short peptides. The mechanical properties of cell walls are different for Gram-positive and Gram-negative bacteria. Cell envelopes of Gram-positive and Gram-negative bacteria are shown in Fig. 1.2.1.

Gram-negative bacteria have a complex cell envelope consisting of the outer membrane layer, the PG layer and the inner membrane layer [13]. The outer membrane is a lipid bilayer. The PG cell wall is made up of repeating units of the disaccharide N-acetyl glucosamine-N-actyl muramic acid, which are cross-linked by pentapeptide side chains [10]. For rod-like bacteria, the gylcan chains are organized in hoops perpendicular to the longitudinal direction, which are linked by peptides [9]. The inner membrane layer is a phospholipid bilayer.

Gram-positive bacteria by contrast have no outer layer but often live in much harsher environments and have a PG cell wall that is many times thicker compared to gram-negative bacteria. The PG structure is reinforced by anionic polymers like lipoteichoic acid (LTA) and wall teichoic acid (WTA). Attached on the surface are proteins, the composition of which can depend on the environment and growth conditions [5].

1.3 Elastic deformations

The increased thickness of the PG layer in gram-positive bacteria allows it to withstand high pressures. These turgor pressures are caused by a higher solute concentration in the cell. This produces an osmotic pressure, which causes solution to flow into the cell [7]. This inflates the cell, which is only being contained by the cell wall. Tab. 1.3.1 shows PG layer thicknesses and turgor pressures observed in *E. coli* and *B. subtilis*, which are Gram-negative and Gram-positive bacteria, respectively. The peptigoglycan cell wall can be modeled as an elastic sheet subject to elastic and plastic deformations [19], [20], [23].

	Peptidoglycan layer thickness	Turgor pressure
Escherichia coli	1.5–6.5 nm [10]	0.3 and 2–3 atm [14], [3]
Bacillus subtilis	30 nm [18]	20 atm [4]

Table 1.3.1: Peptidoglycan layer thickness and turgor pressure of *E. coli* and *B. subtilis*. Adapted from [19].

1.4 Plastic deformations and growth

The cell wall is constantly remodeled by a process that involves a multitude of proteins, which are depicted in Fig. 1.4.1. PG synthases assemble the PG network from PG monomers. (A), (B) Glycosyltransferases polymerize PG monomers into strands, which are cross-linked by (A), (C) Transpeptidases. (D) Inner membrane associated proteins synthesize PG monomers; (E) Flippases flip the monomers across the inner membrane. (F) Scaffolding proteins bind multiple synthases and inner membrane proteins, thereby localizing growth.

A variety of hydrolases on the other hand modify existing PG structures by breaking specific bonds. (G) Endopeptidase can break cross-links and peptide links, the latter of which can also be broken at different positions by (H) Carboxypeptidase. (I) Glucosidase breaks glycan strands, (J) Amidase removes entire peptide stems [25].

The proteins involved in the growth process are orchestrated in a way that leads to robust shape preservation in bacteria. *E. coli* cells elongate while maintaining their radius with high precision. Experiments have shown that cells recover their characteristic shape through growth even if the shape is disturbed in some way, for example by bending [19].

Although the mechanics of cell wall synthesis have been well studied, the feedback between cell properties and growth is not well understood. Three theories of shape preservation have been put forward:

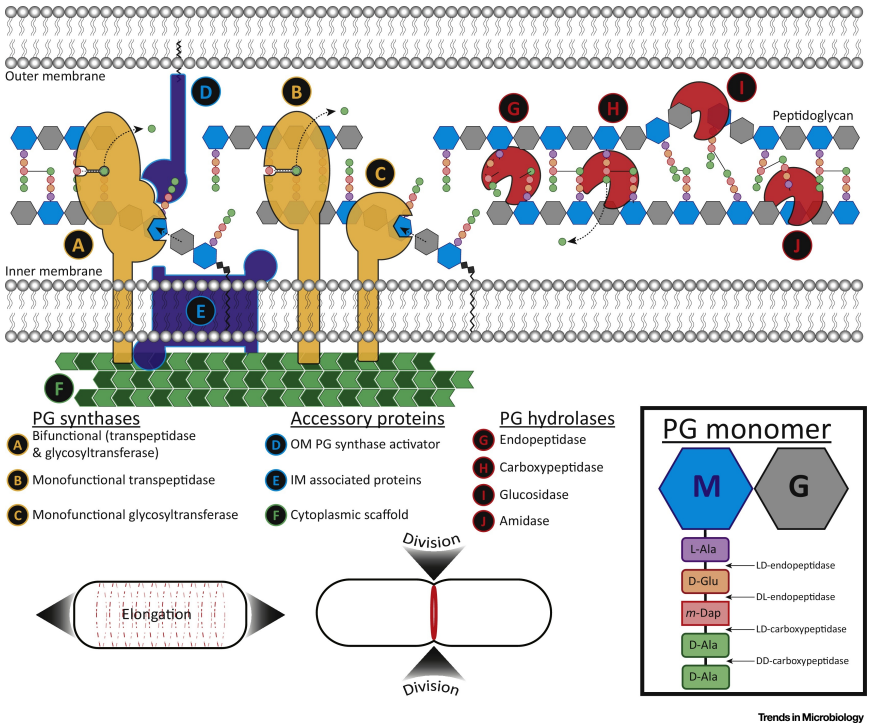

Figure 1.4.1: Simplified schematic depiction of proteins involved in peptoglycan remodeling. Reproduced from [25].

2) A protein related to the peptidoglycan elongation machinery provides a geometry sensing mechanism [21].
3) A strain sensing mechanism that allows for the preferred insertion of new glycan strains at regions of high strain [19].

1.5 Overview of the thesis

I wrote computational models to study bacterial cell growth. The simulation package was used to explore growth laws resulting in robust cell shape. I studied spherical and rod-shaped bacteria and tested various strain- and curvature-sensing mechanisms that leads to controlled growth of micro and macroscale properties.

I employed two different simulation strategies to model the mechanical properties of the bacterial cell walls:

1. Spring based model (Section 2). The cell is described by a triangulated mesh connected by springs. This model maps to a linear elastic continuum for small strains. Growth of the cell envelope is modeled by enlarging the rest lengths of the springs.

2. Finite element based model (Section 3). In this model the surface is described by a triangulated surface. The constitutive equations of lin-

ear elasticity are solved for each triangular element following a static finite element approximation. This model leverages the COMSOL Multiphysics package which is wrapped in the core Python code of the simulation. The growth of the shell is implemented by modifying the dimensions of the triangles.

Chapter 2

Spring based model

I present a physical model that captures the mechanical properties of the bacterial shell and its response to the internal turgor pressure. Growth of the shell is modeled as a stochastic process with rates dependent on strain and curvature.

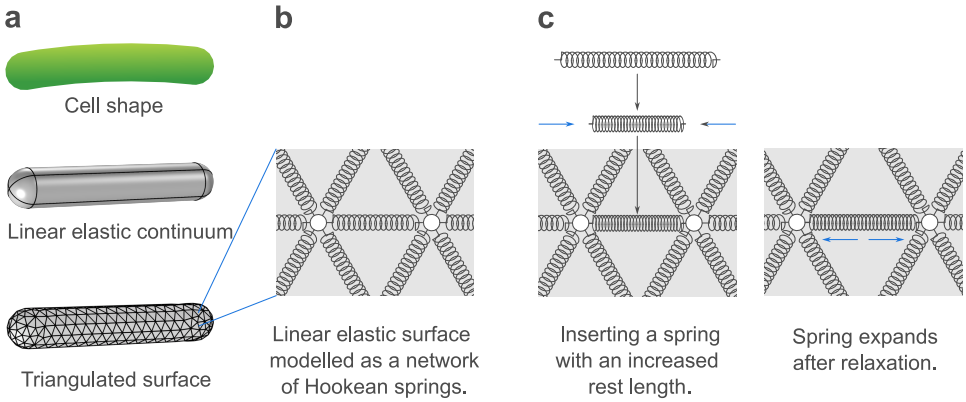

Figure 2.0.1: Growth model of the bacterial shell based on a spring network.
a, The bacterial shell modeled as a surface with linear elastic material. The surface is discretized into a triangle mesh. **b**, Each of the edges in the triangulated surface represents a Hookean spring. The elastic properties of this network map to a linear elastic material [2]. **c**, Local growth is realized by increasing the rest lengths of springs.

The mechanics of the bacterial shell are modeled as linear elastic shell with finite thickness. The geometry of the bacterium is geometry is discretized by triangulation, yielding a triangle mesh of the surface. The physical model is built upon the mesh geometry by assigning physical properties to mesh attributes. In the spring based model, each of the edges in the mesh represents a spring. The triangles are subject to an outwards pointing

pressure force. Local growth is realized by selectively increasing rest lengths of springs in the network. This is illustrated in Fig. 2.0.1. The following chapters present the theoretical basis of the physical model and the stochastic growth simulations. Lastly, I present my results from growth simulations with spherical and rod-shaped bacterial shells.

2.1 Theoretical background

In this chapter I present the theoretical basis for the spring based physical model of the bacterial shell and the stochastic growth simulations.

2.1.1 Modeling the surface as a network of springs

I model the elastic properties of the shell surface with a network of springs. Each of the edges in the triangulation mesh represents a harmonic spring with stiffness k_s and rest length $l_{0,e}$. I use Hooke's law to compute the energy of the springs (Eq. 2.1.1). I model the bending stiffness of the shell as an energetic cost to the bending of neighboring mesh triangles. The spring energy and the bending energy capture the mechanical properties of the bacterial shell. The effect of turgor pressure is modeled as normal forces on the mesh triangles. The complete energy equation reads

$$E = \underbrace{\sum_{e}^{\text{edges}} \frac{k_\mathrm{s}}{2}(l_e - l_{0,e})^2}_{\text{Spring energy}} + \overbrace{\sum_{\substack{\{i,j\} \\ \text{(adjacent)}}}^{\text{triangles}} k_\mathrm{bending}(1 - \vec{n}_i \cdot \vec{n}_j)}^{\text{Bending energy}} \underbrace{-pV}_{\text{Pressure energy}}, \quad (2.1.1)$$

where k_bending is the bending stiffness, $\{i, j\}$ are adjacent triangles and $\vec{n}_i$, $\vec{n}_j$ are their normal vectors, p is the turgor pressure and V is the volume enclosed by the surface. The resulting forces are illustrated schematically in Fig. 2.1.1. Equation 2.1.1 captures the mechanical properties of the bacterial shell surface and its response to the turgor pressure.

The spring stiffness $k_s$and bending stiffness k_bending can be mapped to properties of a continuous linear elastic surface with a 2D Young's modulus Y and bending rigidity κ_b

$$Y = \frac{2}{\sqrt{3}} k_s, \quad (2.1.2)$$

$$\kappa_\mathrm{b} = \frac{\sqrt{3}}{2} k_\mathrm{b}, \quad (2.1.3)$$

The continuous bending rigidity κ_b can also be computed from the 3D Young's modulus of the surface E and the thickness of the surface t as

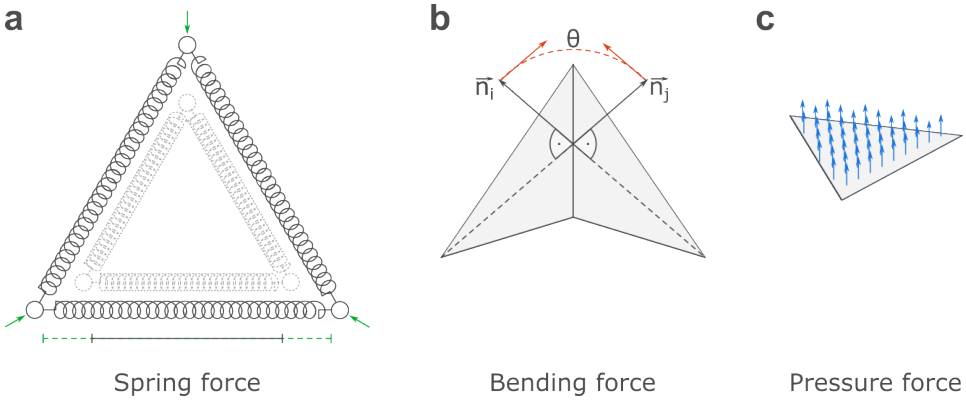

Figure 2.1.1: Spring energy and bending energy.
a, Springs in the network are modeled as harmonic springs with stiffness k_s. Each spring has its independent rest length $l_{0,e}$. **b**, The bending energy of two adjacent triangles is proportional to $1 - \cos(\theta)$, where θ is the angle between their normal vectors $\vec{n}_i$, $\vec{n}_j$. **c**, The pressure force on a given triangle is proportional to its area.

$$\kappa_b = \frac{Et^3}{12(1-\nu)}, \tag{2.1.4}$$

where $\nu = 1/3$ [2] is the Poisson's ratio [1]. This allows the mapping from a given linear elastic surface with Young's modulus E and thickness t

$$E, t \to k_s, k_{\text{bending}} \tag{2.1.5}$$

to a spring network with spring stiffness k_s, and bending stiffness k_{bending}. In this thesis, I consider the case of Gram-negative bacteria and assume a Youngs's modulus of 50 MPa and a thickness of 2 nm [19]. The growth dynamics are assumed to be slow compared to the equilibration of the surface, the surface is modeled in equilibrium. The equilibrium configuration of the surface is found in the minimum of the surface energy 2.1.1. The implemented software package applies a conjugate gradient algorithm to this problem.

2.1.2 Modeling the growth dynamics

I realize local growth by selectively increasing rest lengths of springs. At time t, each edge can be transformed from $l_{0,e}$ to $l_{0,e} + \delta l$ with a rate $\lambda(\vec{r})$, where $\vec{r}$ are local surface properties and the unit of λ is t^{-1}, so that

$$l_{0,e} \xrightarrow{k_e} l_{0,e} + \delta l. \tag{2.1.6}$$

I can write the the master equation for a given edge as

$$\partial_t p_e(l_{0,e} + \delta l, t) = \lambda_e p_e(l_{0,e}, t). \tag{2.1.7}$$

A sample from the solution of the master equation is generated by simulating the time evolution of the edges as a discrete stochastic process. The Gillespie algorithm generates time steps and selects reactions based on the Monte Carlo Method. Two random numbers $r_1,\ r_2\ \epsilon\ [0,\ 1]$ are generated, from which the time step τ is computed as

$$\tau = \frac{1}{\sum_e \lambda_e} \log\left(\frac{1}{r_1}\right) \tag{2.1.8}$$

and the selected growth reaction e is computed as

$$e = \text{smallest integer for which} \sum_{e'=0}^{t} \lambda_{e'} > r_2 \sum_e \lambda_e. \tag{2.1.9}$$

The selected reaction transforms the edge e from $l_{e,0} \rightarrow l_{e,0} + \delta l$, where δl is a fraction of the mean initial edge length as $\delta l = \epsilon \cdot \langle l_{e,0} \rangle$ with $0 < \epsilon \leq 1$. Let the mapping from local properties to growth reaction rate

$$f(\vec{r}) = \lambda \tag{2.1.10}$$

be called a *growth rule*. Negative growth reaction rates are capped to zero. The growth rule determines the growth reaction rates and the growth trajectory. In the simulations the growth reaction rate is always multiplied with an associated reaction area before applying the Gillespie algorithm. The reaction area occupied by a given edge in the mesh is the area spanned by its two vertices and the two neighboring triangle centroids. This ensures that the reaction rates per surface area are independent of the size distribution of the mesh triangles.

2.2 Computational methods

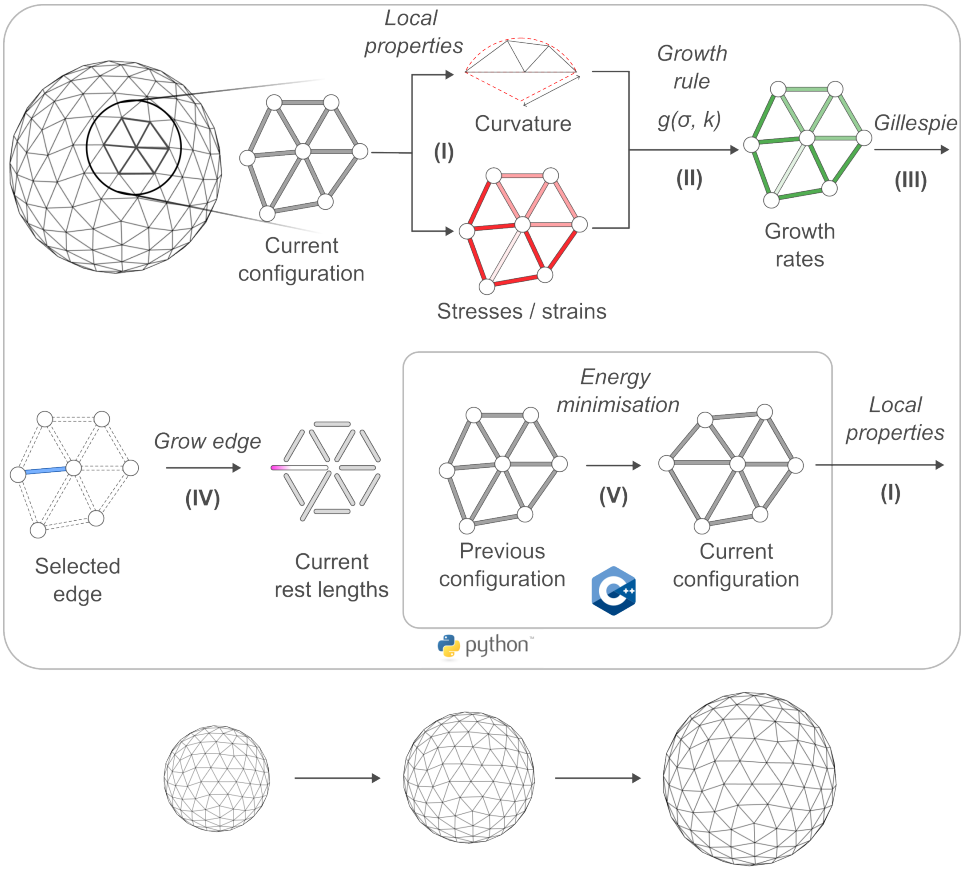

Figure 2.2.1: Simulation process loop.
The growth simulation involves a loop that consists of 5 steps.

- **(I)** Compute curvature and stresses of the current configuration.
- **(II)** The growth rule $g(\sigma, k)$ computes growth reaction rates of edges in the mesh.
- **(III)** One edge is selected with the Gillespie algorithm.
- **(IV)** The rest length of the selected edge is grown.
- **(V)** The new current configuration is computed from the rest lengths and physical parameters. The previous configuration is used as initial guess for the minimum.
  → Go to **(I)**.

The simulation package is based on the energy function and Gillespie algorithm introduced in the previous section. Python was chosen as the main programming language, though I implemented some modules in C++ and Java for performance and compatibility reasons. I was responsible for the design and implementation of the main code base. I also wrote the

documentation and managed the git of the project. Cesar Lopez Pastrana implemented a vertex-based stress computation, a kinetic Monte Carlo routine and provided a program for the generation of the initial mesh geometry. Julius Lehmann was responsible for implementing a fast and reliable way to compute the curvature and also worked on various improvements of the package, including the documentation.

2.2.1 Design principles

The software package is divided into modules with each module being responsible for a single task: loading meshes, storing shell data, computing the energy and gradient, minimization, computing curvature, computing stresses, logging data and data output. For a full list of modules see Tab. A.0.1. This allowed us to extend the functionality of the package with new modules and substitute one module for another, for example when implementing faster energy and gradient computations (see B), and the finite element based model and COMSOL interface. All class interfaces abstract away from the implementational details and are tailored to their application in the context of the growth simulation. I kept the class hierarchies flat in order to avoid complex inheritance patterns. Class relationships are implemented through composition instead of relying on inheritance, making use of dependency injection. For an overview over the class relationships in the program see Fig. A.0.1 in the appendix.

2.2.2 Notable features

The package is PEP8 compliant [32], all files are well documented and type hints are used throughout the package.

The package offers implementations of two different physical models of the shell surface. In addition to the spring based model, a finite element based model implemented on the basis of COMSOL Multiphysics (see 3.1) is included.

I implemented a custom C++ extension with Cython [29] which reduced the computational time by over 160 times (see B).

All observables of interest related to the state of the shell such as volume, stresses and surface roughness are automatically logged to the Tensorboard visualization toolkit [33] during growth. This allowed me to track metrics and visualize the 3D geometry during growth simulations (see Fig. A.0.2).

All logged data is combined into a single pandas DataFrame and exported as Pickle and JSON file. The data output also includes a copy of the shell state in the Pickle file format, which can be used to continue the simulation. A zipped folder with .OBJ files for OVITO [34] or Cinema 4D [27] is exported for visualization and rendering.

2.2.3 Simulation process flow

The execution of the simulation can be divided into two stages. In the initialization stage, a triangulated mesh of the initial, unpressurized bacterial shell geometry is loaded into the simulation. The pressurized configuration is found with a conjugate gradient algorithm implemented in the SciPy package [26]. This concludes the first stage.

The growth stage consists of an iterative process that computes rates from local properties, selects an edge to grow with the Gillespie algorithm and then minimizes the energy to find the new current configuration. The process is illustrated in Fig. 2.2.1. Local properties are computed from the current configuration of the bacterial shell. A growth rule computes the growth reaction rate for each edge based on the curvatures and stresses / strains. An implementation of the Gillespie algorithm selects a the growth reaction of an edge. The rest length of the selected edge is grown and the energy is minimized to find a new configuration. The minimization routine repeatedly calls the energy and gradient computation functions. These functions represent a bottleneck in the computational time and are implemented in Cython and compiled to C++ code.

2.3 Results and discussion

I first compare the pressurization behavior expected from linear elasticity theory with the response of a spherical shell in the spring model and discuss the differences that I observed. I then show that a spherical and rod-shaped bacterial shells can benefit from strain cues in their growth. I finally present the limitations of the spring based model.

2.3.1 Spherical bacterial shell under turgor pressure

This chapter gives an overview over the expected pressurization behavior from linear elasticity. It will serve as a reference in the characterisation of the behavior of the bacterial shell under pressure in the spring based model. A sphere with a linear elastic surface material and radius r_0 is pressurized to pressure p. The expected surface stress is

$$\langle \sigma_{\theta\theta} \rangle = \langle \sigma_{\phi\phi} \rangle = \frac{p r_0}{2}. \tag{2.3.1}$$

The stress-strain relations for a linear elastic material are

$$\langle \varepsilon_{\phi\phi} \rangle = \frac{1}{Et}(\langle \sigma_{\phi\phi} \rangle - \langle \sigma_{\theta\theta} \rangle \nu) \tag{2.3.2}$$

and the surface strain is

$$\langle \varepsilon_{\phi\phi} \rangle = \frac{r - r_0}{r_0}, \tag{2.3.3}$$

where r is the pressurized radius. Substituting equations 2.3.1 and 2.3.3 into equation 2.3.2 yields

$$r = r_0(1 + \frac{p r_0}{2Et}(1 - \nu)), \tag{2.3.4}$$

which relates the pressurized radius r to the material properties, the undeformed radius r_0 and the pressure p.

I next explored the relation of pressurized radius r and pressure p in simulation. The result is presented in Fig. 2.3.1a. The spring network properties map to a linear elastic surface for small pressures, but large pressures produce non-linear behavior. A critical pressure p_{crit} exists beyond which no energy minimum exists.

The behavior can be understood by considering the balance between the spring forces and the pressure forces. The spring forces scale linearly with the radius, while the pressure forces scale quadratically with the radius. The effect of the pressure is limited to a shift in the energy minimum for small pressures, but is unconfined for large pressures. To understand and quantify the behavior I set up a toy model with a simplified energy function. Let the only allowed change to the shell be a scaling in size that preserves the shape, as is expected for the pressurisation of a spherical body with

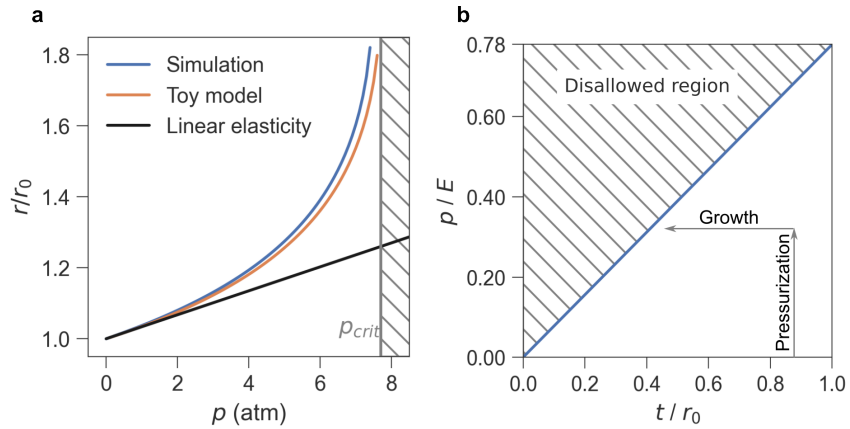

Figure 2.3.1: Spherical shell toy model and phase space.
a, Simulated pressurization of spherical shell with r_0=100 nm, $E = 50\,\mathrm{pN\,nm^{-2}}$ and t=2 nm (blue). Expected response from linear elasticity (eq. 2.3.4) in black. Toy model pressurization behavior in orange. Critical pressure p_{crit} indicated with vertical grey line. **b**, Predicted disallowed region in phase space where no energy minima exist. The allowed region is defined by $\frac{p}{E} < M\frac{t}{r_0}$, $M \approx 0.78$ for the given spherical mesh. This imposes both a limit in the allowed pressures and a limit in growing the undeformed radius r_0.

isotropic material properties. The volume can be written in terms of the pressurized radius r as $V \approx \frac{4}{3}\pi r^3$.

The scaling effects the radius r and edge lengths as

$$r = c \cdot r_0, \tag{2.3.5}$$

and

$$l_e = c \cdot l_{e,0} \tag{2.3.6}$$

where c is the factor by which the shell is scaled. Angles are preserved, which results in a constant bending energy. This permits the energy function to be written as a function of the scaling factor

$$E(c) = \sum_{e}^{\text{edges}} \frac{k_\text{s}}{2} l_{0,e}^2 (c-1)^2 - p\frac{4}{3}\pi r_0^3 c^3 + \text{const.}, \tag{2.3.7}$$

which can be used to find the pressurized radius where the energy is minimized. The derivative of the energy with respect to the scaling factor c

$$\frac{\mathrm{d}E(c)}{\mathrm{d}c} = k_\text{s} \sum_{e}^{\text{edges}} l_{0,e}^2 (c-1) - p4\pi r_0^3 c^2 \stackrel{!}{=} 0 \tag{2.3.8}$$

yields the energy minimum. The solution for c is a real number only if the relation

$$\frac{k_\text{s} \sum_e l_{0,e}^2}{p4\pi r_0^3} > 4 \tag{2.3.9}$$

holds. Substituting with Young's modulus E and thickness t as $k_\text{s} = \frac{\sqrt{3}}{2} Et$, gives

$$\frac{p}{E} < M\frac{t}{r_0} = \frac{\sqrt{3}}{2\pi} \frac{\sum_e l_{0,e}^2}{r_0^2} \frac{t}{r_0}, \tag{2.3.10}$$

where the factor M depends on the size distribution of rest lengths $l_{0,e}$ in relation to the radius of the undeformed sphere. M depends on the triangulation procedure of the shell surface. An energy minimum only exists if the relation 2.3.10 holds true.

Plotting the solution for $c = r/r_0$ in Fig. 2.3.1a, I observed good agreement with the data obtained from the simulation. Inequation 2.3.10 revealed a disallowed region in phase space where no energy minimum exists, shown in Fig. 2.3.1b. This region is approached with increasing pressure p and by growing undeformed radius r_0. As the disallowed region is approached, hyperelastic behavior emerges. These findings highlight the importance of carefully selecting initial parameters to stay in a well-behaved region of phase space.

2.3.2 Strain-based growth of the spherical bacterial shell

I quantified the shape of the spherical shell with the set of goals and corresponding metrics.

Goal	Metric
1. Increasing the size of the shell	Volume V
2. Conserving the spherical shape of the shell	Asphericity b
3. Conserving the smoothness of the surface	Roughness R

Table 2.3.1: Objectives for growing the spherical bacterial shell.

The volume served as an indicator of cell size. Fig. 2.3.2 shows a schematic view of shape deviations and how they score on the shape measures. Small-scale deviations were quantified with the roughness measure R. Large-scale deviations were quantified with the asphericity measure b. The asphericity of a three-dimensional shape is computed by comparing the

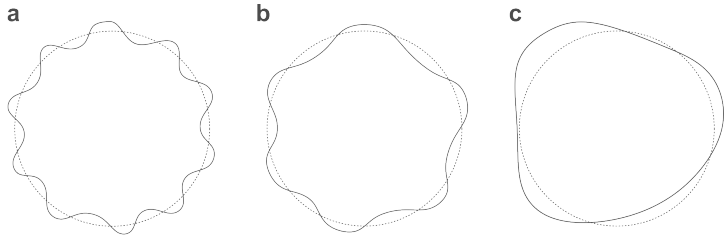

Figure 2.3.2: Schematic view of shape deviations from spherical
a, Low asphericity, high roughness.
b, Moderate asphericity, moderate roughness.
c, High asphericity, low roughness

principal moments λ_x, λ_y and λ_z as

$$b = \lambda_z^2 - \frac{1}{2}(\lambda_x^2 + \lambda_y) \tag{2.3.11}$$

of the gyration tensor S_{mn}

$$S_{mn} = \frac{1}{N}\sum_{i=0}^{N} r_m^{(i)} r_n^{(i)}, \tag{2.3.12}$$

where r_m^i is the m^{th} component of the position vector of the i^{th} vertex [17]. The roughness R of the surface can be measured by comparing the angles between adjacent triangles on the shell surface

$$R = 1 - \frac{\Sigma_{<\alpha\beta>} \frac{\hat{n}_\alpha \cdot \hat{n}_\beta}{A_\alpha + A_\beta}}{\Sigma_{<\alpha\beta>} \frac{1}{A_\alpha + A_\beta}}, \tag{2.3.13}$$

where α and β are adjacent triangles. $\hat{n}_i$ and A_i are the unit normal vector and the surface area of the triangle i, respectively. This measure of roughness was developed by Julius Lehmann and Cesar Lopez Pastrana.

To test the hypothesis that mechanical strain sensing could aid in cell shape regulation in bacterial shells [24], I set up a simulation experiment with a spherical shell. Before testing strain dependent growth, I simulated a growth trajectory where I assumed no interaction between the incorporation of material and local properties. I extracted geometric observables of the shell with the roughness and asphericity measures. The constant per area growth reaction rates produced a moderate size increase and conserved the spherical shape, however a high surface roughness emerged as a result of the random incorporation of material (Fig. 2.3.3). I concluded that the growth without surface property interaction was not sufficient in conserving the shape of the bacterial shell in the spring based model simulation.

I next asked whether strain dependent growth could reduce the roughness. The strain of a given edge e is

$$\varepsilon_e = (l_e - l_{e,0})/l_{e,0}, \tag{2.3.14}$$

where l_e is the deformed length of the edge and $l_{e,0}$ is the corresponding rest length. I modeled strain dependent growth by adjusting the growth reaction rates as $\lambda_e = -\lambda_0 + \lambda_1 \varepsilon_e$, where λ_0, λ_1 are constants and $\lambda_0 = \lambda_1 \langle \varepsilon_e \rangle_e$. I reasoned that the inverse relationship between rest length and strain could stabilize the growth through negative feedback. I simulated the growth evolution of the bacterial shell with the spring based model until the shell reached a volume of over eight times its initial volume. Fig. 2.3.3 shows the geometry and the time evolution of extracted roughness and asphericity values. The strain dependent growth conserved the spherical shape, indicating long-term stability.

In the spring based model of the spherical bacterial shell, strain dependent growth performed better on all objectives.

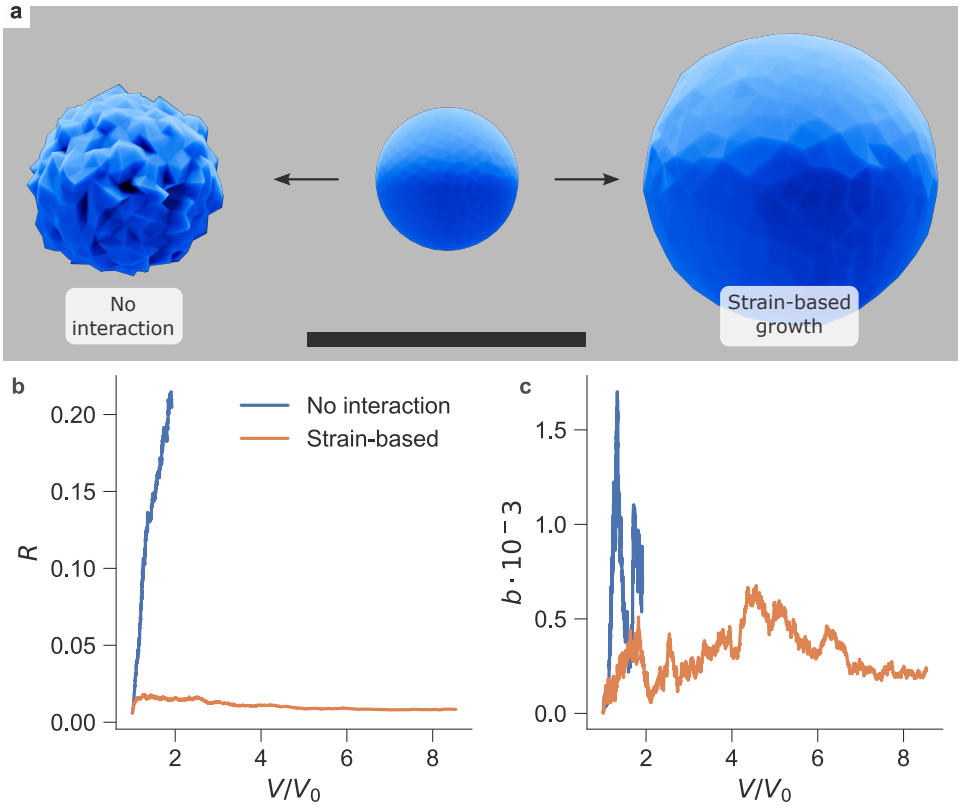

Figure 2.3.3: Spherical shell grown without interation and with strain sensing.
Initial radius 250 nm, E=50 pN/nm^2, t=2 nm, p=0.3 atm. Growth step $\epsilon = 1/5$. Scale bar, 1000 nm. **a**, Initial geometry, geometry of shell grown without interaction and geometry of shell grown with strain sensing. Rendered with [27]. **b**, Roughness measure of the surface. The surface roughness of the shell grown without interaction increases sharply. The roughness of the surface is nearly unchanged for the shell grown with strain-based rates. **c**, Asphericity of the shell. The asphericity is very low for both growth modes throughout the growth process, which indicates a nearly spherical shape.

2.3.3 Growing the rod-shaped bacterial shell

My simulation results of the spherical shell indicate that strain dependent growth reaction rates can facilitate stable growth. I repeated the previous simulation experiment with an initial rod-shaped geometry, anticipating comparable results. Before reporting on the simulation results, I revisit the shape preservation objectives and adjust them for the rod-shaped shell (Tab. 2.3.2).

To extract the length, radius and bending angles from the geometry I used a slicing approach, shown in Fig, 2.3.4.

Goal	Metric
1. Increasing the length of the shell	Length of center line L
2. Conserving the radius of the shell	Mean center line distances r
2. Conserving the bending angle of the shell	Center line piece angles φ
3. Conserving the smoothness of the surface	Roughness R

Table 2.3.2: Objectives in growing the spherical bacterial shell.

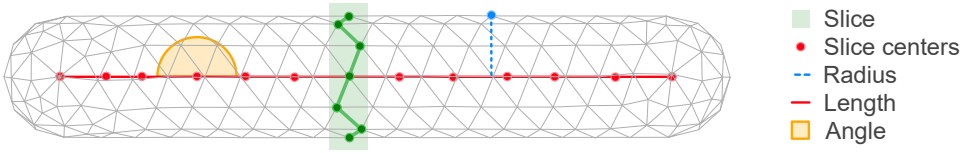

Figure 2.3.4: Computing length, radius and bending angle of the rod-shaped shell.
The shell is divided into slices (green) along its length. At initialization, each vertex is assigned a slice. The gravitational centers of the slices (red dots) can be computed at any point in the simulation. Radius, length and bending angles are computed from the line connecting the slice centers.

The results from the simulation experiment with the rod-shaped shell are presented in 2.3.5. I was able to replicate the previously observed roughness reduction in the rod-shaped shell. The resulting geometry experienced a significant increase in radius, however.

In a pressurized cylindrical shell with isotropic material properties, the stress in the circumferential direction is doubled compared to the longitudinal direction. This results in higher growth reaction rates for edges aligned in the circumferential direction, amplifying the radius.

These results indicate that edge directions can be discriminated via strains. In the spring based model, the possible growth directions are reduced to the directions that the edges are already aligned with. I next wondered whether the direction of edges in the triangulated mesh could influence the increase in radius in the spring based model. I compared two meshes with opposing edge alignments but identical geometry. I set up a deterministic growth experiment where I grew the edges with the best longitudinal alignment simultaneously and compared the radius gain between the meshes. Notably, their responses were contradictory: The shell with the underlying longitudinally aligned mesh contracted in radius, while the shell with the underlying circumferentially aligned mesh expanded in radius (Fig. 2.3.6). The spring mesh maps to isotropic material for small strains as long as triangles in the mesh are equilateral. With directional growth, this is no longer the case. Depending on the orientation of the mesh, triangles get deformed in a different way which results in the distinct outcomes observed.

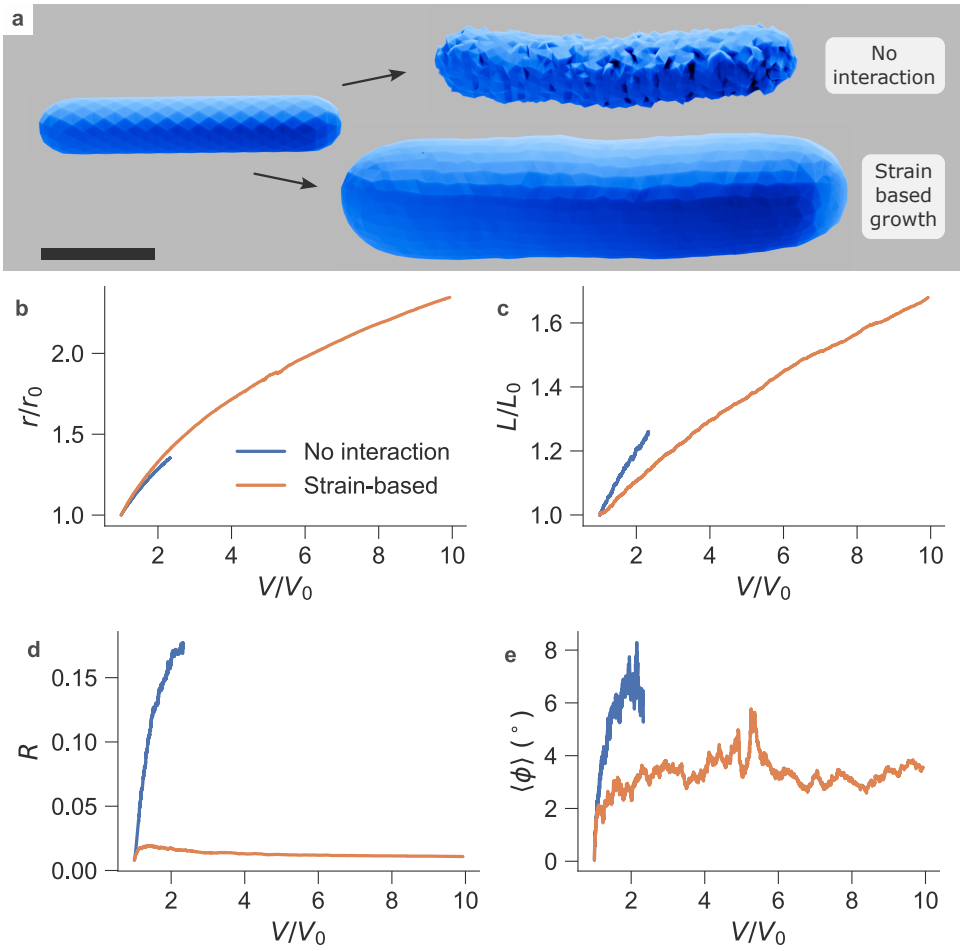

Figure 2.3.5: Growth without interaction vs. with strain sensing in the rod-shaped bacterial shell.
$r_0 = 200\,\text{nm}$, $L_0 = 2000\,\text{nm}$, E=50 pN/nm^2, t=2 nm, p=0.3 atm. Growth step $\epsilon = 1/5$. Scale bar, 1000 nm. **a**, Initial geometry, geometry of the shell grown without interaction and with strain-based rates. . Rendered with [27]. **b**, The radius increase is more pronounced in the shell grown with strain-based rates. **c**, Time evolution of length. **d**, Surface roughness. The surface roughness in the randomly grown shell increases rapidly. The roughness of the shell grown with strain-based rates is low and consistent. **e**, Bending angle. The bending angle of the randomly grown cylinder reaches a value between 6° and 8°, (panel a). The cylinder grown with strain rates has a slight bend of 2° to 4°, however this is not visible by eye (panel a).

2.4 Limitations of the spring based model

The spring based model was useful in exploring the response of the spherical shell to strain-based growth. However, it shows hyperelastic responses

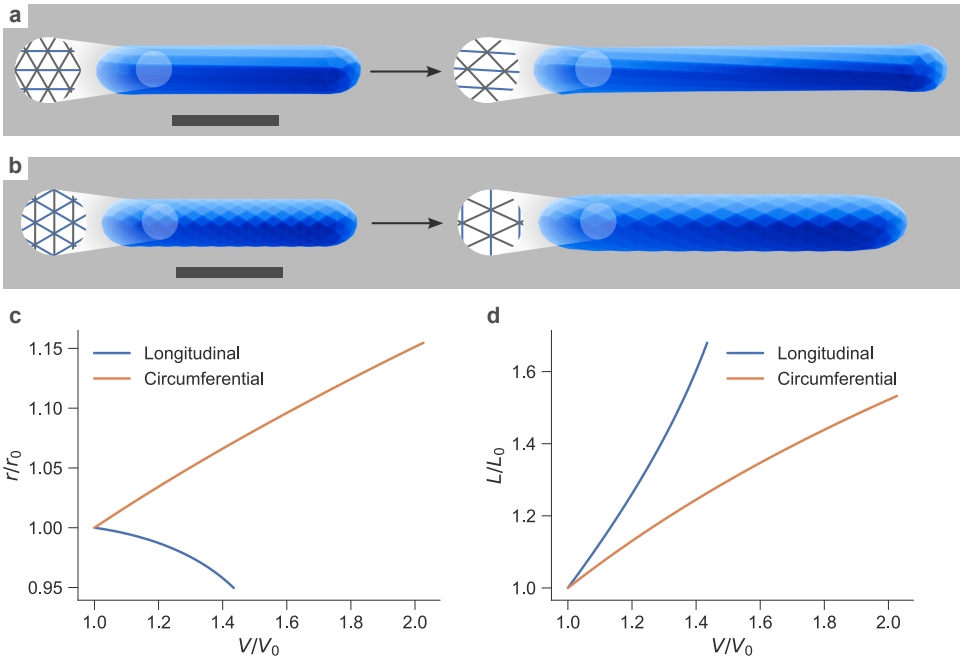

Figure 2.3.6: Deterministic longitudinal growth in longitudinal and circumferential aligned meshes.
a, **b** Initial and grown geometries of shells with longitudinally and circumferentially aligned meshes. L_0=2000 nm, r_0 =200 nm, 300 vertices. E =50 pN/nm^2, t =2 nm, p=0.3 atm. Scale bar, 1000 nm. Rendered with [27]. **b**, Relative change in radius. **c**, Relative change in length.

to large strains. I showed that some properties of the surface do not stay isotropic in the context of growth. Another drawback of the spring model is that topological defects on the surface in the form of vertices with coordination numbers $\neq 6$ can give rise to high stress points on the surface [22].

These limitations could be circumvented with a variety of approaches. As long as initial parameters are chosen with care, the hyperelastic behavior can be avoided. The triangulation algorithm could be adjusted to produce meshes without a preferred direction. Another approach is a different physical model of the bacterial shell that is based on the finite element method, which I present in the next chapter.

Chapter 3

Finite element based model

In this chapter I present the finite element based model of the bacterial shell surface. The idea to implement a physical model of the shell based on the finite element method was initially suggested to me by Paul Nemec. In the following chapters I will introduce the finite element method as a computational technique in general and then present the workflow that I developed for the growth simulations, before finally reporting on my results.

3.1 The finite element method as a computational technique

Allen F. Bower's book 'Applied Mechanics of Solids' [11] has served me as an accessible introduction to the finite element method and helped me revisit many concepts from continuum mechanics. The following section is based on chapter 7.

The finite element method is in general a technique for solving partial differential equations. It can be used to solve problems of solid mechanics, fluid mechanics, heat transfer, and electromagnetics. A typical use case of this analysis is to calculate the stresses, strains and displacements of a solid body in equilibrium under outside forces.

Assumptions:
The focus of application in the context of modelling the bacterial shell surface is on static linear elastic analysis in solid mechanics. The surface thickness is assumed to be small so that the plane stress condition holds. We are solving the system for the equilibrium solution, which imposes that the sum of forces be zero $\Sigma\vec{F} = 0$.

Interpolation:
The undeformed geometry of the shell is defined by a mesh, which divides the surface into a number of elements. The displacement is computed for

the vertex nodes in the mesh. The solution for any point on the surface is computed by linear interpolation between the nearest three vertices.

Strains are computed from the displacement field by differentiation, and once known, can be used to compute stresses on the surface. We compute the potential energy of the mesh as the sum of the strain energy and a boundary force term and minimize it to obtain the displacement vector.

3.1.1 Stress-strain relations

We can write the strain for a given element as

$$\vec{\varepsilon} = [B]\vec{u}_{\text{element}} = \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ 2\varepsilon_{12} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_a}{\partial x_1} & 0 & \frac{\partial N_b}{\partial x_1} & 0 & \frac{\partial N_c}{\partial x_1} & 0 \\ 0 & \frac{\partial N_a}{\partial x_2} & 0 & \frac{\partial N_b}{\partial x_2} & 0 & \frac{\partial N_c}{\partial x_2} \\ \frac{\partial N_a}{\partial x_1} & \frac{\partial N_a}{\partial x_2} & \frac{\partial N_b}{\partial x_1} & \frac{\partial N_b}{\partial x_2} & \frac{\partial N_c}{\partial x_1} & \frac{\partial N_c}{\partial x_2} \end{bmatrix} \begin{bmatrix} u_1^{(a)} \\ u_2^{(a)} \\ u_1^{(b)} \\ u_2^{(b)} \\ u_1^{(c)} \\ u_2^{(c)} \end{bmatrix} \tag{3.1.1}$$

where N_a, N_b and N_c are linear element interpolation functions which are 1 at the corner a, b and c, respectively, and 0 at the other corners. $u_i^{(a)}$ is the i^{th} component of the displacement vector of vertex a. We can conveniently write the strain tensor in Voigt notation as a a vector with three entries, because $\varepsilon_{12} = \varepsilon_{21}$. This allows us to write the strain-deformation relation as a vector equation.

For any linear elastic material stress $\boldsymbol{\sigma}$ and strain $\boldsymbol{\varepsilon}$ are related by generalized Hooke's law for an isotropic solid. Assuming plane stress, we can write the stress-strain relation as

$$\begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{bmatrix} = [D] \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{12} \end{bmatrix} = \frac{E}{(1-\nu)^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{bmatrix} \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{12} \end{bmatrix} \tag{3.1.2}$$

with Young's modulus E and Poisson's ratio ν, where [D] is the material property matrix..

3.1.2 Computing the global strain energy

We can write the strain energy density (energy per unit area) as

$$U = \frac{1}{2}\vec{\varepsilon}^{\mathbf{T}}\vec{\sigma} = \frac{1}{2}\vec{\varepsilon}^{\mathbf{T}}[D]\vec{\varepsilon}. \tag{3.1.3}$$

Using equation 3.1.1 we can write the strain energy of an element as

$$W_{\text{element}} = A_{\text{element}} U_{\text{element}} = \frac{1}{2} \vec{u}^{\mathbf{T}}_{\text{element}} (A_{\text{element}} [B]^{\mathbf{T}} [D][B]) \vec{u}_{\text{element}}, \tag{3.1.4}$$

where A_{element} is the area of the given element and 3.1.1 was used. The stiffness matrix $K_{\text{element}} = (A_{\text{element}}[B]^{\mathbf{T}}[D][B])$ is symmetric because the the material property matrix [D] is also symmetric. We can write the total strain energy as the sum over all the element strain energies:

$$W = \sum_{\text{elements}} W_{\text{element}} = \frac{1}{2} \sum_{\text{elements}} \vec{u}^{\mathbf{T}}_{\text{element}} K_{\text{element}} \vec{u}_{\text{element}} \tag{3.1.5}$$

This equation can be written in matrix form by writing a global displacement vector $\vec{u}$ and the global stiffness matrix K:

$$W = \frac{1}{2} \vec{u}^{\mathbf{T}} [K] \vec{u} \tag{3.1.6}$$

This concludes the computation of strain energy.

3.1.3 Computing the boundary term in the potential energy

This section gives an overview over the procedure to compute the boundary term in the potential energy. The pressure force acts as a face load on the triangular elements. Let us denote the pressure force on a given element as $\vec{f}_p^{(\text{element})} = p\vec{n}_{(\text{element})}$, where p is the pressure and $\vec{n}$ is the normal vector on the given element, pointing outwards. The contribution to the potential energy from the boundary force on the given element is

$$P_{\text{element}} = -\int_A \vec{f}_p^{\text{element}} \cdot \vec{u} \mathrm{d}A, \tag{3.1.7}$$

where $\vec{u}$ is the displacement field. It is assumed that the pressure force is constant over any one element. This allows us to write

$$P_{\text{element}} = -\vec{f}_p^{\text{element}} \cdot \int_A \vec{u} \mathrm{d}A \tag{3.1.8}$$

Computing the integral is trivial because the deformations $\vec{u}$ vary linearly over any single element and can be written as a linear combination of the displacements at the three corners $\vec{u}_a$, $\vec{u}_b$ and $\vec{u}_c$

$$\vec{u} = \vec{u}_a N_a(v, w) + \vec{u}_b N_b(v, w) + \vec{u}_c N_c(v, w), \tag{3.1.9}$$

Where v and w are local 2D coordinates on the element surface. The integral then becomes

$$\int_A \vec{u} \mathrm{d}A = \beta_a \vec{u}_a + \beta_b \vec{u}_b + \beta_c \vec{u}_c, \tag{3.1.10}$$

where $\beta_i = \int_A N_i(v,w)\mathrm{d}A$ are constants, which account for the size of the triangle. The boundary force contribution to the potential energy of the triangular element can be written as

$$P_{\text{element}} = -\beta_a \vec{f_p} \cdot \vec{u}_a - \beta_b \vec{f_p} \cdot \vec{u}_b - \beta_c \vec{f_p} \cdot \vec{u}_c. \tag{3.1.11}$$

We can rewrite this as a vector product of the residual force vector $\vec{r}_{element} = [\beta_a \vec{f_p}, \beta_b \vec{f_p}, \beta_c \vec{f_p}]^{\mathrm{T}}$ and the deformations $\vec{u}_{\text{element}} = [\vec{u}_a, \vec{u}_b, \vec{u}_c]^{\mathrm{T}}$ as

$$P_{\text{element}} = -\vec{r}_{\text{element}} \cdot \vec{u}_{\text{element}} \tag{3.1.12}$$

The total boundary force contribution to the potential energy is the sum of all boundary terms for all the elements

$$P = -\sum_{\text{elements}} \vec{r}_{\text{element}} \cdot \vec{u}_{\text{element}}, \tag{3.1.13}$$

or written in terms of the global displacement vector $\vec{u}$

$$P = -\vec{r} \cdot \vec{u}, \tag{3.1.14}$$

with the global residual force vector $\vec{r}$. This concludes the potential energy computation

$$V = W + P = \frac{1}{2}\vec{u}^{\mathbf{T}}[K]\vec{u} - \vec{r} \cdot \vec{u} \tag{3.1.15}$$

This system of linear equations can be solved for the displacements with gaussian elimination, Cholesky factorization or conjugate gradient algorithms.

I developed a workflow based on the finite element software COMSOL Multiphysics for the computations, which I present in the next chapter.

3.2 Computational methods

This chapter explains the workflow that I developed based on COMSOL Multiphysics for the bacterial shell growth simulations. A major challenge was the correct implementation of local growth, which I solved by introducing strains to grown elements.

3.2.1 Initial Strains in COMSOL

The procedure I developed is based on the concept of initial strains in COMSOL. The COMSOL user guide [28] suggests a workflow involving strains to alter the elastic rest state and size of simulated objects. I summarize the example from the COMSOL user guide before presenting my adapted workflow.

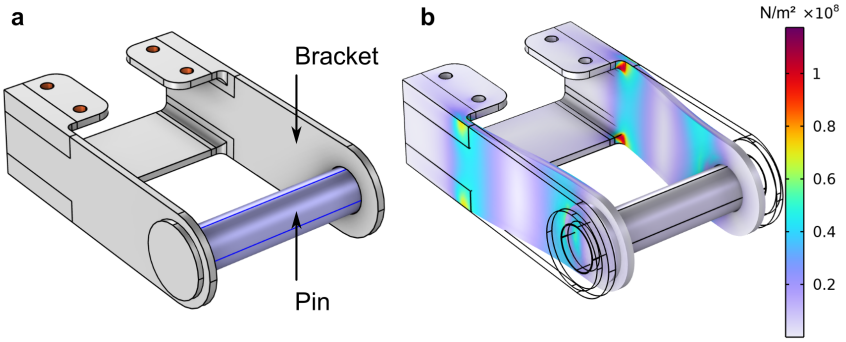

Figure 3.2.1: Bracket, and pin with an initial strain set.
a, Initial, undeformed state. An initial strain $\varepsilon_0 = \frac{L_0 - L}{L_0}$ is set for the pin along its symmetry axis. This changed the elastic rest state of the pin. The initial pin length $L_0 = 215\,\text{mm}$ is different from its current or rest length $L = 214\,\text{mm}$. **b**, Deformation and stresses. The pin has contracted towards its rest length and deformed the bracket. Colors indicate stresses.

The guide presents us with a geometry consisting of two objects, a pin and a bracket to hold the pin in place (Fig. 3.2.1a). The pin fits exactly in the bracket. The reader is presented with a question: What if the pin is slightly shorter than the width of the bracket? This could occur from a manufacturing defect or a thermal deformation, for example. How would the resulting stresses and deformations be computed? The pin cannot be shortened directly, because it is constrained by the width of the bracket. The solution lies in the way the initial configuration is set up. Instead of shortening the pin, the same initial geometry is used, but it is assumed that the pin has been extended to fit into the bracket. This implies a strain

$$\varepsilon_0 = \frac{L_0 - L}{L_0} \tag{3.2.1}$$

in the pin, with L the length of the pin in its undeformed state and L_0 the bracket width. In COMSOL this strain is called the initial strain. I adopted this terminology. Internally, this strain is added to the deformation strain as an offset:

$$\vec{\varepsilon} = [B]\vec{u}_{\text{element}} + \vec{\varepsilon}_{0,\text{element}}, \tag{3.2.2}$$

where $\vec{\varepsilon}_{0,\text{element}}$ is the initial strain vector $[\varepsilon_{11}, \varepsilon_{22}, \varepsilon_{12}]^{\mathbf{T}}$ of the given element. Fig. 3.2.1b shows the simulated results. The pin has contracted and deformed the bracket, inducing stresses. I used a similar approach to simulate the growth of triangular elements on the shell surface.

3.2.2 Local growth with initial strains

Here I present my workflow for introducing local growth in the finite element based model of the bacterial shell. I grew triangular elements via the

initial strain method outlined above. I adapted the previously presented workflow for two-dimensional growth transformations and applied an initial area correction to the pressure force. Let me first present the calculation of strains for each triangle.

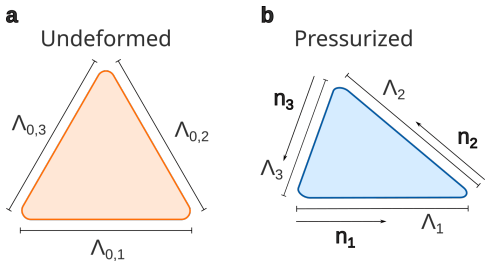

Figure 3.2.2: Triangle in the undeformed and the pressurized configuration.
a, Triangle in the undeformed configuration with undeformed edge lengths $\Lambda_{0,1}$, $\Lambda_{0,2}$ and $\Lambda_{0,3}$. No strains are present. **b**, The triangle in the pressurized configuration with deformed edge lengths Λ_1, Λ_2 and Λ_3. Normalized edge vectors $\vec{n}^{(1)}$, $\vec{n}^{(2)}$ and $\vec{n}^{(3)}$. The strain in this configuration is computed with equations 3.2.3 and 3.2.5.

The strain of a triangle in any deformed configuration was computed analogously with the pin example as

$$\varepsilon(\vec{n}^{(i)}) = \frac{\Lambda_i - \Lambda_{i,0}}{\Lambda_i} \tag{3.2.3}$$

where $\Lambda_{0,i}$ are the edge lengths of the triangle in the undeformed configuration, and Λ_i, $\vec{n}^{(i)}$ are the edge lengths and edge direction unit normal vectors, respectively. Using the definition of the 2D strain matrix, the strain in the edge direction can be written as

$$\varepsilon_{\text{direction}}(\vec{n}^{(i)}) = \varepsilon_{kl} n_k^{(i)} n_l^{(i)}, \tag{3.2.4}$$

which yields the system of linear equations

$$\begin{bmatrix} \varepsilon_{\text{direction}}(\vec{n}^{(1)}) \\ \varepsilon_{\text{direction}}(\vec{n}^{(2)}) \\ \varepsilon_{\text{direction}}(\vec{n}^{(3)}) \end{bmatrix} = \begin{bmatrix} n_1^{(1)} n_1^{(1)} & n_1^{(1)} n_2^{(1)} & n_2^{(1)} n_2^{(1)} \\ n_1^{(2)} n_1^{(2)} & n_1^{(2)} n_2^{(2)} & n_2^{(2)} n_2^{(2)} \\ n_1^{(3)} n_1^{(3)} & n_1^{(3)} n_2^{(3)} & n_2^{(3)} n_2^{(3)} \end{bmatrix} \cdot \begin{bmatrix} \varepsilon_{11} \\ 2\varepsilon_{12} \\ \varepsilon_{22} \end{bmatrix}. \tag{3.2.5}$$

The strain tensor components ε_{11}, ε_{12} and ε_{22} were computed by solving the linear equations. Applied to the bacterial shell, this allowed me to compute the two-dimensional strain for any triangle in any configuration. In my workflow, three configurations came into play. The pressurized configuration corresponds to the pressurized geometry of the shell. The undeformed

configuration contains the rest state of each triangle. The initial configuration has no physical equivalent. It was a convenient intermediate step to be used within COMSOL.

The three configurations are identical at the start of a simulation run. Once the turgor pressure is applied, triangles in the pressurized configuration expand. When a triangle is transformed by growth, the rest state in the undeformed configuration is modified. The geometry of the initial configuration is not altered. Instead, the initial strain is applied to the grown triangle. The effect on the pressurized configuration is computed from the initial configuration. It should be noted that the pressurized configuration computed in the previous simulation step could be used as the initial configuration in COMSOL, with its strains applied as initial strains. The intermediate initial configuration was convenient because it minimizes the number of elements with an initial strain, which I found to have a significant impact of the computational time required by the COMSOL solver. I used the pressurized configuration computed in the previous simulation step as initial guess for the solution, however.

The undeformed triangle in the finite element based model is analogous to the rest lengths of springs in the spring based model. As triangles are two-dimensional objects, the manner in which growth modifies a triangle is not immediately clear. Two distinct growth modes are possible in general. Self-similar growth and directed growth.

Let me first introduce self-similar growth. The undeformed area $A_{i,0}$ of a triangle can be computed via Heron's formula considering the undeformed edge lengths $\Lambda_{i,0}$ as

$$A_{i,0} = \sqrt{s \prod_{j=1}^{3} (s - \Lambda_{j,0})}, s = \frac{1}{2} \sum_{i=1}^{3} \Lambda_{i,0}. \tag{3.2.6}$$

To grow the triangle area isotropically to an area of $A_{i,0} + \delta A$, each undeformed edge length should be multiplied by the same factor. This conserves the shape of the triangle. Let $A_{i,0} \to A_{i,0} + \delta A$ and $\Lambda_{i,0} \to g\Lambda_{i,0}$. Plugging both in equation 3.2.6 yields

$$A_0 + \delta A = A_0 g^2. \tag{3.2.7}$$

Solving for g gives $g = \sqrt{1 + \delta A / A_0}$. I grew triangles in an isotropic manner as

$$\Lambda_{i,0} \to \Lambda_{i,0} \sqrt{1 + \delta A / A_{i,0}}. \tag{3.2.8}$$

for an area increase of δA.

In order to achieve growth in a specific direction, I followed the scheme presented in Fig. 3.2.3.

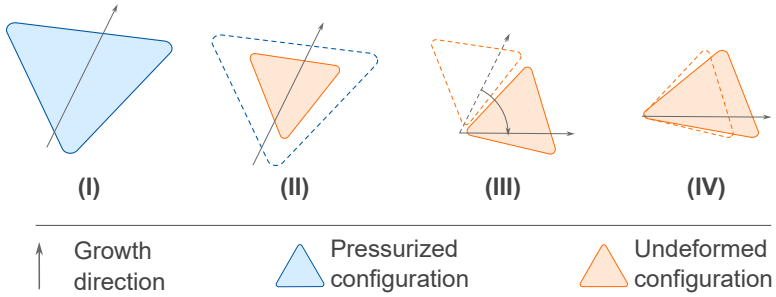

Figure 3.2.3: Directed growth of a triangle in a given direction. Procedure for growing a triangular triangle in a particular direction.

The preferred direction for growth is given ad-hoc only in the pressurized configuration. For directed growth, I used the principal curvature and principal strain directions as growth directions, which are 3D vectors.

In order to work in the 2D coordinate system of the triangle, I projected the vector onto the triangle surface corresponding to the pressurized configuration (Fig. 3.2.3.I). The shape of the triangle in the pressurized configuration is in general different from the undeformed configuration. To obtain the correct transformation, I translated the growth direction into the undeformed configuration (Fig. 3.2.3.II) by considering the alignments of edges with the growth direction. In the next step, I rotated the local coordinate system to align the x-axis with the desired growth direction (Fig. 3.2.3.III). The new shape of the triangle was computed by transforming x-coordinates as $x \to (1 + \frac{\delta A}{A_{i,0}}) \cdot x$ (Fig. 3.2.3.IV). This allowed me to extract the new shape of the triangle, which is defined by its three edge lengths. I saved the new edge lengths in the undeformed configuration and computed the initial strain with equations 3.2.3 and 3.2.5. The edge lengths $\Lambda_{i,0}$ fully define the triangle in the undeformed configuration, because strains are computed with Eq. 3.2.3, where only the directions in the pressurized configuration $\vec{n}^{(i)}$ enter (See Fig. 3.2.2), but not directions in the undeformed configuration.

3.2.3 Initial area correction

To test my implementation of the presented workflow, I designed a simple simulation experiment. I grew a spherical shell with the initial strains work-

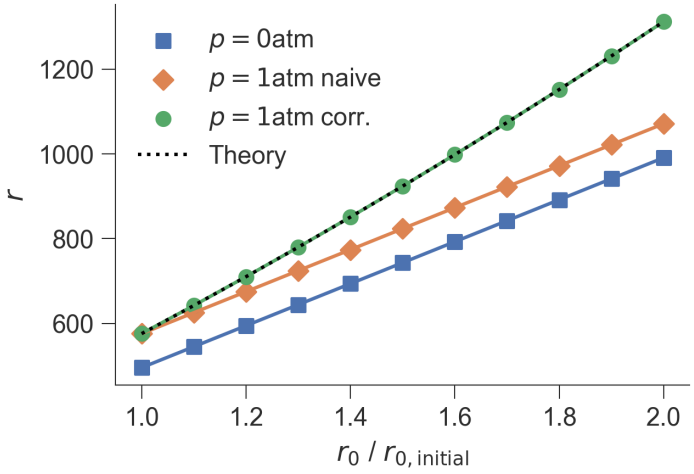

Figure 3.2.4: Initial area correction for a pressurized spherical shell. The radius r of the shell increased as expected for zero pressure. Doubling the undeformed edge lengths of all triangles, a doubling in radius was reached (from $r_{0,\text{initial}}$ =500 nm to r_0 =1000 nm). The naive approach at p =1 atm yielded a pressurized radius that is smaller than the expected radius from linear elasticity theory (eq. 2.3.4). I obtained the correct deformations after introducing the inital area correction (eq. 3.2.10).

flow (Fig. 3.2.4) and observed the increase of the radius. In the absence of pressure, the measured radius corresponded exactly to the surface area increase that the shell was subjected to. Once a pressure was applied, the measured radius contradicted the expected values from linear elasticity theory. I reexamined the pressure force computation in COMSOL in order to rectify this. It is computed as

$$\vec{f}_p^{(i)} = pA_i\vec{n}_i, \tag{3.2.9}$$

where $\vec{f_p}$ is the pressure force on a the triangle i, A_i is the triangle size in the initial configuration for COMSOL and $\vec{n}_i$ is the unit normal vector on the triangle. COMSOL computes the pressure force on a given triangle with the area in the initial configuration, which coincides with the undeformed configuration in the usual case, but not after applying the initial strains workflow to a triangle. The physically correct computation would involve the triangle area in the undeformed configuration instead. I replaced the pressure p on a given triangle i with a corrected pressure p_i^* as

$$p \to p_i^* = \frac{A_i}{A_{0,i}}p. \tag{3.2.10}$$

Using this corrected pressure, the deformations computed by COMSOL matched the predictions by linear elasticity. The full initial strains workflow can be summarized in three steps:

(I) A selected triangle is grown (Eq. 3.2.8, Fig 3.2.3).
(II) The initial strains for COMSOL are computed (Eq. 3.2.3, Eq. 3.2.5).
(III) The corrected pressure forces are computed (Eq, 3.2.10).
(IV) COMSOL solves the system.

I used this workflow in all bacterial shell simulations with the finite element based model.

3.2.4 Simulation process flow

I present the revised process flow for the finite element based model of the bacterial shell. I adapted the workflow presented in section 2.2.3 to the finite element based model (Fig. 3.2.5) by substituting edge-based properties with triangle based properties. The minimization routine was replaced with a wrapper class for COMSOL.

3.2.5 Technologies

The simulation interacts with COMSOL through a custom wrapper class. The wrapper class interfaces with COMSOL Client, which runs in the Java virtual machine, using the MPh library [30]. The Client sends instructions to the COMSOL Server, which solves the model for deformations. See Fig. A.0.1 for a full description of the involved classes.

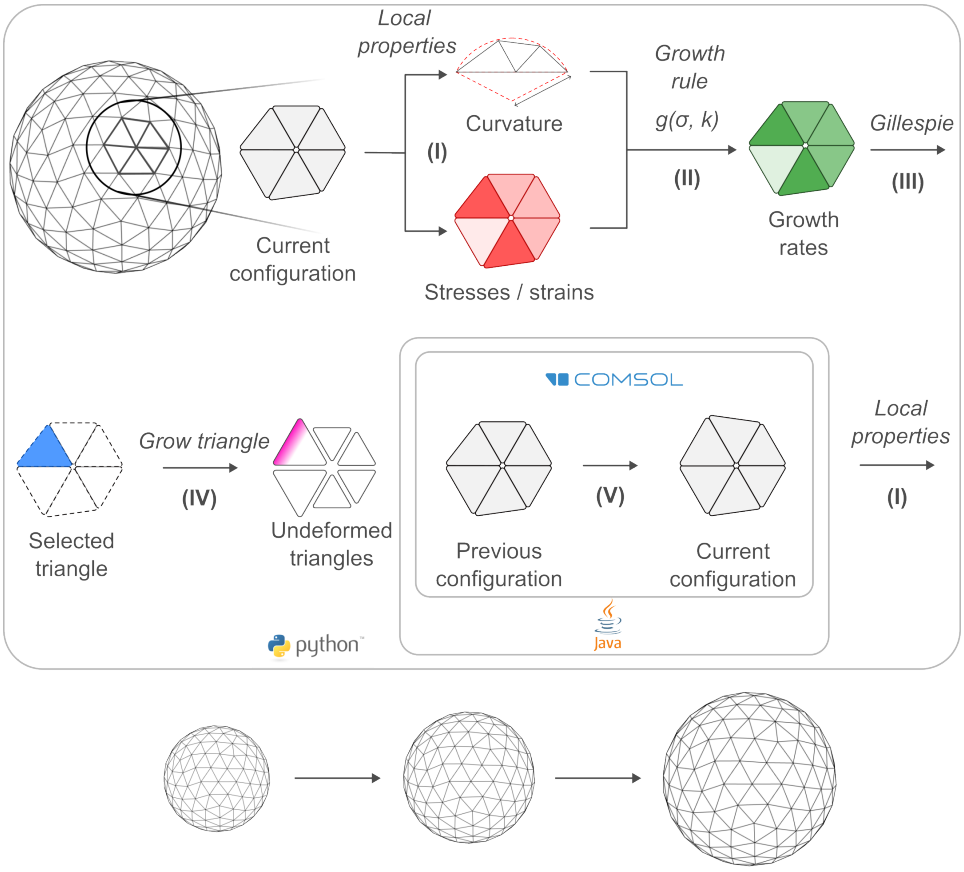

Figure 3.2.5: Simulation process loop for the finite element based model.
The growth simulation consists of a loop that consists of 5 steps.

- **(I)** Compute curvature and stresses of the current configuration.
- **(II)** The growth rule $g(\sigma, k)$ computes growth reaction rates for triangles in the surface.
- **(III)** One triangle is selected with the Gillespie algorithm.
- **(IV)** The the triangle is grown in the undeformed configuration.
- **(V)** The new current configuration is computed from the rest lengths and physical parameters. The previous pressurized coordinates are used as initial guess for the minimum.
  → Go to **(I)**.

3.3 Results and discussion

I repeated my simulation experiments presented in chapter 2.3, modeling the response of the bacterial shell with the finite element based model. This includes a characterisation of the pressurization behavior, random and strain-based growth in the spherical and rod-shaped shell, and deterministic elongation of the rod-shaped shell. The finite element based model allowed me

to conduct further experiments in the rod-shaped shell, testing a variety of strain- and curvature-based growth laws. I also tested growth laws for their potential to reverse bending in rod-shaped bacteria. A significant downside of the finite element based model is the computational time required by COMSOL.

3.3.1 Spherical bacterial shell under turgor pressure

The pressurization behavior of the spherical bacterial shell in the finite element based model (Fig. 3.3.1) is consistent with linear elasticity theory (Eq. 2.3.4).

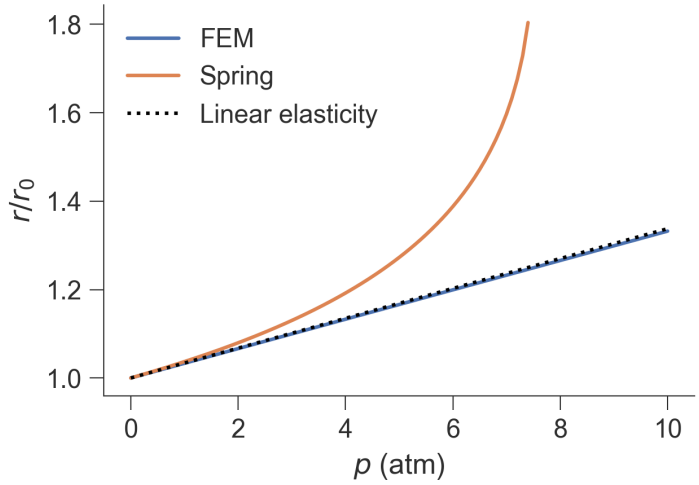

Figure 3.3.1: Pressurization of the spherical shell in the finite element based model.
The radius after pressurization (blue) closely follows the expected result from linear elasticity theory (eq. 2.3.4). The pressurized radius in the spring based model is also shown for comparison (orange).

3.3.2 Strain-based growth in the spherical bacterial shell

I extended the objectives for the spherical shell by a measure of mechanical stability. High surface stresses can indicate a risk for material failure. I measured the mechanical stability of the shell via the variability of hydrostatic surface stresses, which is correlated with peak stresses, but not affected by stochastic noise. The initial geometry of the shell was a sphere with a radius of 200 nm in the following simulation experiment. A Young's modulus of 50 pN nm^{-2} and thickness of 2 nm was simulated. The pressure was set to 0.3 atm to allow a comparison with the simulations in the spring model. To test the theory that strain-based growth can act as a sensory cue for robust

Goal	Observable
1. Increasing the size of the shell.	Volume
2. Conserving the spherical shape of the shell.	Asphericity
3. Conserving the smoothness of the surface.	Nearest-neighbor angles
4. Conserving the mechanical stability.	Standard deviation of hydrostatic stress

shape regulation in bacterial shells, I repeated the simulation experiment outlined in section 2.3.2 with the finite element based model, results presented in Fig. 3.3.2. Notably, no significant change in shape was observed throughout the simulated growth process, which contradicts results from the same experiment performed with the spring based model.

Material was not distributed homogeneously on the surface, which results in increasingly inhomogeneous surface stresses as indicated by the rising standard deviation of stresses. The inhomogeneity of stresses gives rise to surface roughness in the spring based model but not in the finite element based model. This difference in surface roughness could be due to the following reasons: 1) The mechanism used (Eq. 3.2.8), introduces less shear stresses compared to the growth mechanism in the spring based model, where growth is always directional. This leads to a more homogeneous distribution of forces on the surface in the finite element based model. 2) The solving algorithm by COMSOL favors smooth surfaces. 3) The hyperlinear behaviour for large strains increases out-of-plane deformations in the spring based model.

The increasing inhomogeneity of stresses in the finite elements based model of the shell indicate that the mechanical stability of the bacterial shell would be compromised in the long term. I anticipated that strain-based growth could reverse this trend and produce homogeneously distributed surface stresses. I computed the hydrostatic strain of triangular elements as

$$\varepsilon_{\mathrm{h}}^{(e)} = \frac{\varepsilon_{11}^{(e)} + \varepsilon_{22}^{(e)}}{2} \tag{3.3.1}$$

and growth reaction rates as

$$\lambda^{(e)} = -\lambda_0 + \lambda_1 \varepsilon_h^{(e)}, \tag{3.3.2}$$

where $\lambda_0 = \lambda_1 \langle \varepsilon_h^{(e)} \rangle_e |_{t=0} = const.$ and $\lambda_1 = 1$. The constants λ_j are determined after pressurization and held constant throughout the simulation. Fig. 3.3.2 shows the geometrical and mechanical properties of the grown bacterial shell. Geometrical properties are comparable between growth without interaction and growth with strain-sensing. However, the variability of surface stresses is much lower for strain-based growth, which results in lower peak stresses.

To understand the mechanism by which the strain-based growth operates, let us consider the strains in elements as they and their neighbors

increase in size. Initially, all elements are of roughly equal size. The elements experience a positive strain $\varepsilon_{(\text{not grown})} > 0$ induced by the turgor pressure.

A triangle has grown in a random spot. Suddenly, there is a mismatch between the size of the grown triangle and its neighbors. The effect of the turgor pressure on the grown triangle is reduced because of the increased rest lengths $\Lambda_{(i,0)}$ (See Eq. 3.2.3). The mismatch in size between the grown triangle and the ungrown triangle neighboring it induces a tensile strain in the larger triangle and a compressive strain in the smaller triangle. These strains have a direct effect on the growth reaction rates. The growth reaction of the grown triangle is now effectively disabled as the strain drops below the threshold

$$\lambda_1 \varepsilon_{(\text{grown})} < \lambda_0, \tag{3.3.3}$$

resulting in negative growth reaction rates

$$\lambda_{(\text{grown})} < 0 \rightarrow 0, \tag{3.3.4}$$

which are set to 0. The increased strain in the ungrown neighboring elements

$$\varepsilon_{(\text{ungrown , not neighbor})} < \varepsilon_{(\text{ungrown, neighbor})} \tag{3.3.5}$$

results in faster growth reactions, so

$$0 < \lambda_{(\text{ungrown , not neighbor})} < \lambda_{(\text{ungrown, neighbor})}. \tag{3.3.6}$$

The increase in growth rates around a grown triangle means that the first triangle represents a nucleation spot which promotes adjacent growth and leads to an expanding area of grown triangles.

As more triangles grow and grown triangles have more neighbors of equal size, compressive forces are reduced and strains are somewhat equalized in the grown triangles.

Extreme increased and decreased surface stresses are a consequence of mismatches between element sizes. The variability of surface stresses is therefore also a measure of the variability of element sizes. The time evolution of surface stresses shows unexpected periodic behavior for strain-based growth (Fig. 3.3.2c). Five peaks can be observed, which correspond to the five growth steps of the full shell. What is the source of this periodic pattern in the strain-based mechanism?

Fig. 3.3.2d shows a representative selection of triangle sizes at times t_0, t_2 and t_2. Triangle sizes are homogeneous at time t_0. As more and more triangles grow, the variability in size increases. This explains the increase in the variability of stresses visible in Fig. 3.3.2c from time t_0 to time t_1. But why does the variation of surface stresses reduce between times t_1 and t_2?

The strain-based growth mechanism favors elements which have not yet grown and and renders already grown elements inactive. Time t_1 is reached

when approximately half of the elements have grown and half of the elements have not grown. Many elements are mismatched in size compared to their neighbors, which results a high variability in stresses. However, it is still unlikely that an element which has grown once will grow again. Consequently, more and more triangles grow for the first time. The majority of triangles has now grown once.

Time t_2 marks a point of maximum homogeneity on the surface, which reduces surface stresses. The cycle repeats as triangles grow for the second time. The cyclical behavior therefore arises from the discrete increase in elements' sizes.

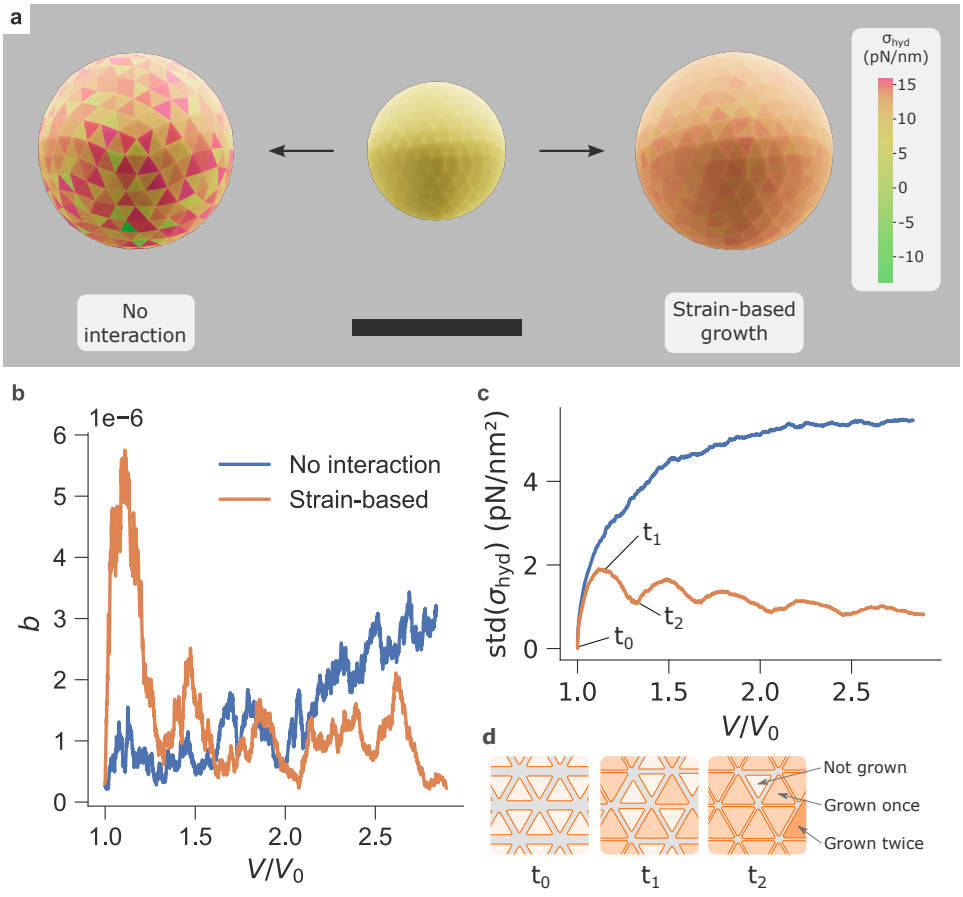

Figure 3.3.2: Spherical shell grown without interaction vs. with strain sensing.
a, Geometry and hydrostatic stresses before and after growth. Stresses are increased for both growth modes, but peak stress is much higher in the shell grown without interaction. **b**, Asphericity is very low for both growth modes, but better and more stable with strain-based growth. **c**, Standard deviation of hydrostatic stresses. Key time points indicated with t_0, t_1, t_2. **d**, Representative triangle sizes in the undeformed configuration. At t_0, all triangles are equally sized. At t_1, half triangles have grown. At t_2, most triangles have grown once.

3.3.3 Strain-based growth in the rod-shaped bacterial shell

I extended the target metrics in growing the rod-shaped bacterial shell by the standard deviation of hydrostatic stress to account for mechanical stability. Length of center line and mean center line distances are computed with the scheme detailed in Fig. 2.3.4. The roughness of the surface is computed with nearest-neighbor angles (Eq. 2.3.13). The geometry of the shell was a spherocylinder with a length of 2000 nm and a radius of 200 nm in all following simulation experiments. A Young's modulus of $50\,\mathrm{pN\,nm^{-2}}$ and

Goal	Metric
1. Increasing the length of the shell.	Length of center line.
2. Conserving the radius of the shell.	Mean center line distances.
3. Conserving the smoothness of the surface.	Nearest-neighbor angles.
4. Conserving the mechanical stability.	std(hydrostatic stress).

thickness of 2 nm was simulated. The pressure was set to 1 atm to explore higher pressures. These parameters are characteristic of Gram-negative bacteria (see 1.3.1).

I next studied the response of the rod-shaped shell to growth without local surface property interaction. This growth mode increased the shell radius and surface stresses. The strain-based growth law applied on the rod-shaped shell mediated uniform incorporation of material as seen in the spherical shell, which resulted in lower peak surface stresses. However, the radius of the shell was not conserved. This matches the response of the rod-shaped shell in the spring model in Fig.2.3.5.

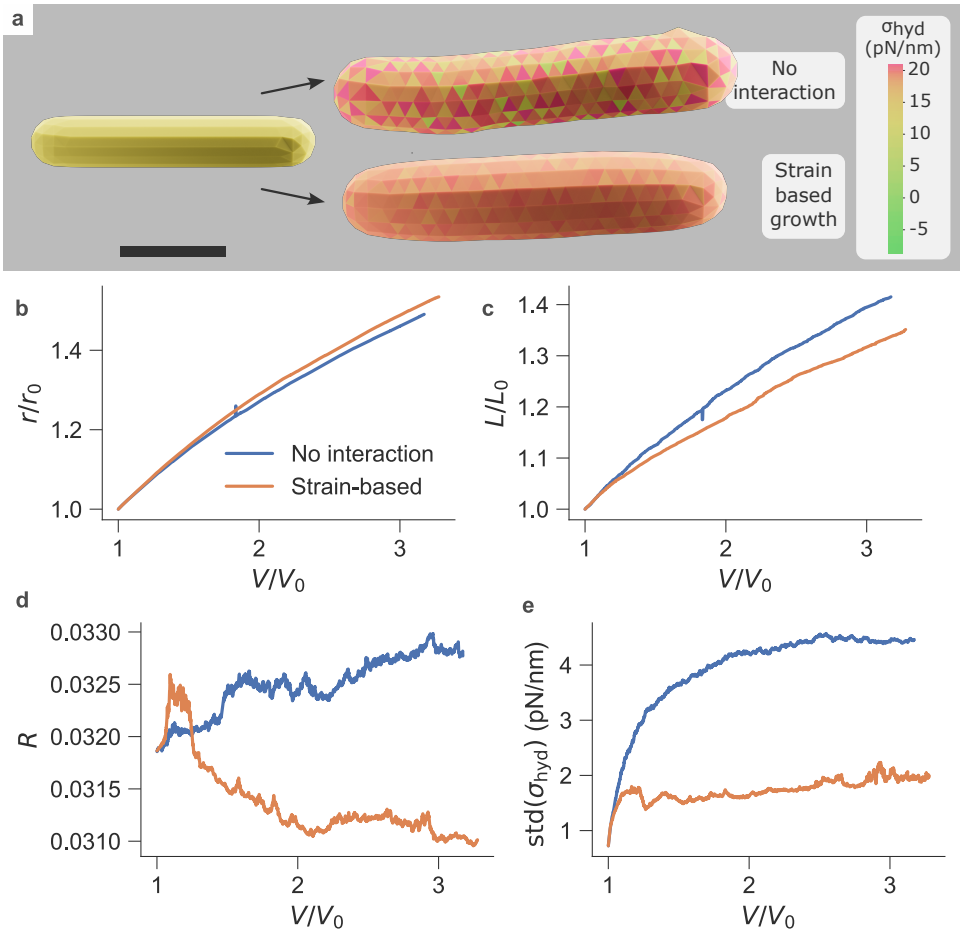

Figure 3.3.3: Rod-shaped shell grown without interation and with strain sensing.
a, Geometry and hydrostatic stresses before and after growth. Stresses are increased for both growth modes, but peak stress is much higher in the shell grown without interaction. **b**, **c**, Simulated radius and length increases were very similar for both growth modes. **d**, Surface roughness is lowered for the strain based growth as the edge between cap and shaft is smoothed out. **e**, The variability of hydrostatic stresses is stable for strain-based growth but increases for growth without interaction.

3.3.4 Directed growth in the rod-shaped bacterial shell

I next asked whether the rod-shaped shell would show any dependence on the underlying mesh alignment, as was observed in the spring based model. I repeated the simulation experiment from Fig. 2.3.6, comparing the growth response of two rod-shaped bacterial shells with identical geometry but different mesh alignments. Triangles in the shaft of the shell were elongated along the longitudinal direction with the procedure described in Fig. 3.2.3.

I observed an increase in radius for both mesh alignment configurations. (Fig. 3.3.4). This stands in contrast to the response of the rod-shaped shell in the spring-based model, where the responses were qualitatively distinct. These results indicate that the response of the bacterial shell to growth in the finite element based model is isotropic and largely independent of the structure of the underlying triangulation mesh.

The observed increase in radius serves as lower bound for radius conservation with purely increasing growth laws in the finite element based model.

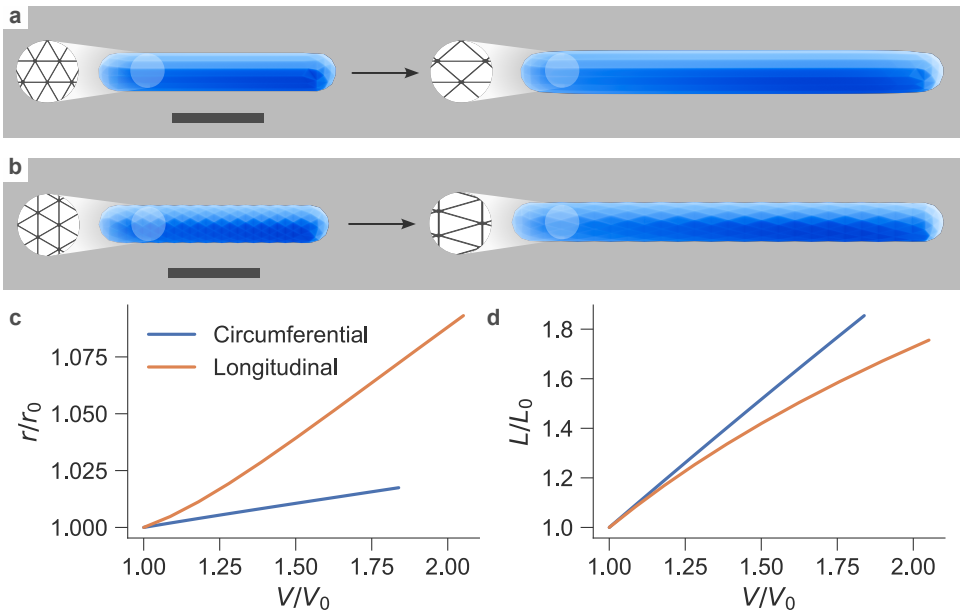

Figure 3.3.4: Deterministic growth of the rod-shaped bacterial shell in the longitudinally aligned and the circumferentially aligned mesh
a, **b**, Geometry change for longitudinal growth . **c**, **d**, Radius increase and length change for longitudinal growth.

3.3.5 Mechanical and geometric cues for directional and positional sensing

The anisotropicity of the cell wall and the microscopic details of the cell wall synthesis machinery [16], [15], [12] are not part of my model. Radius conservation in the growth of bacterial shells might be a consequence of the synthesis machinery or the structure of the cell wall. However this model, radius conservation needs to be mediated through local properties.

The trials in deterministic growth indicate that elongation of the rod-shaped bacterial shell is achievable in the finite element based model with minimal radius increase, provided that local properties can discriminate between

	Growth reaction rates	Growth direction
I.	$\lambda_0 + \lambda_1 \varepsilon_{\min} - \lambda_2 \varepsilon_{\max}$	$\varepsilon_{\min}$
II.	$\lambda_0 + \lambda_1 K_{\min}$	$\varepsilon_{\min}$
III.	$\lambda_0 + \lambda_1 K_{\min}$	$K_{\min}$

Table 3.3.1: Strains vs. curvatures for directional and positional sensing Parameters $\lambda_0 = 1$, $\lambda_1, \lambda_2 = 5 \cdot 10^{-5}$. With minimum and maximum strain $\varepsilon_{\min}$, $\varepsilon_{\max}$ and minimum curvature $K_{\min}$. Parameters were selected to achieve constant growth rates in the shaft and negative (set to zero) growth rates in the caps.

a) the longitudinal and the circumferential direction and
b) the shaft and the caps of the rod.

Both of these are intimately connected, because the spherical symmetry of the caps renders any distinction between longitudinal and circumferential direction meaningless.

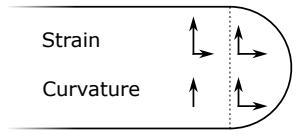

Figure 3.3.5: Principal strains and curvatures in caps and shaft. Strains and curvatures are symmetric in the caps and have no preferred direction. The minimum and maximum strain directions in the shaft point in the longitudinal and circumferential direction, respectively. The maximum curvature is constant throughout the cylinder. The minimum curvature in the shaft is 0.

Fig. 3.3.5 illustrates the principal strains and curvatures throughout the cylinder. The symmetric nature of the strain and curvature in the hemispherical caps of the rod can be exploited to suppress growth. The asymmetry in the shaft allows a distinction between the longitudinal and circumferential direction. In principle, both strain and curvature would be sufficient for direction and position sensing. I tested combinations of curvature-based and strain-based growth and evaluated them on their ability to conserve the radius of the shell. Fig. 3.3.6 shows the time evolution of length and radius. Direction sensing via strains (Tab. 3.3.1.I, II) performed worse than randomly incorporating material with no interaction. Using the curvature to sense the direction (Tab. 3.3.1.III) performed slightly better than randomly growing the sphere. Using local curvatures to differentiate between shaft and caps performed the better than using strains.

	Growth reaction rates	Growth direction
IV.	$-\lambda_0 + \lambda_1 K_{\min} + \lambda_2 \varepsilon_{\text{hyd}}$	$\varepsilon_{\min}$
V.	$-\lambda_0 + \lambda_1 K_{\min} + \lambda_2 \varepsilon_{\min}$	$\varepsilon_{\min}$
VI.	$-\lambda_0 + \lambda_1 K_{\min} + \lambda_2 \varepsilon_{\text{hyd}}$	$K_{\min}$
VII.	$-\lambda_0 + \lambda_1 K_{\min} + \lambda_2 \varepsilon_{\min}$	$K_{\min}$

Table 3.3.2: Combinations of strain- and curvature-based growth. Parameters $\lambda_0 = \lambda_2 \langle \varepsilon_e \rangle_e - 1, \lambda_1 = 5 \cdot 10^{-5}, \lambda_2 = 5$. With minimum, maximum and hydrostatic strain $\varepsilon_{\min}$, $\varepsilon_{\max}$, ε_{hyd} and minimum curvature $K_{\min}$. Parameters were selected to achieve negative (set to zero) rates in the caps and constant rates along the cylinder. The role of λ_0 is to suppress growth in already grown triangles.

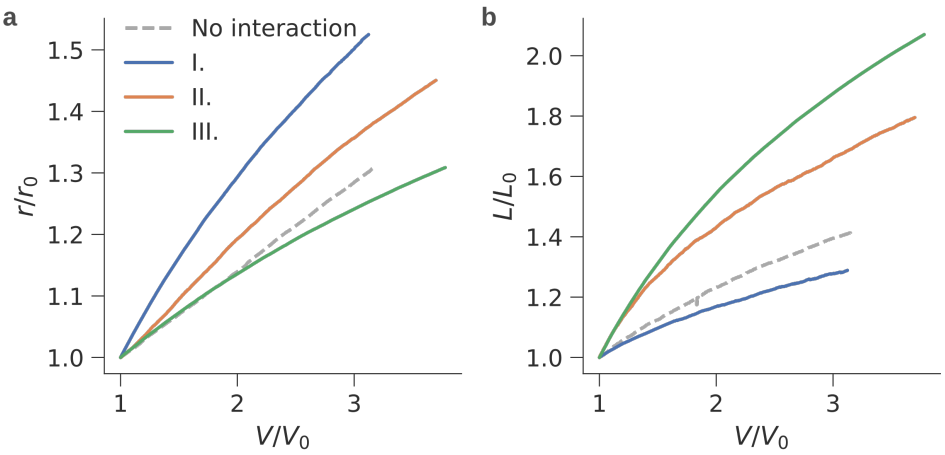

Figure 3.3.6: Strains vs. curvatures for directional and positional sensing **a**, **b**, Relative length and radius change with the growth setups from Tab. 3.3.1. The pure geometrical growth based on curvature performed the best with a radius gain of only 14% per volume doubling.

I next wondered what a combination of directed growth and strain-based growth could look like. I anticipated an improvement in both geometric and mechanic properties compared to growth without interaction.

I hypothesised that growth in the longitudinal direction would primarily effect the strains in the longitudinal direction $\varepsilon_{\min}$. This would leave $\varepsilon_{\max}$ in the circumferential direction largely unchanged. I reasoned that $\varepsilon_{\min}$ would provide a clearer picture of the mechanical state of the shell compared to the full hydrostatic strain ε_{hyd}.

The combination of $\varepsilon_{\min}$ vs. $K_{\min}$ for directionality and $\varepsilon_{\min}$ vs. ε_{hyd} to reduce stresses resulted in the four growth laws presented in Tab. 3.3.2.

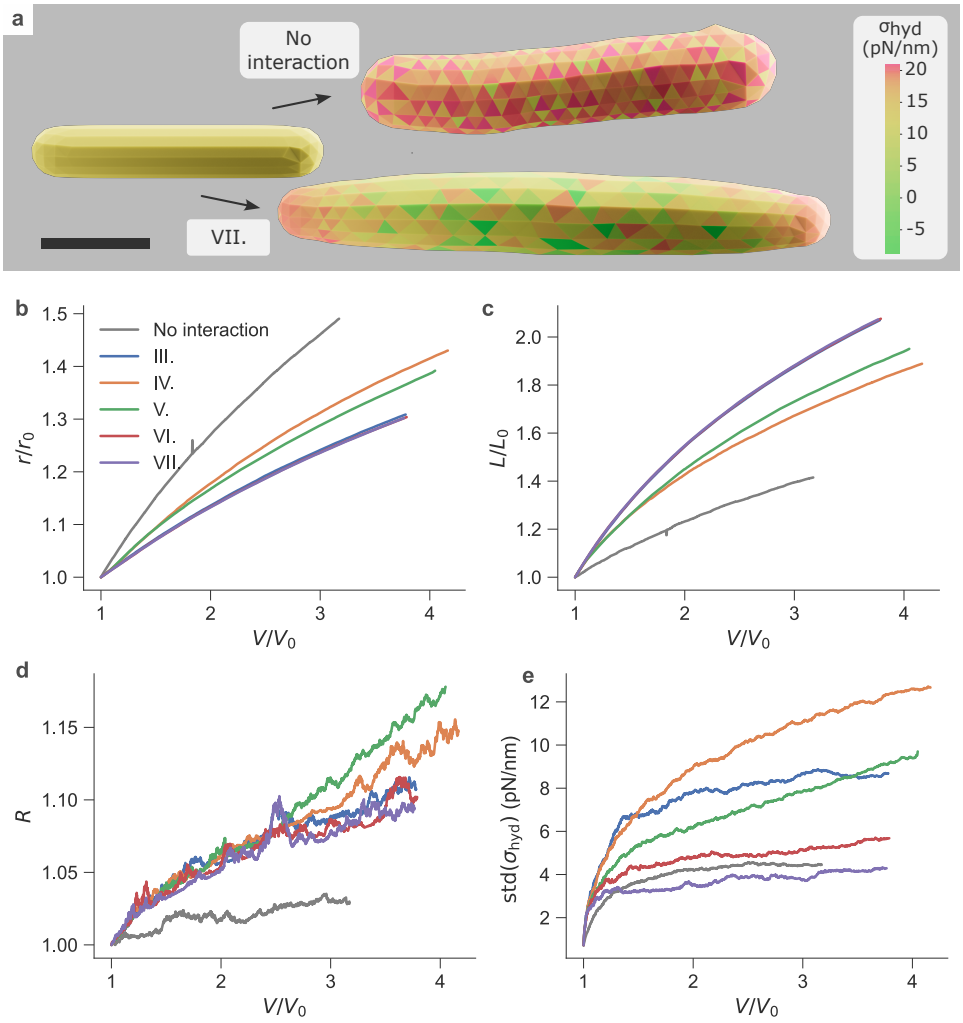

Figure 3.3.7: Combinations of strain- and curvature-based growth. **a**, Resulting geometry and surface stresses from VII compared to no interaction. **b**, **c**, Almost identical geometries for VI, VII. **d**, Lowest roughness value for growth without interaction because in this case the caps were grown. **e**, Surface stresses are lowest for VII, slightly better than growth without interaction.

Fig. 3.3.7 compares the performance of these growth laws. The radius was better conserved with local minimum curvature to sense the longitudinal direction (Tab. 3.3.2 VI, VII) compared to direction sensing via minimum strain (Tab. 3.3.2 IV, V). All four growth laws performed better than randomly incorporating material without interaction.

I found that growth law VII, where curvature was used for directional sensing and minimum strain to reduce surface stresses, performed the best overall and reduced surface stresses the best.

None of the growth laws were able to conserve the radius however. I

reasoned that the radius increase emerged due to the elongation direction not always being aligned with the longitudinal direction of the cylinder, so that some proportion of growth would be applied in the circumferential direction.

I next wondered whether a second growth reaction could complement the existing longitudinal growth reaction so that the radius could be conserved. I hypothesized that a growth reaction which would reduce material along the circumferential direction could aid in achieving this. I used the symmetry of curvatures to reduce the radius only in the shaft of the rod (akin to Tab. 3.3.1). I set up the growth rates as

$$\lambda = \lambda_0 + \lambda_1 K_{\min} - \lambda_2 \varepsilon_{\max} \tag{3.3.7}$$

with $\lambda_0 = \lambda_2 \langle \varepsilon_{e,\max} \rangle_e - 0.04, \lambda_1 = 10^{-5}, \lambda_2 = 1$, growing in the direction of maximum curvature, which points in the circumferential direction. Each iteration results in a reduction of material of $\epsilon = 0.2$. Parameters were selected to suppress growth in the caps in the case of λ_1 and to disable growth in already grown triangles in the case of λ_0 and λ_2. $\varepsilon_{\max}$ is the strain in the circumferential direction, and is the equivalent to $\varepsilon_{\min}$ for growth law VII. The sign is reversed from VII because the reduction of material increases strains instead of reducing them. Already grown triangles with higher maximum strains are suppressed in reducing growth along the circumferential direction.

To test the ability of this mechanism to work against the radius increase, I put the mechanism to work on a previously grown rod-shaped shell with growth law VII, which had an enlarged radius. The circumferential breaking mechanism reduced the radius to the initial size with minimal changes to the length. The resulting shell still had about twice the initial length but recovered the initial diameter of the shell. This suggests that this mechanism is able to manipulate the radius of the shell independent of the length.

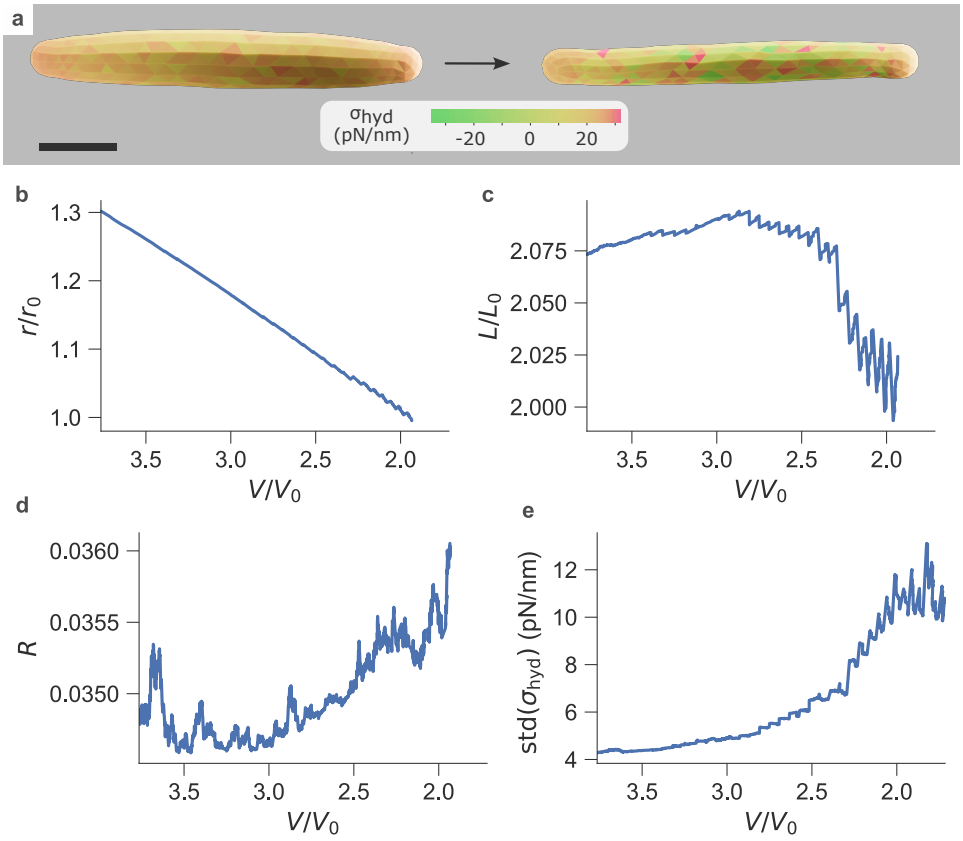

Figure 3.3.8: Radius recovery with circumferential reverse growth
a Geometry and surface stresses after applying the circumferential material reducing growth. **b**, **c** The growth reaction resulted in a radius recovered to the initial state while maintaining the increased length. **d**, The mechanism also slightly increased surface roughness. **e**, Peak surface stresses increased strongly.

I hypothesized that a combination of the circumferential growth mechanism and the previously employed growth law VII could elongate the cell with minimal changes to the radius. The circumferential growth mechanism corrects mistakes in the elongation mechanism that increase the radius of the shell. Both mechanisms operate in equilibrium, keeping the radius constant.

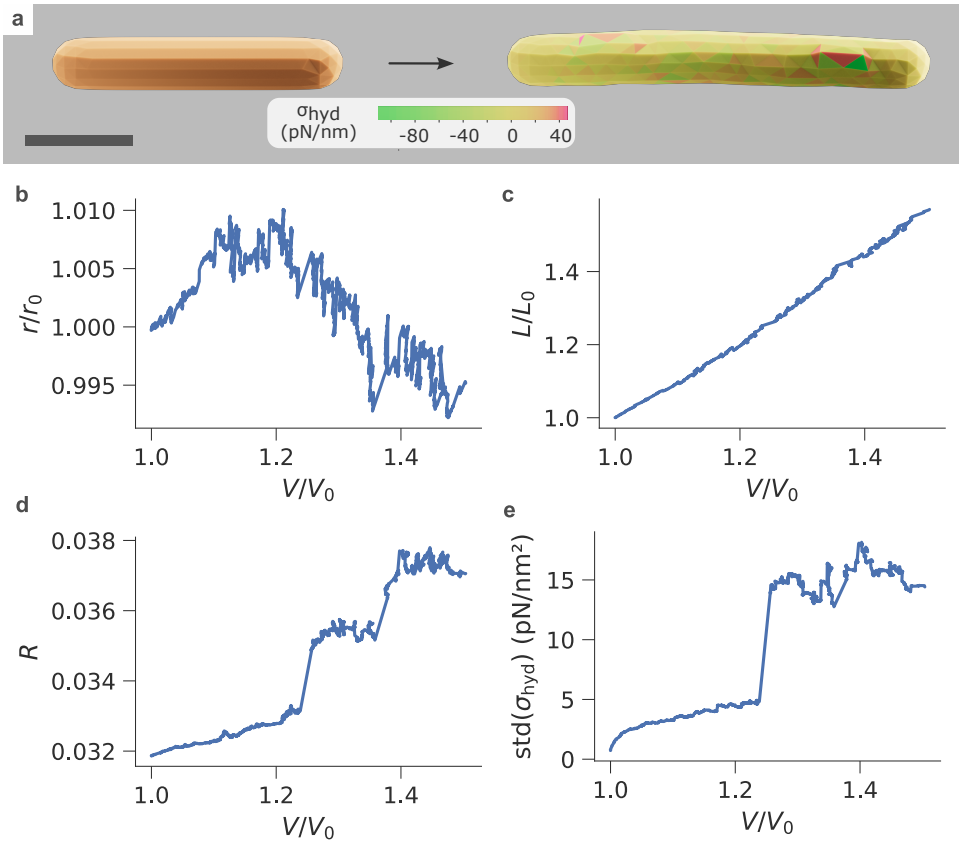

Figure 3.3.9: Radius preservation in rod-shaped bacteria
a Geometry and surface stresses after applying a combination of the circumferential growth mechanism and the growth law VII. **b**, **c** The growth reaction resulted in an elongation with minimal changes to the diameter. **d**, The mechanism also slightly increased surface roughness. **e**, Peak surface stresses increased significantly.

I iteratively adjusted the ratio between the elongation reaction rates and the circumferential reaction rates to reach the parameters shown under Eq. 3.3.7. I was able to achieve radius conservation in rod-shaped bacteria with this combination of growth laws. Fig. 3.3.9 shows the elongated bacterial shell compared to the initial state of the shell. More work is needed to understand how the increase in surface stresses could be avoided.

3.4 Conclusion

In this chapter I showed that the principles in the spring-based growth simulation can be applied to a simulation based on the finite element method. I demonstrated the ability of a strain-based growth mechanism to act as a negative feedback and incorporate material homogeneously. I also demonstrated that a combination of strain- and curvature-based growth is needed for rod-shaped shells in this model.

Chapter 4

Summary and Outlook

This thesis aimed to investigate growth of bacterial shells based on mechanical and geometrical cues. I developed the simulation framework presented in this thesis for the purpose of modeling growth of the shell as a stochastic process.

The initial physical model of the bacterial shell is based on the idea that a network of springs maps to an isotropic linear elastic material with a given thickness and Young's modulus. Local growth is introduced through increasing the rest lengths of the springs. I use an implementation of the Gillespie algorithm to select springs to grow. Growth reaction rates are based on local strains and curvatures.

Investigating the behavior of the simulated bacterial shell under high pressures revealed hyperelastic behavior. This can be avoided with typical shell properties for Gram-negative shells at low pressures. Simulating the growth of a spherical bacterial shell without any local property interaction resulted in visible roughness emerging on the surface. A strain-based growth law by contrast resulted in a smooth surface and consistently spherical shape. The same roughness emerged on a rod-shaped shell grown without local interactions. The strain-based growth law produced a smooth surface for the rod-shaped shell as well, however in addition to the elongation, an increase in radius was observed. This prompted me to investigate directional growth. My results indicate that for directional growth, preferred directions in the underlying spring mesh of the surface determine elongation and radius increase during growth. It also breaks the mapping to an isotropic linear elastic material, as triangle shapes in the spring mesh deviate from being equilateral. These shortcomings of the spring-based model prompted me to develop a second physical model of the shell based on the finite element method.

The finite element based model builds on the existing simulation framework but avoids the shortcomings of the spring-based model. COMSOL Multiphysics provides the interface for finding the geometry of the pressur-

ized shell. A major challenge was the introduction of local growth into the finite elements framework. This is achieved through the concept of initial strains in COMSOL.

Simulating deformations under the turgor pressures confirmed the linear behavior in the finite element based model over a wide range of pressures. I repeated the previously conducted growth simulation experiments in this model. Growth of a spherical shell without local property interaction resulted in high surface stresses, which would compromise the mechanical stability of the shell in the long term. Employing the strain-based growth law on the spherical shell by contrast resulted in a more robust shell, indicated by homogeneous surface stresses. This result was replicated in the rod-shaped cell. Here, a radius increase was observed in addition to elongation for strain-based growth. I achieved elongation with minimal changes to the radius by employing a combination of strain and curvature-based growth.

More work is needed to understand the rising surface stresses emerging with the elongation. Additionally, the simulation framework should be applied to simulate bending and shape recovery of rod-shape bacteria as presented in [24]. The simulation framework should also be applied to model the growth of Gram-positive bacteria, which have a thicker cell wall and can withstand higher pressures, to see if the results from Gram-negative bacteria can be replicated in Gram-positive bacteria.

Appendix A

Structure of the simulation package

Different functionalities of the simulation package are compartmentalized into modules, listed in table A.0.1.

The shell module maps the state of the shell onto three classes. The `Graph` class contains the properties related to the connectivity of the triangulation mesh. It is instantiated with a list of triangles, where each triangle is an array of three vertex indeces. This fully defines the mesh graph and allows us to compute other properties of the graph. If we give each vertex of the triangulation graph a 3D coordinate, then we have fully defined the geometry of the shell. We use the `graph` object and the vertex coordinates to compute other geometrical properties such as triangle surface areas, volumes, and normal vectors on the surface. This information is collected in the `Geometry` class, which contains a reference to its underlying `graph` object. The mechanics of the shell are modeled in the `Physics` class. It contains surface properties and the pressure. The `Physics` class is a base class for the `SpringModel` and `FiniteElementsModel` subclasses, which implement the respective physical models.

The energy function $E(\vec{r})$ and energy gradient function $\nabla_{\vec{r}} E(\vec{r})$, as well as geometric and mesh graph computations itself are not implemented as class methods. They are implemented in C++, which improves the execution time of the program significantly (See Appendix B).

The `load_meshes` module provides an interface to generate and load initial meshes for spherical, spherocylindrical and torus geometries.

The `energy_minimization` module contains an interface to a conjugate gradient algorithm in the SciPy library. We use this interface to minimize the spring based model energy (eq. 2.1.1). The interface class implements methods for initial pressurization and minimization after a growth step. The `initial_pressurization` method iteratively raises the pressure in small steps until the target pressure is reached. This approach leads to more

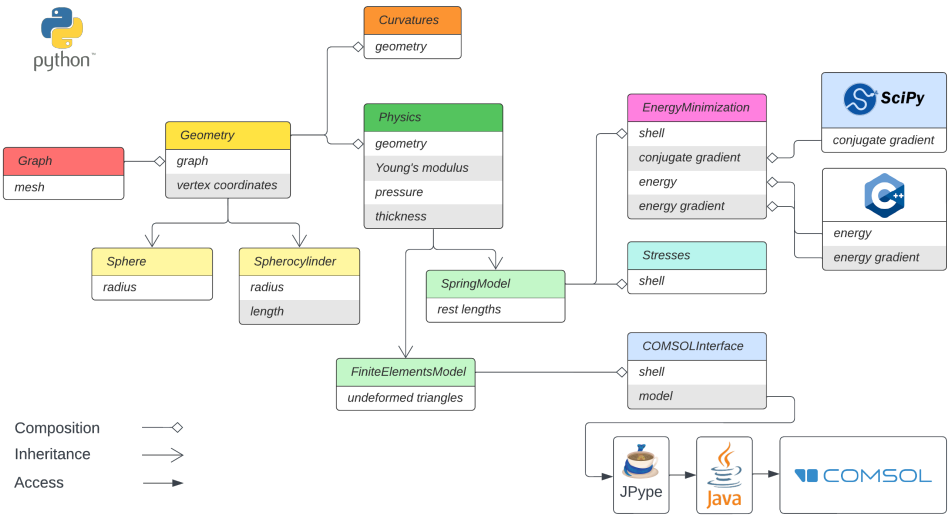

Figure A.0.1: Class relationships in the simulation package.
Properties of the shell are compartementalized into Graph, Geometry and Physics objects. The Geometry class is the base class for the Sphere and Spherocylinder sub classes. Curvatures are computed from the Geometry. The FiniteElementsModel and SpringModel classes are subclasses of the Physics class. The EnergyMinimisation class is used to compute the minimum configuration of the SpringModel. It interacts with the conjugate gradient function in SciPy and the C++ extension module. Stresses in the context of the SpringModel are computed with the Stresses class. The COMSOLInterface computes the pressurized configuration of the FiniteElementsModel and connects through the MPh extension with JPype and Java with the COMSOL server.

`load_meshes`	Load and generate meshes for spherical, spherocylindrical and torus geometry meshes.
`geometry`	Contains current geometric and mesh graph related properties of the shell.
`shell`	Contains current physical state of the shell.
`cython_extension`	Custom C extension written in Cython. Compute graph, geometric and physical attributes of the shell in C++.
`energy_minimization`	Find energy minimizing vertex coordinates. Implements an interface to the SciPy conjugate gradient function.
`stresses`	Compute stresses and strains in the surface for the spring based model (implemented by Cesar Lopez Pastrana).
`comsol_interface`	Compute deformations with the finite element method using the COMSOL software.
`curvatures`	Compute curvatures on the surface (implemented by Julius Lehmann).
`growth_mechanism`	Growing edges and triangles
`kinetic_monte_carlo`	Get index of next edge to grow and time step computed from growth reaction rates with the Gillespie algorithm.
`tensorboard_logger`	Log data from simulation run to tensorboard for a live visualization of observables and geometry.
`export`	Export data from simulation run.
`visualization`	Plot surface properties of the shell onto a 3D representation of the geometry.

Table A.0.1: Simulation package modules

consistent results compared with a naive implementation.

The `comsol_interface` module contains classes and functions to interact with COMSOL for mesh generation, write geometries to file, set parameters and initial strains and import computation results.

Stresses in the spring based model are computed with an implementation by César Lopez Pastrana, curvatures are computed with a module written by Julius Lehmann.

The module `growth_mechanism` contains an interface to grow edges and triangles by increasing rest lengths and undeformed triangles.

To select edges for growth and compute the time delta of a growth step, we use a Gillespie algorithm implemented in the `kinetic_monte_carlo` module.

We log observables and the shell geometry to tensorboard, which is a

data visualisation toolkit. This allows us to monitor the geometry of the shell and observables such as volume, roughness, radius and length during execution.

The `export` module contains functionality for data export. At the end of a simulation run, all data is saved as pandas `DataFrame`. The `shell` object is exported as Pickle file, which can be reopened to continue the simulation run. The pressurized geometries are exported in the `.obj` format, which can be read by visualization software. The export also includes a copy of the main python file from which the execution was started.

Putting the modules together, we can build a growth algorithm. The structure of the simulation package and interfaces of the modules allows the user access to physical parameters and critical functions while wrapping complex background tasks like mesh generation, energy minimization, data logging and data export into descriptive class methods. This allows the user to concentrate on the task of designing growth rules. The following code snippet is a full implementation of a random growth algorithm using the simulation package:

```python
import shell_growth as sg

sphere = sg.getSphere(vertexCount=500, radius=200)
parameters = {
    'osmoticPressure_p':    1,          # in atm
    'thickness_t':          2,          # in nm
    'youngsModulus_E':      50,         # in pN / nm
}
shell = FiniteElementsModel(sphere, parameters)

# starting comsol server on 12 cores
comsol = sg.ComsolInterface(shell, cores=12)
comsol.solve() # pressurize shell

growthMechanism = StrechTriangle(shell, growthFactor=0.2)
kinetics = KineticMonteCarlo(shell, fundamentalGrowthRate_k0=1)
logger = Logger(shell, "random_growth_sphere")

numberOfGrowthSteps = 1000
for growthIteration in range(numberOfGrowthSteps):
    # compute rates from local properties
    rates = shell.geometry.vertexAreas
    # find vertex to grow and compute time step
    _, triangleIndex, time = kinetics.getGrowthSite(rates)
    # grow shell locally at selected triangle
    growthMechanism.modifyShell(triangleIndex)
    # compute new pressurized geometry
    comsol.solve()
    # log data to tensorboard
    logger.logShellData()

logger.export() # Write data output to file
```

Listing A.1: Code example for a full random growth script on a sphere using the simulation package.

One can set up a simple growth algorithm with relatively few lines of code. In the example, a spherical shell with a radius of 100 nm, a thickness of 2 nm and a Young's modulus of 50 pN nm^{-2} is generated as a triangulated mesh with 1000 vertices. The pressure is set to 1 atm. The shell is pressurized with the `EnergyMinimization` class. The `growth mechanism` is selected as growth of rest lengths of edges surrounding a vertex by 5%, `kinetics` and `logger` objects are initialized. The shell is grown iteratively in 1000 iterations. In each growth iteration, growth rates are computed from local properties. In this case, the growth rates are just proportional to the Vonoroi areas surrounding the vertices. A vertex is selected for growth by the Gillespie algorithm implemented in `KineticMonteCarlo`. The `minimizer` computes the new pressurized configuration after the growth `mechanism` has grown the shell at the selected vertex. At the end of each iteration, observables and the mesh geometry is logged to tensorboard and can be inspected live while the shell is growing. After the growth loop is exited, all logged data is exported to file.

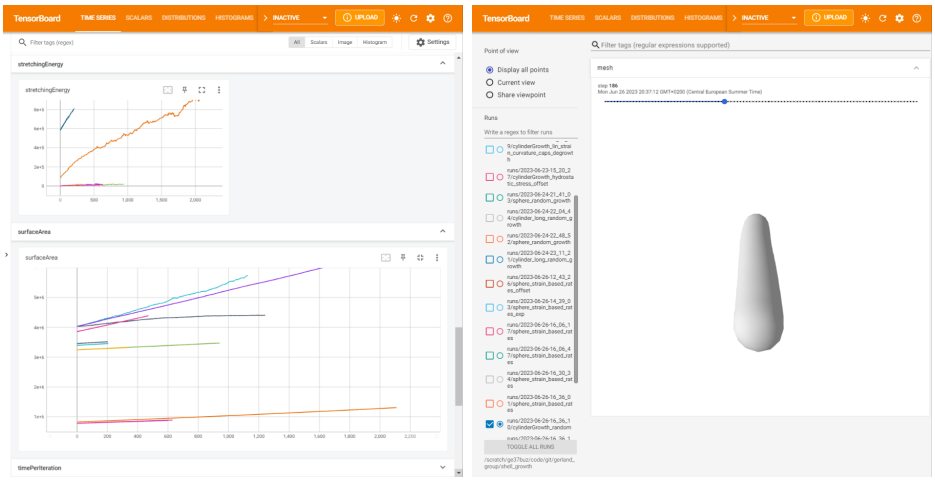

Figure A.0.2: Tensorboard visualization
Logging the data to tensorboard allows us to observe various parameters of the shell and even get alive view of the geometry while the simulation is running.

Appendix B

Improving the execution time of the spring based model simulation

The main operation limiting the execution time of the simulation is the minimization step that minimizes the energy. We use the conjugate gradient algorithm from the Python SciPy to minimize the energy, which repeatedly calls the energy and energy gradient computation routines. These were initially implemented with NumPy.

B.1 Moving from NumPy to Numba

	Numpy (s)	Numba jit (s)	Speed improvement
Growth	1710	43.1	40×
Minimization	1450	37.5	39×
Gradient	766	3.75	204×
Triangle normals	381	6.57	58×
Adjacent triangles	228	3.32	69×

Table B.1.1: Speed improvement of Numba just-in-time (jit) compiled code to a pure numpy implementation. Execution times are total cumulative execution times for a 1000 iterations of random growth of a sphere with 500 vertices. The largest improvement is made in the gradient computation, which was reduced by about 200×. The growth loop runs about 40× faster overall.

Julius Lehmann suggested the Python Library Numba, which works with NumPy arrays and compiles any Numba decorated function to machine code at execution time [31] . I implemented the most computationally intensive functions in Numba for a total execution speed improvement of 40×, reducing a typical growth run with a small sphere from about 28 min to 43 s. (see

table B.1.1).

For the intial tests we ignored the bending energy term because of its low contribution to the energy and compuationally complex gradient implementation. After implementing the bending energy and bending energy gradient computations in Numba however, it became clear that the bending gradient computation took up about 80% of the computational time even after several optimizations.

B.2 Moving from Numba to a custom C extension

I wrote a custom C extension in Cython for the simulation that would take care of the most computationally expensive tasks and could be called with Python functions. This allowed us to run the simulation near C speed while keeping a fast development time and the possibility to use Python extensions for other parts of the program.

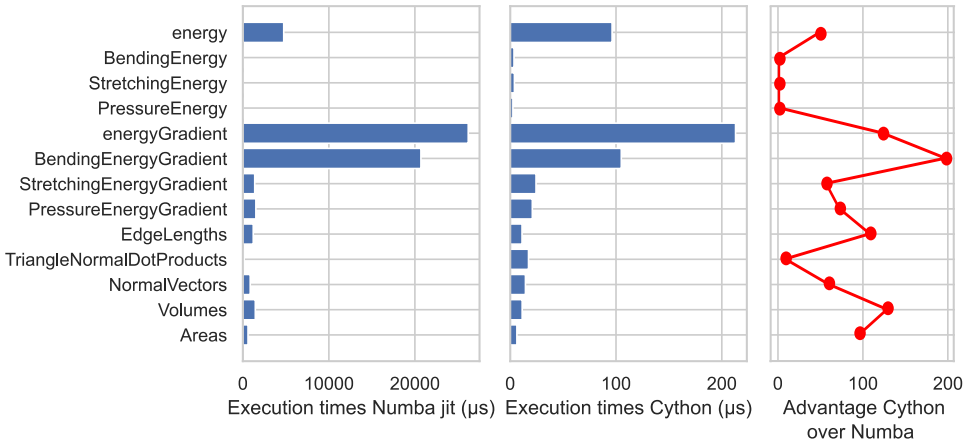

Figure B.2.1: Execution times using using Numba's just in time (jit) compiler and using a custom C++ extension written in Cython. Average execution times over 7 runs with 1000 iterations each, measured with the Python timeit module. The execution time in the Numba implementation is dominated by the bending energy gradient computation, taking 20.7 ms. I was able to reduce this time to 105 µs. The full energy and energy gradient computations are 49× and 123× faster, respectively. This results in a minimization time that is lowered by the same order of magnitude.

Cython is an optimising static compiler for the Cython programming language, which extends Python. It generates C++ code from Cython code that is then compiled as Python extension module. It supports calling external C libraries and static type declarations, which allows for very fast execution times [29].

I implemented all the mesh graph, geometric as well as energy and gradient computations in Cython and optimized them so that the Python exten-

sion would run as fast as normal C code. This reduced the typical execution time of a minimization step from 6 s to 156 ms. The energy and energy gradient computation times were reduced by 49× and 123×, respectively (see Fig. B.2.1).

Bibliography

[1] L. D. Landau and E. M. Lifshitz. *Mechanics, Third Edition: Volume 1 (Course of Theoretical Physics)*. 3rd ed. Butterworth-Heinemann, Jan. 1976. ISBN: 0750628960. URL: http://www.worldcat.org/isbn/0750628960.

[2] H. S. Seung and David R. Nelson. "Defects in flexible membranes with crystalline order". In: *Phys. Rev. A* 38 (2 July 1988), pp. 1005–1018. DOI: 10.1103/PhysRevA.38.1005. URL: https://link.aps.org/doi/10.1103/PhysRevA.38.1005.

[3] AL Koch. "The surface stress theory for the case of Escherichia coli: the paradoxes of gram-negative growth". In: *Research in microbiology* 141.1 (1990), pp. 119–130.

[4] Adrian M Whatmore and Robert H Reed. "Determination of turgor pressure in Bacillus subtilis: a possible role for K+ in turgor regulation". In: *Microbiology* 136.12 (1990), pp. 2521–2526.

[5] Jordan H Pollack and Francis C Neuhaus. "Changes in wall teichoic acid during the rod-sphere transition of Bacillus subtilis 168". In: *Journal of bacteriology* 176.23 (1994), pp. 7252–7259.

[6] Janet L Siefert't and George E Fox. "Phylogenetic mapping of bacterial morphology". In: *Microbiology* 144.10 (1998), pp. 2803–2808.

[7] Jeremy Pritchard. "Turgor pressure". In: *e LS* (2001).

[8] Kevin D Young. "The selective value of bacterial shape". In: *Microbiology and molecular biology reviews* 70.3 (2006), pp. 660–703.

[9] Lu Gan, Songye Chen, and Grant J Jensen. "Molecular organization of Gram-negative peptidoglycan". In: *Proceedings of the National Academy of Sciences* 105.48 (2008), pp. 18953–18957.

[10] W Vollmer, D Blanot, and MA De Pedro. *Peptidoglycan structure and architecture FEMS Microbiol Rev 32: 149-167*. 2008.

[11] Allan F. Bower. *Applied Mechanics of Solids*. Accessed: 2023-07-11. CRC Press, 2009. URL: https://solidmechanics.org/.

[12] Catherine Paradis-Bleau et al. “Lipoprotein cofactors located in the outer membrane activate bacterial cell wall polymerases”. In: *Cell* 143.7 (2010), pp. 1110–1120.

[13] Thomas J Silhavy, Daniel Kahne, and Suzanne Walker. “The bacterial cell envelope”. In: *Cold Spring Harbor perspectives in biology* 2.5 (2010), a000414.

[14] Yi Deng, Mingzhai Sun, and Joshua W Shaevitz. “Direct measurement of cell wall stress stiffening and turgor pressure in live bacterial cells”. In: *Physical review letters* 107.15 (2011), p. 158101.

[15] Julia Domınguez-Escobar et al. “Processive movement of MreB-associated cell wall biosynthetic complexes in bacteria”. In: *Science* 333.6039 (2011), pp. 225–228.

[16] Ethan C Garner et al. “Coupled, circumferential motions of the cell wall synthesis machinery and MreB filaments in B. subtilis”. In: *Science* 333.6039 (2011), pp. 222–225.

[17] Handan Arkın and Wolfhard Janke. “Gyration tensor based analysis of the shapes of polymer chains in an attractive spherical cage”. In: *The Journal of chemical physics* 138.5 (2013).

[18] Morgan Beeby et al. “Architecture and assembly of the G ram-positive cell wall”. In: *Molecular microbiology* 88.4 (2013), pp. 664–672.

[19] Ariel Amir et al. “Bending forces plastically deform growing bacterial cell walls”. In: *Proceedings of the National Academy of Sciences* 111.16 (2014), pp. 5778–5783.

[20] Yaron Caspi. “Deformation of filamentous Escherichia coli cells in a microfluidic device: a new technique to study cell mechanics”. In: *PLoS One* 9.1 (2014), e83775.

[21] Tristan S Ursell et al. “Rod-like bacterial shape is maintained by feedback between cell curvature and cytoskeletal localization”. In: *Proceedings of the National Academy of Sciences* 111.11 (2014), E1025–E1034.

[22] Carlotta Negri et al. “Deformation and failure of curved colloidal crystal shells”. In: *Proceedings of the National Academy of Sciences* 112.47 (2015), pp. 14545–14550.

[23] Fangwei Si et al. “Bacterial growth and form under mechanical compression”. In: *Scientific reports* 5.1 (2015), p. 11367.

[24] Felix Wong et al. “Mechanical strain sensing implicated in cell shape recovery in Escherichia coli”. In: *Nature microbiology* 2.9 (2017), pp. 1–8.

[25] Paul D Caccamo and Yves V Brun. "The molecular basis of noncanonical bacterial morphology". In: *Trends in microbiology* 26.3 (2018), pp. 191–208.

[26] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[27] *Cinema 4D: 3D computer animation, modeling, simulation, and rendering software.* https://www.maxon.net/en/cinema-4d. Accessed: 2023-07-21.

[28] *COMSOL Bracket Tutorial Initial Strains.* https://www.comsol.com/model/download/836281/models.sme.bracket_initial_strain.pdf. Accessed: 2023-03-21.

[29] *Cython - an overview.* https://cython.readthedocs.io/en/latest/src/quickstart/overview.html. Accessed: 2023-03-09.

[30] *MPh: Pythonic scripting interface for Comsol Multiphysics.* https://https://github.com/MPh-py/MPh. Accessed: 2023-07-12.

[31] *Numba: a High Performance Python Compiler.* https://numba.pydata.org/. Accessed: 2023-03-09.

[32] *PEP 8 – Style Guide for Python Code.* https://peps.python.org/pep-0008/. Accessed: 2023-07-21.

[33] *TensorBoard: TensorFlow's visualization toolkit.* https://www.tensorflow.org/tensorboard. Accessed: 2023-07-15.

[34] Alexander Stukowski. "Visualization and analysis of atomistic simulation data with OVITO-the Open Visualization Tool". In: *MODELLING AND SIMULATION IN MATERIALS SCIENCE AND ENGINEERING* 18.1 (JAN 2010). ISSN: 0965-0393. DOI: {10.1088/0965-0393/18/1/015012}.

Acknowledgments

I would like to thank Ulrich Gerland for the supervision and great freedom he allowed me during my thesis. I would like to thank Cesar L. Pastrana for his support, encouragement and the many fruitful discussions. I would also like to thank Julius Lehmann in this context and his contributions to the simulation package.

I want to thank my wife Farisa for her unconditional support and for always being there for me even in the most difficult of times. I want to thank my parents, Doris and Jan, for supporting me over the years and giving me the opportunity to study freely. Thank you Benno for your friendship and encouragement.