Simulation of Two-Dimensional Dry Foam Using a Vertex Model

Master's Thesis

Johan Teichert Bondorf
dnw147@alumni.ku.dk

March 4, 2017

Abstract

This thesis presents a simulation of a vertex-based model of two-dimensional dry foam. A treatment of the necessary physical theory behind the physics of foams will be presented as well as a thorough introduction to the dual mesh data structure used to represent the foam. The simulation uses a local relaxation approach with a backtracking method on the step direction with hard constraints to keep geometrics and physical properties valid at all times. This thesis contributes a more versatile dual mesh representation of the free boundary than previously done as well as an implementation of and comparison between three different approaches; the local, quasi-global and global approach. Among further contributions is an extensive treatment of the initialisation of the foam, ensuring a more robust model.

Contents

1 Introduction

Within the field of computational physics, dynamical simulations span a large variety of interesting applications. From simulations of volcanic ash clouds shaping in the sky to simulations of the movement of a tsunami moving across the ocean. From smoke dissipating in the air to stars orbiting the centre of a galaxy. The list of applications is endless. One area which is perhaps a little more unknown is simulations of foams. Its ubiquitous presence in our everyday lives, when it comes to liquids with foam forming capabilities, is not to be missed: the short-lived foam on the surface of milk when pouring a glass, the more long-lived foam in the shape of the head of a nice dark beer, the foam forming when brushing teeth, the foam on top of the hot water, if one is fortunate enough to have access to a bathtub, shaving foam or the foam in a fire extinguisher. The list goes on. All these are examples of wet foam. Wet foam consists of foam films surrounding the gas in the cells of the foam as well as liquid between the films with there being different extents of how wet the foam is. The head on a beer typically consists of a larger liquid fraction than that of soap foams. When there is no liquid in between the films, it is called *dry foam*.

Foam is not only present in easy-to-understand everyday examples as the ones presented above. Metal foam is a material used in industry, for instance as filters, energy absorbers or as light weight structural components.

As it turns out, foam-like structures are found in a large variety of places - a lot more than one would imagine. Both on a large scale, but in the past decades, quantum physicists have even suggested a link between quantum gravity and the existence of *quantum foam*, the idea that space-time at the Planck scale at $10^{-35}\,\mathrm{m}$ is not smooth, but exhibits foam-like structures. In fact, it is a theoretical concept that was already proposed as early as in the 1950's, but scientists have struggled to create a complete model to describe it. As was suggested, quantum foam is everywhere in space, also where there is no apparent matter. It is not to be thought of as foam in a traditional sense with bubbles moving smoothly and getting smaller or larger, but rather in a fluctuous way with virtual particles constantly appearing and vanishing on a very small time scale. The structures of such phenomena are suggested to be foam-like, because of tiny quantum fluctuations of space and time on extremely small scales - a result of Heissenberg's uncertainty principle. These foam structures are believed to scatter photons in a chaotic way [Kirillov and Turaev, 2007], although recent studies question this effect [Vasileiou et al., 2015].

This thesis, however, deals with the evolution process of traditional foam, also known as coarsening. It is driven by diffusion of a gas between the cell walls of the foam making some bubble cells shrink and others grow, while the whole foam in general will shrink. The foam will be restricted to be two-dimensional dry foam.

At the beginning of the work on this project, it was part of the plan to extend the simulation to be able to take advantage of parallel programming using for instance PyCUDA or Python's own multiprocessing threading interface. The reason for this is that many processes are identical calculations of different attributes throughout

the foam. Furthermore, the mesh data structure used to represent the foam is mutable, which is an advantage when doing parallel updates of the foam. However, emphasis was put on implementing different methods for solving the coarsening of the foam; a *local* method, a *quasi-global* method and a *global* method, and they would all require different methods of parallelisation as a result of their different nature, thereby making comparisons between them biased. As a result, the programming in parallel would be a great way for further work beyond the scope of this thesis, if choosing one of the three methods to speed up the code. For these reasons, the main focus of this thesis was to create an accurate foam coarsening simulation with a versatile underlying data structure making the program stable and able to cope with different initial foam configurations and foam developments throughout the simulation.

1.1 Previous work

[Kermode and Weaire, 1990] presented an overview of their implementation of two-dimensional dry foam on the basis of their initial paper on the subject presented in [Weaire and Kermode, 1983], and they form the foundations of the simulation presented in this thesis. The simulation uses an initial Voronoi network, the dual of the Delaunay triangulation, to initialise the foam. The simulation uses periodic boundary conditions, which separates it from the simulation in this thesis, but the keys of their simulation are the same physical equations and T1- and T2 topological operations, which have been the general driving forces of most foam simulations since. The equilibration of the foam follows from local surface energy minimisation in the shape of forming a local system of 5 linear equations describing desired equilibrium angles and areas in bordering cells for some junction in the foam. The derivatives used in the numerical model are calculated numerically with a forward difference approximation. The relaxation of the junction follows from solving the system of linear equations and finding an update step for the 5 variables, namely the 2-component position of the junction as well as the pressures of the three neighbour cells, but with the pressure update scaled to the number of neighbour cells of that particular junction.

[Weaire and Hutzler, 2001] published a book wrapping up a lot of the work that had been done so far by colleagues working with foam simulations, both dry and wet foam and in two and three dimensions. This book has also been used a lot in working on this thesis. Although it is not an original scientific paper, one of the authors, Denis Weaire, is a pioneer within the field of foam simulations, and a lot of the material in the book is built on material previously published by Weaire himself.

On the basis of [Weaire and Kermode, 1983], [Herdtle and Aref, 1992] developed a global relaxation method, where instead of a 5×5 local system of linear equations, the paper presented a single system of linear equations representing all junctions and couplings in the system. It used analytical derivatives instead of numerical derivatives. They still used periodic boundary conditions. The paper showed interesting results of the couplings in terms of the expected cell area scaling of the simulated foams being very sensitive to initial conditions. The paper claims that such a global approach including coupling effects is essential, and that the good

scaling results of earlier presented local methods were to a certain extend, somewhat fortuitous. This thesis shall not cast judgement of this, but will implement a global solver as well as a local one to compare results.

Still in the relatively early stages of the field of foam simulations, [K.Nakashima et al., 1989] presented another approach to foam simulations using a vertex model in the sense, that the foam junctions were represented as vertices whose motions were governed by the free energy and Rayleigh dissipation function, which are easily computed given the for the foam constant line tension energy and the Onsager kinetic coefficient along with the velocity and position of the vertex. Furthermore, this model completely ignores pressure and diffusion in order to create an as minimalistic model as possible, while still being able to do statistics on the late stages of the movements in foam structures. What is thereby interesting with respect to this thesis is the extensive statistical work on the scaling behaviours of the vertex model.

[Kelager, 2009], at the Department of Computer Science at the University of Copenhagen, wrote his master's thesis on a vertex-based simulation on two-dimensional dry foam with the same sort of equilibration approach in the numerical method as [Weaire and Kermode, 1983]. The author used a polymesh data structure to represent the foam and modelled an open boundary by introducing the so-called *ghost bubble*, representing the outside world and its constant pressure. By introducing this to the numerical model, the author could investigate world facing boundary behaviours and go beyond having periodic boundaries, which represent a cut-out of an infinite foam sample.

[Vedel-Larsen, 2010], also at the Department of Computer Science at the University of Copenhagen, built his master's thesis on the works of Kelager and made some major changes to the simulation. Instead of a polymesh, the author introduced a simpler, but still as least as powerful, underlying triangle mesh data structure holding information about adjacencies and physical variables. The ghost bubble was changed, and *shark fin faces* on the boundary were introduced due to the switch from a polymesh datastructure to a triangle mesh data structure. Among other major contributions was an outline for paralellising the simulation using a GPU-based implementation.

1.2 Contributions to the field of foam simulations

This thesis primarily builds on the work done by Kelager and Vedel-Larsen in particular. This thesis, however, has a set of further contributions.

The work with this thesis has shown a way to create a two-dimensional foam simulation from scratch in Python. Python is a high-level programming language and is accessible to anyone for free and has a lot of extensive libraries. To represent the foam, this thesis shows a way to do this with a triangle mesh data structure (also to be known as the dual mesh in this thesis, see Section 3.1) without any loss of information. It is done using the mesh library OpenMesh, written in C++, but it has an extension for usage in Python. The choice of a mesh library was also part of working on this project, and a thoughts on the choice of mesh library are being discussed later.

Furthermore, this thesis presents a robust method of representing the boundary, enabling the possibility of two-sided bubbles on the boundary, which has not been possible in previous work. This is done by representing the boundary differently in the dual mesh.

Emphasis has also been placed on the initialisation of the foam, before the simulation is started. This way, a more homogeneous initial foam is achieved, while still posessing enough variation for the foam to be realistic. In some cases, [Vedel-Larsen, 2010] experienced strange artefacts an anomalies on the boundary after only a handful of iterations. Doing a more thorough initialisation as well as reworking the boundary representation in the dual mesh, a much more stable boundary is provided in this thesis.

Additionally, three different simulation methods with different numerical models will be introduced, implemented and eventually compared.

1.3 Acknowledgements

I would like to thank my supervisor on this thesis, associate professor Kenny Erleben of the Department of Computer Science at the University of Copenhagen, for his untiring willingness to answer my questions with patience and understanding, and for the many discussions on the theory and implementations of foam simulations, we have had over the past six months. They have inspired me in my work with foam simulations and have helped me keep the overview of the project.

2 The physics of foams

The complex structures of foams are fascinating and seem very complicated. Yet, they are governed by only a handful of equations and laws. This chapter will introduce these physical laws and describe and justify, which simplifications are made in the implementation.

2.1 The laws of Plateau

A foam in equilibrium obeys some specific laws, known as the laws of Plateau. These simple and very important laws were formulated in 1873 by the Belgian physisist Joseph Plateau [Herdtle and Aref, 1992] and will play a fundamental role throughout this thesis. The laws are as follows [Weaire and Hutzler, 2001]:

- **Equilibrium rule A1**
  In a dry foam in two or three dimensions, only three films can intersect at a time and must do so at 120° where three surfaces meet in a Plateau border junction (3D) or where three lines meet in a Plateau junction (2D).
- **Equilibrium rule A2**
  For dry foams in three dimensions, we may assert, following Plateau, that at the vertices of the structure no more than four of the intersection lines (or six of the surfaces) may meet, and that this tetrahedral vertex is perfectly symmetric. Its angles all have the value $\phi = \cos^{-1}(1/3) \approx 109.5°$*, sometimes called the Maraldi angle.*
- **Equilibrium rule B**
  In a wet foam, where a Plateau border joins an adjacent film, the surface is joined smoothly, that is, the surface normal is the same on both sides of the intersection.

Equilibrium rule A1 was originally an observational fact. However, one can show this fact by considering the T1 topological process, which will be treated in more detail in Section 2.4.1.

As the work of this thesis is focussed on two-dimensional dry foams, only equilibrium rule A1 will apply to the work presented in this thesis. For the remainder of this thesis, this will be known as *Plateau's first rule of equilibrium*. Equilibrium rule A2 is, in a sense, an elaboration of equilibrium rule A1 and describes the geometry of intersections and angles at borders and tetrahedral vertices in three-dimensional foams. At the borders between three cells, the angles between cell walls are 120°, whereas the angles at the tetrahedral junctions between four cells and six borders form angles of 109.5°.

In two dimensions, this situation simplifies, since there are no angles at the cell borders, and we shall only focus on the angles of 120° at the Plateau junctions.

Equilibrium rule B applies to wet foams and is implicit in dry foams, since the films are joined everywhere in the foam as depicted in Figure 2.1.

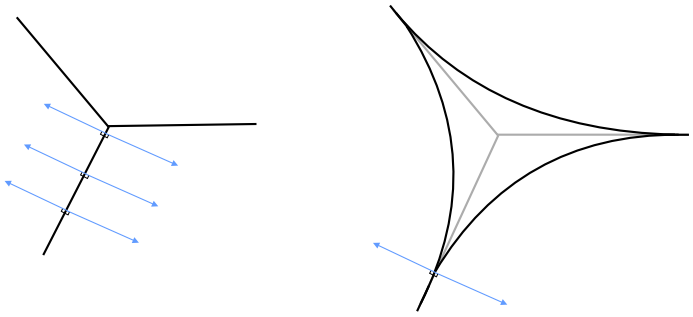


Figure 2.1: The figure on the left shows a junction in a dry foam, whereas the figure on the right shows the same, only in a wet foam. The blue arrows show the surface normals. In the dry foam, the surface normals on each side of the intersecting films are opposite to each other at 180° everywhere in the foam. Compared to the wet foam with liquid in between the films, the films are imagined to be infinitesimally close to each other with no liquid in between.

2.2 The law of Laplace-Young

In a two-dimensional dry foam, all films between cells are circular arcs, if the pressure difference between the cells is different from zero [Weaire and Hutzler, 2001]. To describe the radius of curvature of these arcs, it turns out there are only a couple of variables determining this. The law of Laplace-Young governs this radius of curvature and plays a fundamental role in the structure of foams, and will therefore be a cornerstone of this thesis as well.

Consider a single two-dimensional bubble with radius r. The surface tension γ is a force per unit length that acts on the bubble film and has dimensions $\mathrm{N/m}$. It is a physical constant related to the chemical composition of the bubble material. The surface tension acts along the direction of the film. Shrinking the circumference of the bubble with film length l by a very small amount of δl can also be written in terms of lowering the radius by an amount of $\delta l = 2\pi\delta r$, ultimately lowering the potential surface energy. With the surface energy given by the surface area times two times the surface tension γ, the energy difference is given as

$$\delta E = 2\gamma\delta l = 2\gamma 2\pi\big((r+\delta r) - r\big) = 4\gamma\pi\delta r \tag{2.1}$$

Similarly, the work done by the pressure inside and outside the bubble can be written as the pressure difference acting as a force on the film times the area it shrinks the bubble, thereby giving

$$\begin{aligned}\delta W = \Delta p\, 2\pi((r+\delta r)^2 - r^2) &= \Delta p\, \pi(r^2 + 2r\delta r + (\delta r)^2 - r^2) \\ &\simeq \Delta p\, 2\pi r\, \delta r \end{aligned} \tag{2.2}$$

where Δp is the pressure difference between to adjacent cells in a dry foam,

$$\Delta p = p - p_{neighbour}$$

Since we work in the infinitesimal limit, we can reasonable assume that $(\delta r)^2 \simeq 0$. Now, no other external forces than the pressure act on the bubble, and we can thus equate the loss of surface energy (2.1) with the work done by the excess pressure (2.2):

$$\begin{aligned} 4\gamma\pi\delta r &= \Delta p\, 2\pi r\, \delta r \\ \Updownarrow & \\ \Delta p &= \frac{2\gamma}{r} \end{aligned} \tag{2.3}$$

also known as the law of Laplace-Young [Weaire and Hutzler, 2001]. Even though the derivation is based on an example of a single-sided bubble, this law also holds for n-sided bubbles, where each film has a curvature radius r_i and a pressure difference Δp_i associated with it.

In the case of a wet foam, this also holds, but here the pressure differences refer to pressure differences between gas and liquid, thus enabling different radii of curvature on each side of the wet film between cells [Weaire and Hutzler, 2001].

Rearranging (2.3), one can write an expression for the curvature radius of the film:

$$\boxed{r = \frac{2\gamma}{\Delta p}} \tag{2.4}$$

2.3 The law of von Neumann

Having established some equations and laws that describe some static properties of the dry foam, let's move on to the dynamics of the foam. How does the diffusion affect the change in area of the cells in the foam?

2.3.1 The sum rule in two dimensions

In order to derive von Neumann's law, one makes use of a little trick. Consider an n-sided bubble. Each bubble film spans an angle, which is related to its curvature radius and the length of the bubble film. The length of the film l_i can be written

as the fraction of the angle it spans θ_i compared to angle of the full circle 2π with circumference O_i and curvature radius r_i:

$$l_i = \frac{\theta_i}{2\pi} \cdot O_i = \frac{\theta_i}{2\pi} \cdot 2\pi \cdot r_i = \theta_i \cdot r_i \tag{2.5}$$

$$\Updownarrow \tag{2.6}$$

$$\theta_i = \frac{l_i}{r_i} \tag{2.7}$$

A tangent to a circle undergoes a rotation of 2π with a rotation of the full circle. Now, consider the three-sided bubble shown in Figure 2.2.

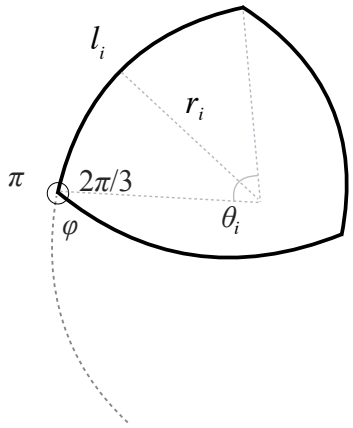


Figure 2.2: An illustration of a three-sided cell meeting Plateau's first rule of equilibrium. When this is the case, the angle φ is invariant.

Over the length of the film with length l_i, the tangent turns the same angle as the angle θ_i as shown in Figure 2.2, since the films are all circular arcs. Now, when the tangent meets the next film, if Plateau's first rule of equilibrium holds and the foam is in a state of equilibrium, the tangent instantly turns an angle of φ and onto the next film, where it once again turns over an angle related to that particular film. But how large is this angle φ? The figure shows the pivotal point, where the tangent turns and the three angles associated with this point. The inner angle is $2\pi/3$, which we know from Plateau's first rule of equilibrium. The outer angle is π (180°), and thus we know that

$$\frac{2\pi}{3} + \varphi = \pi$$

$$\Updownarrow$$

$$\varphi = \pi - \frac{2\pi}{3} = \frac{\pi}{3} \tag{2.8}$$

We know, that the tangent will turn a total of 2π in the course of moving over all films in the cell. Since φ is invariant for each junction of bubble films, this is a constant added to each such junction, and hence one can write

$$\sum_i^n \theta_i + n\frac{\pi}{3} = \sum_i^n \frac{l_i}{r_i} + n\frac{\pi}{3} = 2\pi \ , \tag{2.9}$$

which ultimately leads us to the *sum rule*

$$\sum_i^n \frac{l_i}{r_i} = 2\pi\left(1 - \frac{n}{6}\right) \tag{2.10}$$

This of course holds for any n-sided cell in equilibrium. But it also tells us something interesting about the number of sides in a cell compared to the curvature radii of its films, and thus the pressure in that particular cell compared to its neighbour cells. For any cell with $n > 6$, by (2.10), the sum of angles $\sum_i^n \theta_i = \sum_i^n \frac{l_i}{r_i}$ must be negative. The only way for this to become negative, is if some or all r_i are negative. Recalling the law of Laplace Young (2.4), this means we must have $\Delta p < 0$ and thus larger pressures in some or all adjacent cells causing the corresponding films in the cell to bend inwards.

2.3.2 Derivation of von Neumann's law

Fick's second law of diffusion is a partial differential equation describing how concentrations change with respect to time in a medium with a given spatial concentration gradient. The general form of this is

$$\frac{\partial c}{\partial t} = D\nabla^2 c \ ,$$

where c is the concentration of the medium, D is a diffusion constant and ∇^2 is the Laplacian.

In the case of a two-dimensional foam where cells are separated by films, the spatial concentration derivatives, i.e. the Laplacian, turn into delta functions, since the concentrations change instantly turning the derivatives into differences in concentrations on each side of the membrane, the film in the case of the foam. However, instead of looking at change in concentrations, we are interested in changes in areas. And by assuming constant temperature, the ideal gas law makes us able to write Fick's law in terms of pressure differences instead of concentration differences. In a two-dimensional foam, Fick's second law of diffusion can hence be written in such a way that it shows us a direct proportionality between diffusion and film length l and the pressure difference between two adjacent bubbles [Weaire and Hutzler, 2001]:

$$\frac{\mathrm{d}\,A}{\mathrm{d}\,t} = -\kappa \cdot l \cdot \Delta p \,, \tag{2.11}$$

where κ is the permeability constant of diffusion describing the magnitude of gas molecules that the bubble film in the foam lets through by diffusion.

When dealing with a foam, a cell is often exposed to neighbour cells with different pressures bordering the cell with different film lengths. Summing all neighbour cells turns (2.11) into

$$\begin{aligned}\frac{\mathrm{d}\,A}{\mathrm{d}\,t} = -\kappa \cdot \sum_i^n \Delta p_i \cdot l_i &= -\kappa \cdot \sum_i^n \frac{2\gamma}{r_i} \cdot l_i \\ &= -2\gamma\kappa \cdot \sum_i^n \frac{l_i}{r_i}\end{aligned} \tag{2.12}$$

As beforehand, Δp is negative, whenever the surroundings have higher pressure, thus making the gradient positive and the gas diffusing into the cell thus enlarging the area.

Inserting (2.10) into (2.12) yields

$$\frac{\mathrm{d}\,A}{\mathrm{d}\,t} = -2\gamma\kappa \cdot \left[2\pi\left(1 - \frac{n}{6}\right)\right] = 2\pi\gamma\kappa \cdot \left(\frac{n}{3} - 2\right) = \frac{2\pi}{3}\gamma\kappa \cdot (n-6) \tag{2.13}$$

Consequently, we have two different versions of von Neumann's law, as derived in (2.12) and (2.13). Summing, von Neumann's law states that the change in cell area of an n-sided bubble in two-dimensional dry foam can be calculated either by the pressure differences to adjacent cells or the number of sides of the bubble itself:

$$\boxed{\frac{\mathrm{d}\,A}{\mathrm{d}\,t} = -\kappa \sum_i^n (p - p_i) l_i} \tag{2.14}$$

$$\boxed{\frac{\mathrm{d}\,A}{\mathrm{d}\,t} = \frac{2\pi}{3}\gamma\kappa(n-6)} \tag{2.15}$$

The equality between these two different takes on von Neumann's law (2.14, 2.15) is quite remarkable. The pressure part is intuitive, but to know the change in area solely by knowing the number of sides in the bubble is very powerful. It all comes down to the laws of Plateau, as shown in the derivation in (2.8). Therefore, it is very crucial to stress that (2.15) is limited to the case, where Plateau's first rule of equilibrium holds true.

But if this is the case, a bubble with $n < 6$ will shrink in area, whereas a bubble with $n > 6$ will grow. Yet, an 6-sided bubble will not have any area gradient, and a foam with only 6-sided bubbles, and where Plateau's first rule of equilibrium holds true, will be in a state of perfect equilibrium. A topic, which we will revisit for a brief moment in Section 2.4.2.

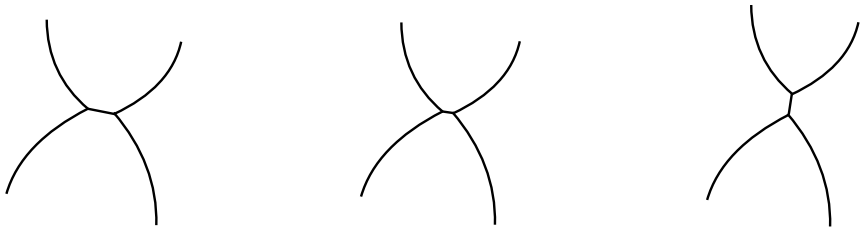
The use of pressures (2.14), however, was derived directly from Fick's law and holds true for any cell, also when it has non-Plateau angles. However, a foam not obeying Plateau's laws will generally quickly rearrange itself in such a way that Plateau's first rule of equilibrium is fulfilled, and thus both versions of von Neumann's law will give the same result in the vast majority of cases. But as [Stavans and Glazier, 1989] showed with real world experiments of two-dimensional foams between two glass plates, Plateau's first rule of equilibrium does not *always* hold, and as a result, (2.14) is used as the general formula in this thesis and in the implementation. Additionally, as will be seen later in this thesis, Plateau's laws are not obeyed in the initial foam configuration, but after only a handful of iterations, the simulation will have smoothed out most such violations of Plateau's laws.

2.4 Equilibrium structures

Combining Plateau's first rule of equilibrium with von Neumann's law, we can describe a foam in equilibrium: All films shall meet at an angle of 120°, and the pressure must be equal in all cells. If these criteria are not met, the foam will not be in a state of equilibrium, and will immediately rearrange itself in order to move towards a state of lower energy, until it eventually finds itself in a state of equilibrium.

2.4.1 Stability and properties of a Plateau junctions

But why is it that we cannot have four films meeting in one junction? Why must we have Plateau junctions with three incoming films? An unstable foam will have angles different from 120°, but we will never obtain four film junctions over a longer period of time, since such a configuration is not stable, not even when the foam is not in equilibrium. When two three film junctions in a foam move towards each other, they will immediately perform a so-called T1 operation once they meet, which is a topological operation that alters the connectivities between the adjacent cells. Figure 2.3 shows such a process.



(a) Two junctions move towards each other.

(b) The two junctions are close to forming a 4 sided junction.

(c) Immediately, a T1 operation is performed resulting in two new junctions.

Figure 2.3: An example of how foams perform, when two junctions move close to each other. The two junctions meeting would form a four-sided junctions, however, this is an unstable state, and the foam immediately rearranges. In Plateau's model and based on observations, this is indeed what is observed.

[Spencer et al., 2017] treated this instability in detail and came with a mathematical proof, why one cannot have a four-sided junction. The authors first stated two conditions, which must hold in order for a fourfold junction to be stable in a general vertex model. These conditions were based on the tension of the shared film (edge) between the two junctions (vertices) and the forces of the four outgoing edges in the shape of two inequalities. By using proof by contradiction and reaching an angle larger than some threshold, which the stability conditions demanded should be larger than the angle, the authors showed that such a configuration cannot exist in the Plateau model. For a more detailed walk through, I will refer to the paper by M. A. Spencer, Z. Jabeen, and D. K. Lubensky [Spencer et al., 2017] .

2.4.2 The honeycomb structure

The process of relaxation and equilibration of foam minimises the energy in the foam, which is directly proportional to the total film length in the foam [Kermode and Weaire, 1990]. In Section 2.4.1, we treated the threefold property of a Plateau junction. As the foam will always move towards a state of minimum surface energy, it will also have the junctions arranged, as to achieve minimum energy. An analogy to help the realisation of the threefold global junction property can be observed in nature.

The structure of a honeycomb is a remarkable phenomenon. Given a fixed cell area, the honeycomb structure reveals the minimum amount of material (comparable with bubble films in the case of foams) needed to create cells of equal size. This has been a well known fact for years, but was first proved theoretically in 2001 by mathematician Thomas Callister Hales [Hales, 2001]. This is in total agreement with von Neumann's law, especially clear when observing (2.15), where a cell with six sides will experience no diffusion, if Plateau's first law of equilibrium is fulfilled.

Figure 2.4: [Neatorama, 2013] Honey bees have instinctively been building honeycombs using the optimal hexagonal structure throughout time, thereby saving material at optimal revenue.

2.5 Foam statistics

Once one has a simulation up and running, it will be nice to have something to measure the results of the simulation up against in order to validate the simulation. Certain statistical properties associated with the evolution of foam both in the real physical world as well as in other vertex-based simulations can be applied in order to validate the simulation.

2.5.1 Aboav-Weaire's law

The Aboav-Weaire is an empirical law relating the average number of sides of cells neighbouring an n-sided cell.

Let μ_2 define the *second moment*, describing the spread or variance of n-sided cells

$$\mu_2 = \sum_n (n - \bar{n})^2 p(n) = \sum_n (n - 6)^2 p(n) \ , \tag{2.16}$$

where $\bar{n}$ is the average number of sides for a cell, which is 6 in the case of an *infinite* sample of dry foam in two dimensions [Weaire and Hutzler, 2001]. $p(n)$ is the distribution of n-sided cells in the particular sample.

Aboav-Weaire's law gives the average number of sides of neighbour cells to an n-sided cell:

$$\boxed{m(n) = 6 - a + \frac{6a + \mu_2}{n}} \tag{2.17}$$

where a is a parameter related to the foam in question. μ_2 can be calculated, whereas a can be fitted, and one can for instance investigate, whether different foam configurations statistically have the same values for a.

2.5.2 Scaling

To see how the simulation fares over time, it is interesting to look at the scaling properties of the foam in the simulation.

One property that changes through the process of foam coarsening is of course the cell area, which is related to the average radius of a cell in the foam. Apart from the transient phase in the very beginning of the simulation, one should expect a scaling of the average cell radius $\bar{r}$ to increase proportionally with the square root of the time:

$$\boxed{\bar{r}(t) \propto \sqrt{t}} \tag{2.18}$$

and similarly the average cell area $\bar{A}$ to increase proportionally to the square of $r(\bar{t})$, and thus proportional to the time [K.Nakashima et al., 1989]:

$$\boxed{\bar{A}(t) \propto t} \tag{2.19}$$

2.5.3 Other measures

Other physical quantities in the foam can be analysed in order to see, how the physics behaves in the simulation. Those can for instance be

- The distribution of pressure
- The distribution of number of sides in the cells, hereby the variance or the second moment μ_2
- The distribution of cell areas
- The total film length in the foam. This is directly proportional to the surface energy in the foam, i.e. the energy of the foam

Section 6 will visualise these quantities and analyse the results.

2.6 Simplifications

This thesis presents a method for simulating two-dimensional dry foam. In the real world, this is an idealised model and certain factors have been neglected. Three-dimensional foams and wet foams are extensions to this, but would build upon many of the same key components, albeit with necessary additional considerations needed.

Below is a list of some of the simplifications made in this idealised model of a two-dimensional foam compared to three-dimensional and wet foams.

- The simulation treats two-dimensional foam only .

- Only dry foam is treated. This means that the films separating the cells in the foam are thought to be massless and as having infinitesimal width with no liquid in between. The important subject of drainage experienced in wet foams is thus not part of this model.
- The two-dimensional dry foam is though to be ideal, but in fact, it is more a approximation than a real possible phenomenon. A real world representation of two-dimensional dry foam could be obtained by having two parallel glass plates very close to each other and let some foam in between drain, until one would reasonably be able to call the foam dry. [Janiaud et al., 2007] showed rheological studies analysing the impact of viscous drag forces between foam and glass and the extend of their effects.
- Thermodynamic processes due to temperature are neglected, since they play an infinitesimal role, and the temperature is considered constant throughout the whole foam regime and throughout the whole coarsening period. Diffusion is dependent on temperature, but if no outside factors alter the temperature, it will remain constant.
- The energy of the foam is thought of as the total film length. Thus, any kinetic energy in the moving gas through diffusion is treated as negligible.
- The surface tension of the films is considered to be constant.

3 Representation of the foam

The visual appearance of foam exhibits interesting and beautiful structures. One could represent this structure directly in data structures by for instance the junctions of the bubbles and the curvatures of the bubble films.

This thesis, as mentioned, uses another underlying mesh data structure presented by [Kermode and Weaire, 1990] and also used by Vedel-Larsen in his master's thesis [Vedel-Larsen, 2010], namely a triangle mesh - the dual mesh.

3.1 Primary and dual mesh

The mesh structure used to represent the foam shall be referred to as the *dual* mesh throughout this thesis, whereas the actual foam structure as it appears to the eye shall be known as the *primary* mesh. The origin of these terms gets its inspiration from the mathematical concept of duality.

The primary and dual meshes are, mathematically speaking, each others duals. This might sound a little confusing, and thus this section explains the concepts with some figures and examples. When initialising the foam, knowing the dual mesh lets us know the primary mesh without loss of information and vice versa. Later on, this duality will break, as the geometrics of the dual mesh will no longer be of importance, as clarified in Section 3.3.9.

[K.Nakashima et al., 1989] presents this exact representation by first constructing the Voronoi network of a random two-dimensional point cloud and doing an equilibrium process of this network. This network is their version of the primary mesh in this thesis. From this, they construct the triangle mesh, which is the dual of the Voronoi network.

Since the meshes are each others' duals, we are free to do the opposite, and so we do, as did [Kermode and Weaire, 1990], namely by constructing the dual mesh first and using this to create the primary mesh instead of doing the opposite.

The advantage of the dual mesh is that a triangle mesh is simpler to implement and understand than a polymesh, which one would use to use to represent the primary mesh on its own. Whereas [Kelager, 2009] represented the foam solely as a primary mesh, [Vedel-Larsen, 2010] introduced a dual mesh as an underlying data structure, an approach also adopted in this thesis.

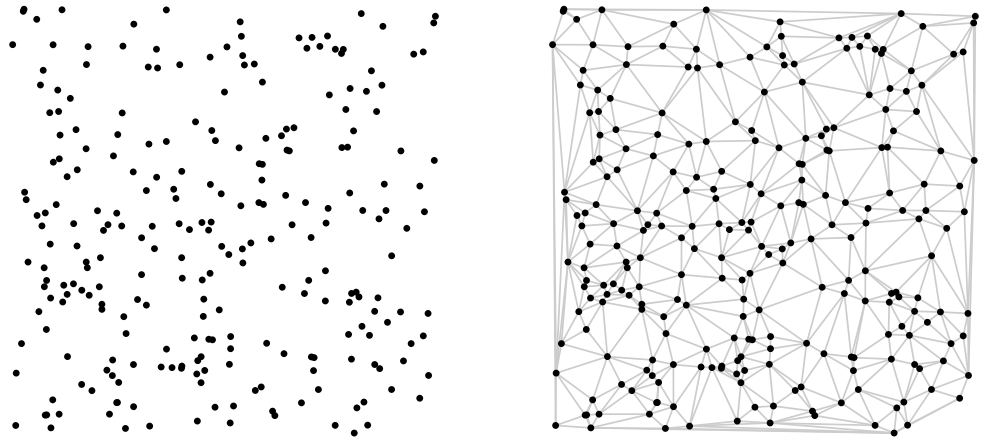
3.1.1 Delaunay triangulation

In the above section, we argued that a primary mesh can be initialised by first creating the dual mesh. But how is this dual mesh initialised in the first place? It is constructed by spreading points randomly in two dimensions within a desired range. The *Delaunay triangulation* connects these points to a triangular mesh, where no edges cross each other and no points are isolated. From now on, we shall refer

to the points as *vertices*. A vertex can have more outgoing edges, but all *faces* (geometrical figures bordered by the edges) in the the mesh are triangles. Generally, the Delaunay triangulation always ensures that no vertex lies inside the circumcircle of any face in the triangulation.

Since we shall not implement the Delaunay triangulation from scratch (there are plenty of methods, both in the Matplotlib library for Python as will as the OpenMesh library, that I will use), we well not go through a detailed description of the Delaunay triangulation. [Berg et al., 2008] has a nice and thorough description of this whilst also highlighting theorems applying to the Delaunay triangulation.

Figure 3.1 shows an example of a Delaunay triangulation. It only consists of triangles, no edges are crossing each other, no vertex lies on an edge and therefore no vertex lies inside the circumcircle of any other triangle.



(a) Random point cloud **(b)** The Delaunay triangulatuon

Figure 3.1: A visualisation of a dual mesh created from a random point cloud.

3.1.2 Drawing the primary mesh using the dual mesh

Having constructed the dual mesh, one can draw the primary mesh solely by knowing faces and vertices of the dual mesh. As [Kermode and Weaire, 1990] describes, one way to do this is to connect the centres of the faces to centres of adjacent faces. Now, Plateau's laws describe a foam in equilibrium, which is not the case of our initial foam, but the rule of the existence of only three-fold junctions still holds. The vertices in the dual mesh each represent a bubble, apart from on the boundary, which will be treated in Section 3.3.6. Figure 3.2 shows the primary mesh drawn on top of the dual mesh from Figure 3.1b. Each bubble film inside the primary mesh crosses an edge in the dual mesh. The boundary is yet to be treated and has been left out thus far.

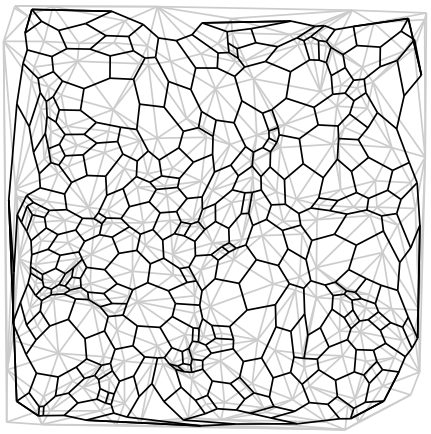


Figure 3.2: Primary mesh drawn on top of dual mesh in Figure 3.1b

3.2 Mesh attributes

The representation of the dual mesh does not only consist of vertices and faces. Other variables are attached to the mesh in the shape of properties. Such a variable could be the cell pressures. This is one of the strengths of this representation. Pressure values can be stored in the vertices of the dual mesh, since there is one and only one such vertex inside a bubble in the primary mesh. As a result, each non-boundary vertex in the dual mesh represents a cell in the primary mesh.

To give an overview of the connections of where things are stored and what they represent, Table 1 shows the various attributes associated with dual and primary mesh. The different columns in the table describe:

- The ***foam attribute***, the physical descriptor
- The ***dual mesh feature***, how and where it is represented in the dual mesh
- Which ***labels*** are used to identify the feature. The Plateau angle needs two labels, since it needs the vertex label of the neighbouring bubble to identify which of the three angles in the Plateau junction it faces
- Type of variable
  - ***Physical variables*** are ones that are primary physical variables driving the model, and upon which other variables depend
  - ***Derived variables*** are also physical, but can be derived from the physical ones
  - ***Helper variables*** are ones used for the visualization in the primary mesh, but they do not have a physical importance as such

Foam attribute	Dual mesh feature	Stored in label	Type of variable		
			Physical	*Derived*	*Helper*
Centre coordinate of cell	Vertex	v			✓
Cell area	Vertex	v		✓	
Pressure	Vertex	v	✓		
Plateau junction coordinate	Face	f	✓		
Plateau angle	Face	f, v		✓	
Film length	Edge	e		✓	
Curvature radius	Edge	e			✓

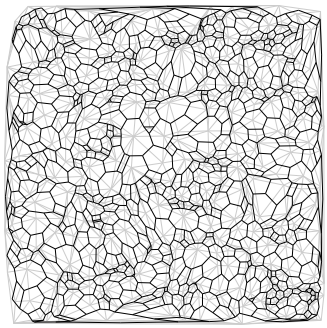
Table 1: A table listing the different features represented in the dual mesh.

3.3 Initialisation of the foam mesh

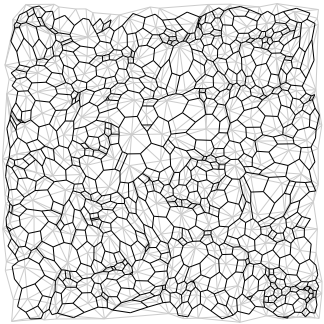
Figure 3.2 shows a very rough initialisation of the foam mesh. It needs further treatment to make a more realistic scenario for an initial two-dimensional foam that we can work with.

3.3.1 Removing sliver faces from dual mesh

Although the Delaunay triangulation generally tends to avoid the generation of sliver faces, Figure 3.2 is a good example of cases, where sliver faces are created on the boundary. These have no physical usage in this case and are removed to ease the equilibration process. Figure 3.3 shows an initial set of dual and primary meshes before and after the removal of the boundary sliver faces with an angle larger than a certain threshold.



(a) Original dual and primary mesh



(b) After removal of sliver faces on boundary with angles above 155°

Figure 3.3: An example of removing sliver faces on the boundary

3.3.2 Equilibration using average positions

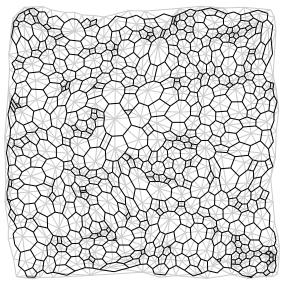
One way to create a bit more balance in the mesh is to use the position of neighbour vertices to move a particular vertex to a new position, which lies more towards the centre of the neighbour vertices. Algorithm 1 presents such an algorithm, where all

vertices in the dual mesh are iterated over, after which their positions are updated at the very end.

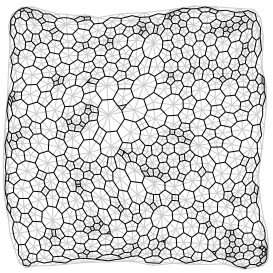
input : dual mesh mesh, iterations for algorithm $\text{iter}_{a,max}$, equilibration rate scale
output: None (since OpenMesh data structure is mutable)

```
1  for i ← 0 to iter_{a,max} do
2      foreach vertex v_0 in mesh do
3          v_tot = (0, 0);
4          foreach vertex v in 1-ring neighbourhood of v_0 do
5              v_tot ← v_tot + v * scale
6          end
7          v_new(v_0) ← v_0 + v_0/numberOfNeighbours
8      end
9      foreach vertex v_0 in mesh do
10         Update position of each vertex, v_0 ← v_new(v_0)
11     end
12 end
```

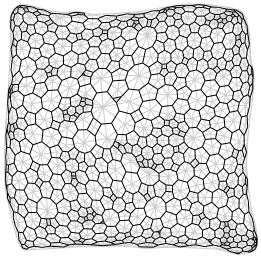
Algorithm 1: Average equilibriate vertices by moving a vertex towards the geometrical centre of the 1-ring neighbourhood vertices



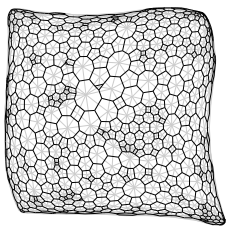
(a) `iter=10, scale=0.05`



(b) `iter=10, scale=0.2`



(c) `iter=100, scale=0.05`



(d) `iter=100, scale=0.2`

Figure 3.4: Equilibration using different parameters of the same random mesh as presented in Figure 3.3b

Figure 3.4 shows the problem when doing many iterations - vertices pile up on the boundary. There is a simple explanation for this. As the one-ring neighbourhood is taken into consideration for each vertex in the mesh, when a vertex happens to

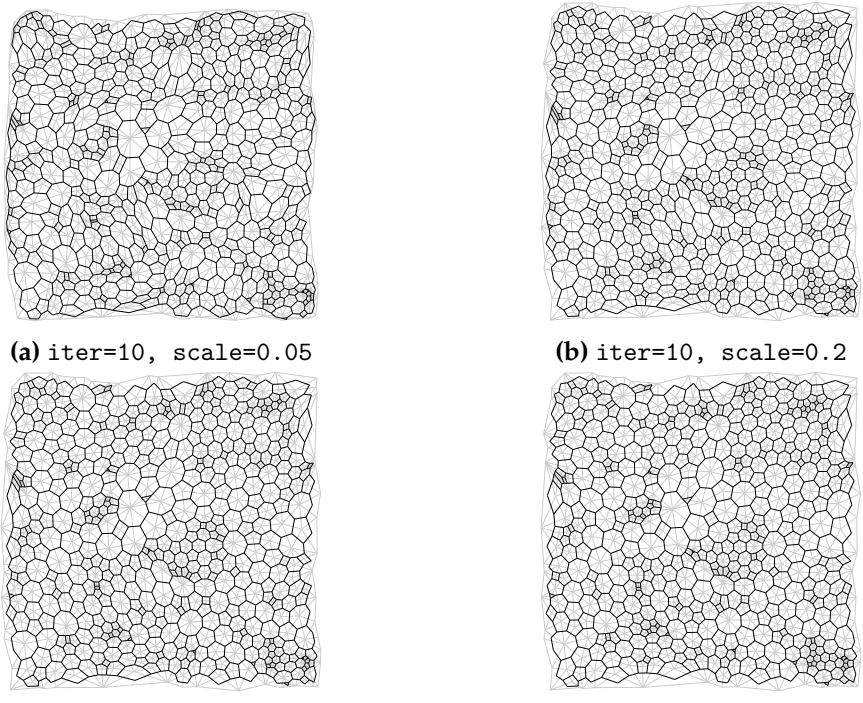
be on the boundary, it only has neighbours to its one side, and is thus inevitably dragged in the approximate direction on the centre of the foam.

3.3.3 Fixing boundary vertices

One can solve this by fixing the boundary vertices to create a sort of frame of the foam. It makes the boundary vertices static and conclusively this part of the equilibration less flexible, but this will be dealt with in a minute.

Since the sliver faces on the boundary have already been removed, a problem is prevented that would otherwise have occurred. Imagining still having the sliver faces. Fixing the boundaries causes these sliver faces to have two out of three vertices fixed. Focusing on one of these sliver faces, the equilibration using the average position in the one-ring neighbourhood will never move these two very distant vertices in the sliver face closer to each other. At the same time, because of it having only neighbours towards the centre of the foam, the one free vertex in the sliver face will be dragged away from the fixed vertices in the sliver face that are sitting stationary on the boundary. By removing the sliver faces beforehand, this problem is averted.

Let us again turn our attention to the average position equilibration. Figure 3.5 shows this equilibration process with the same parameters and the same initial mesh as in Figure 3.4, but with fixed boundary vertices. The positive effects are clearly evident, however, one should not do this infinitely, since we would end up with an initial system in total equilibrium. It is an act of balance.



(a) `iter=10, scale=0.05` **(b)** `iter=10, scale=0.2`

(c) `iter=100, scale=0.05` **(d)** `iter=100, scale=0.2`

Figure 3.5: Same equilibration as in Figure 3.4, but with fixed boundary vertices.

3.3.4 Equilibration using springs between vertices

To deal with the now stationary boundary vertices as well as to introduce a global equilibrium idea to the initial foam mesh, a spring equilibrium routine is introduced in this section. This algorithm will eventually work together with the average position equilibrium algorithm presented in the previous section.

The idea is to introduce small imaginary springs on each edge in the dual mesh. Hooke's law states that

$$F = -kx \ , \tag{3.1}$$

where F is the force a spring is pulling or pushing with, if it is displaced a distance x from its position of equilibrium, $x > 0$ being a displacement away from the source, upon which the spring acts. k is the spring constant describing the stiffness of the spring.

Introducing this idea to our dual mesh, we can experiment with all the vertices in the mesh either pushing or pulling on each other depending on their distance and a predefined global equilibrium length l_0. Thus, the vertices will not *only* be dependent on their immediate neighbours, but also be affected of the predefined equilibrium length. This equilibrium length could, for instance, be defined as the mean or median of all the lengths of the edges in the dual mesh.

Iterating through all vertices in the mesh, for each vertex at time t with coordinate v_0^t, one can calculate the sum of forces to the neighbouring vertices. Looking at (3.1), the displacement x can be written as the difference between the equilibrium length and the distance between the vertex and the neighbour vertex i. The resulting magnitude of the force then looks like

$$F_i^t = -k(l_0 - |v_i^t v_0^t|) \tag{3.2}$$

The resulting force on the vertex in vector form is the sum of the forces acting on the vector

$$\boldsymbol{F} = \sum_i F_i^t \cdot \hat{\mathbf{F}}_\mathbf{i}^\mathbf{t} \ , \tag{3.3}$$

where $\hat{\mathbf{F}}_\mathbf{i}^\mathbf{t}$ is the unit vector going from v_0^t to v_i^t.

Using Newton's second law of motion, one can approximate the position of v_0^t at time $t = t_0 + \Delta t$. Assuming Δt to be small and all vertices to be of mass $m = 1$, one can write up the new position as

$$v_0^{t+\Delta t} = v_0^t + \boldsymbol{F}\Delta t \tag{3.4}$$

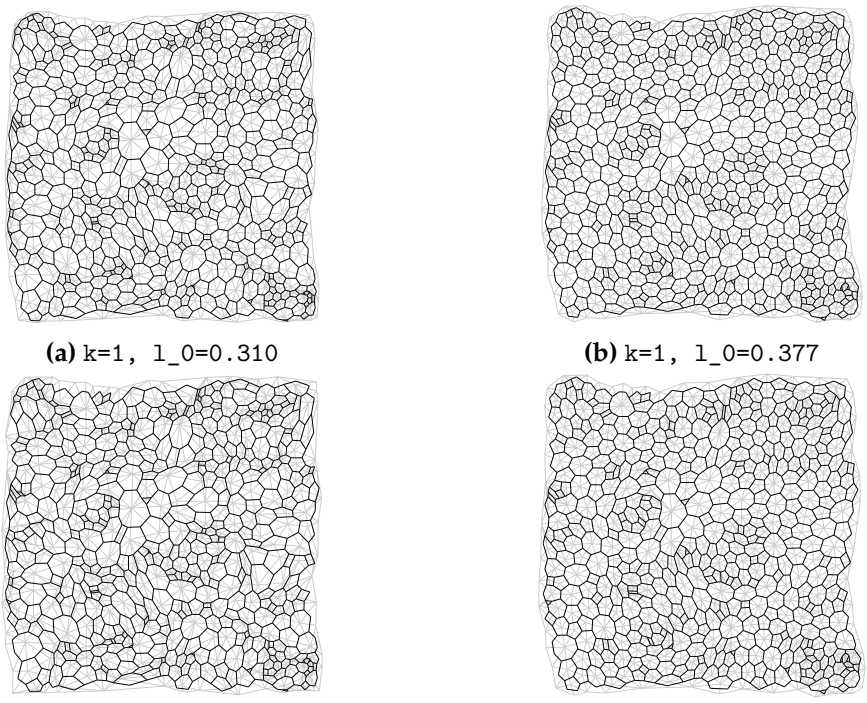
input : dual mesh mesh, iterations for algorithm $\text{iter}_{s,max}$, spring constant k, spring equilibration length l_0, time step for spring equilibration Δt
output: None (since OpenMesh data structure is mutable)

```
for i ← 0 to iter_{s,max} do
    foreach vertex v_0 in mesh do
        F = (0, 0);
        foreach vertex v in 1-ring neighbourhood of v_0 do
            d ← |v_0 v| F_i ← −k ∗ (l_0 − d);
            F_hat ← (v − v_0)/d;
            F ← F + F_hat ∗ F_i
        end
        v_new(v_0) ← v_0 + F ∗ Δt
    end
    foreach vertex v_0 in mesh do
        Update position of each vertex, v_0 ← v_new(v_0)
    end
end
```

Algorithm 2: Equilibriate mesh using a spring system with imaginary springs with a certain equilibrium length instead of edges between vertices.

Algorithm 2 outlines the implementation of this idea.

Figure 3.6 shows this equilibration process for different parameters.



(a) `k=1, l_0=0.310` **(b)** `k=1, l_0=0.377`

(c) `k=10, l_0=0.310` **(d)** `k=10, l_0=0.377`

Figure 3.6: The rest length is estimated as being around the initial average distances between the vertices in the mesh. All examples have done 10 concecutive iterations with $\Delta t = 0.005$. Note, that Δt and k, although having different physical meanings, in the end could be turned into one variable, since they are both constant multipliers in all terms in (3.2).

3.3.5 Combining equilibration algorithms

To arrive at a proper point for initialising the foam, combining these equilibrium methods was tried. Since there are no strict rules for when our foam is properly initialised, it is a question of visually reaching a satisfying result. The Delaunay triangulation gives us a nice dual mesh to start with [Kermode and Weaire, 1990], but it uses random points each time. Thus, the whole equilibration step is used to make the initialisation more robust. The initial foam, of course, shall not be in total equilibrium obeying Plateau's rules. Then nothing would happen during the simulation. An amount of disorder and differences in area and number of neighbours for the bubbles is desired.

Algorithm 3 shows the combination of the equilibrium processes.

A re-triangulation is performed after the two equilibration steps. This is to prevent a violation of the mesh-structure with edges crossing each other and vertices moving into other faces etc. Sliver faces resulting from the re-triangulation are removed as well as *shark fin faces*. These are faces on the boundary with two boundary facing edges. The reason for removing these will be justified in Section 3.3.6.

Table 2 shows a list of the recommended parameters for Algorithm 3. Remember, this is not exact values based on statistical tests or strict parameter tests. Both the

input : dual mesh mesh, average equilibration force scale, number of equilibration iterations iter_{max}, iterations of average position and spring algorithms $\text{iter}_{a,max}$ and $\text{iter}_{s,max}$, spring constant k, spring equilibration length l_0, time step for spring equilibration Δt
output: None (since OpenMesh data structure is mutable)

```
RemoveSliverFaces(mesh);
for i ← 0 to iter_max do
    AverageEquilibriateVertices(mesh, scale, iter_a,max);
    SpringEquilibriateVertices(mesh, k, l_0, iter_s,max);
    SpringEquilibriateVertices(mesh, k, l_0, iter_s,max);
    Retriangulate(mesh);
    RemoveSliverFaces(mesh);
    RemoveSharkfinFaces(mesh);
end
```

Algorithm 3: Combining equilibration algorithms

non-equilibrated mesh and the equilibrated mesh are valid meshes, but the aim is to get a controlled mesh without weird bubbles or features. The whole process of finding suitable parameters is an act of smoothing the foam without bringing it too close to a state of total equilibrium and while still keeping a degree of variance. Of course, the initialisation and equilibration process could be seen as an optimisation problem of its own optimising for variance, smoothness etc., but it is beyond the scope of this project to focus too much on the initialisation, once a reasonable initialisation meeting the above criteria is at hand.

Parameter	Meaning of parameter	Equilibration algorithm	Rec. value
`scale`	size of step	`average_eq_ver`	0.1
`iter_av`	number of iterations	`average_eq_ver`	10
`k`	spring constant	`spring_eq_ver`	0.5
`l_0`	equilibrium spring length	`spring_eq_ver`	0.36
`delta_t`	time step	`spring_eq_ver`	0.005
`iter_sp`	number of iterations	`spring_eq_ver`	10
`iter_tot`	number of iterations	`combine_eq_ver`	1

Table 2: A table listing the recommended parameter values for the equilibration algorithm presented in this thesis. The parameters listed are the ones used to arrive at the situation shown in Figure 3.9c.

3.3.6 Dealing with bubbles on the boundary

So far, the bubbles on the boundary have not been paid attention to. All midpoints of the faces have simply been connected to midpoints of neighbouring faces. Weaire, both in [Weaire and Hutzler, 2001] and [Kermode and Weaire, 1990] used periodic boundary conditions. This caused the foam to have no problems on the boundary - one could simply think of the foam as a small foam sample taken from

an infinite foam. [Vedel-Larsen, 2010] altered [Kelager, 2009]'s *ghost bubble* and introduced the *world bubble* - an addition of faces holding information of pressure in the world surrounding the foam. This world pressure was considered constant at all times. The way he did this was to add *shark fin faces* as introduced in the previous section to all faces with one boundary edge. The vertices on the tips of these added faces would then all hold the world pressure value using Dirichlet boundary conditions. The boundary bubbles in the foam would then be connected from the midpoints of these added shark fin faces around the boundary.

However, this thesis presents a slightly different approach to dealing with the boundary. Figure 3.7 shows a section of an equilibrated initial foam. There are no shark fin faces, since any potential shark fin faces are removed. Had there been such faces, the visualisation would have left us with unconnected tails of bubble films not connected to anything, since the shark fin faces would have only had one neighbouring face. But Figure 3.7 actually has all, we need. All boundary vertices in the dual mesh are not bubble vertices, i.e. they do not belong to any foam bubbles, and they could thus easily be used as "world" vertices instead, holding world pressure values. But the foam still has some weird edgy world facing boundary bubbles.

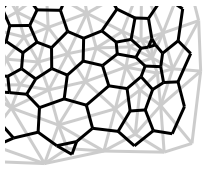


Figure 3.7: A section of the final equilibrated initial foam.

The visualisation routine has to know, we are on the boundary and perceive these edgy boundary bubble films as one bubble film, in some cases spanning multiple faces as in the Figure 3.7 in the bottom right corner. These world facing boundary faces with two boundary vertices do not hold any information other than that they connect the two faces holding each end of the bubble film. The latter are faces with only one boundary vertex and in general characterise faces belonging to boundary junctions in the foam. Implementing this, we arrive at the situation shown in Figure 3.8.

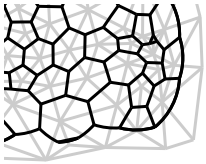
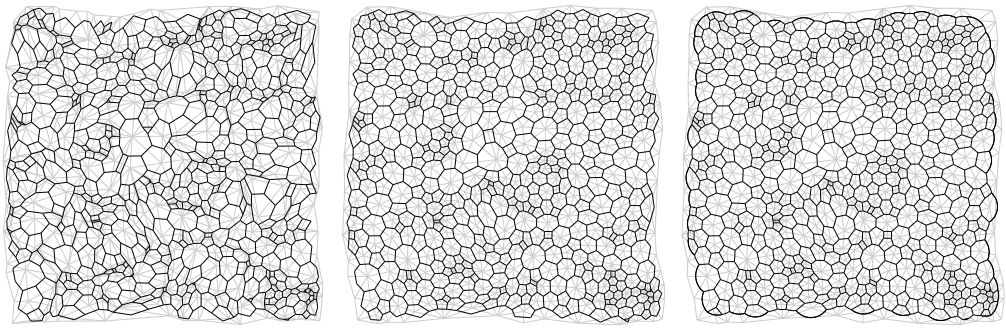


Figure 3.8: A section of the final equilibrated initial foam with the boundary haven been taken care of. Further more, a world pressure outside the foam has been added, slightly lower than the pressure inside the foam. Thus, the bubble films on the boundary bend outwards with a curvature radius defined by the law of Laplace-Young, see (2.3).

Zooming out again, the main steps of the equilibration process are shown in Figure 3.9.



(a) Before equilibration **(b)** After equilibration **(c)** Boundary treated

Figure 3.9: The process of equilibration and boundary treatment. Tuning the parameters for the algorithms in the combined equilibriate algorithm, the used parameters to arrive Figure 3.9c are *average_equilibriate_vertices(mesh, iterations=10, scale=0.1)*, *spring_equilibriate_vertices(mesh, iterations=10, k=0.5, l_0=0.36, delta_t=0.005)*.

3.3.7 Sampling from another distribution

The rectangular initialisation is just one way to initialise. One can easily initialise the foams drawing random numbers from other samples. Figure 3.10 shows such a sample after equilibration.

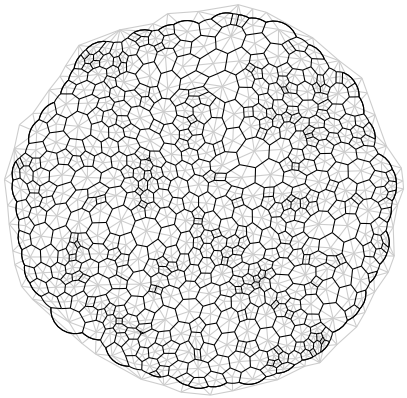


Figure 3.10: Circular initialisation of foam.

The pseudo-code for generating uniformly distributed random numbers from a circular distribution is shown in Algorithm 4.

However, we will proceed with the rectangular case, since it does not have as nice symmetry as the circular one, making us able to study, how the behaviour is on on the corners of the foam and how it affects the simulation.

3.3.8 Special cases

This type of handling the boundary can also handle cases of very few bubbles. [Vedel-Larsen, 2010] had a lower minimum of three bubbles - the dual mesh could

input : Circle center (x_{center}, y_{center}) and radius rad from which the random point will be drawn
output: Random point (x, y)

```
1 u, θ = [(Rand(0, 1) + Rand(0, 1)) ∗ rad, 2 ∗ π ∗ Rand(0, 1)];
2 if u < rad then
3 |   r ← u
4 else
5 |   r ← 2 ∗ r − u
6 end
7 x = x_center + r ∗ cos(θ);
8 y = y_center + r ∗ sin(θ);
```

Algorithm 4: Random sampling of a single point from a uniform circular distribution.

not represent any fewer bubbles. The method presented above can also handle two bubble cases as is shown in Figure 3.11

(a) Small arc **(b)** Large arc

Figure 3.11: 2-bubble case. Two types of arcs with the same curvature radius and the same junction positions. Only the small arc will be used in the initialisation, as did [Kermode and Weaire, 1990].

Figure 3.11 also shows two kinds of bubbles with the same curvature radius. [Kermode and Weaire, 1990] and [Herdtle and Aref, 1992] consider only the small arcs in their work. We shall also only consider the small arcs. This is also reasonable, when being within the foam - otherwise, if one chose the large arcs, border films would cross, making the foam non-valid. It is, however, on the border, where such a situation would not be unrealistic.

Consider a 2-bubble foam as a circle with a straight bubble film down the middle. The angles between the film between the bubbles and the border arcs will all be π as shown in Figure 3.12. This also means that the curvature radius would be equal half the distance between the two Plateau junctions equal to $r = \frac{2\gamma}{\Delta p} = \frac{2\gamma}{p_{cell} - p_{world}}$ according to the law of Laplace-Young. As is also described in [Weaire and Hutzler, 2001], when approaching this state on the boundary of a larger foam corresponding to one half of the foam in Figure 3.12, the bubble could tend to "pop", moving from an angle of $\theta < \pi$ across the situation shown in the figure with $\theta = 90°$ to a large arc with $\theta > \pi$.

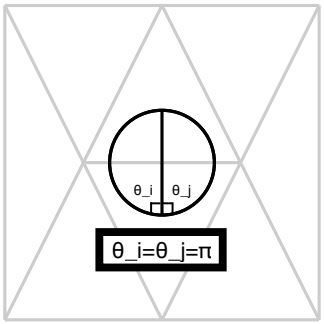
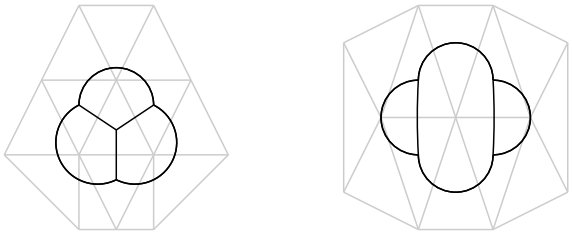


Figure 3.12: A situation of a 2-bubble foam, where the small and the large arc representations are equal.

Figure 3.13 shows other odd shapes of cases with a few number of bubbles, which this kind of mesh representation is also able to represent.



(a) 3 bubbles **(b)** 3-bubble chain

Figure 3.13: Two cases of few bubbles, which the visualisation routine and the dual mesh are able to represent without any change of interpretation.

To show that different dual meshes can result in the same primary mesh, Figure 3.14 shows the same foam, but with Figure 3.14b having additional boundary faces, which are do not hold any importance to the primary mesh. Therefore, a reduction of the number of such faces makes sense, i.e. when having a case like the one shown in Figure 3.14b, one could without loss of information reduce the number of faces to the amount to the situation shown in Figure 3.14a. This removal of redundant faces would speed up the visualisation routine and free memory.

(a) Simplest dual mesh **(b)** Additional faces in dual mesh

Figure 3.14: 4 bubbles forming an H-shape border with different dual meshes. Figure 3.14b has a different dual mesh configuration than 3.14a, since it has additional boundary faces. Remember, these faces don't play any role in the conversion to the primary mesh, which is seen by the identical primary meshes.

3.3.9 The dual mesh in the simulation

Once the foam is allowed to coarsen, the junctions and films move. Whereas the primary mesh was initially created on the basis of the dual mesh, the roles switch from then on. During the simulation, the dual mesh is constantly adjusted to adopt to the evolving foam. The vertex positions on the boundary are not adjusted and remain stationary, but a sweep through the foam cleans the boundary by joining adjacent world-facing faces, if topological operations on the boundary result in such faces to occur. Thus, there will never be shark fin faces on the boundary, and the dual mesh representation on the boundary remains consistent.

The positions of the non-boundary vertices inside the dual mesh are of no importance, once the simulation is started. Only the connectivity information is important, as it tells about connectivities in the foam. For the sake of convenience when plotting and debugging, the vertices in the dual mesh are constantly adjusted to be in the centre of the cells, they represented.

3.4 Pressure initialisation

The pressure is initialised as uniform within the foam, and lower outside the foam. This uniform pressure will quickly change, as topological changes and moving equilibration processes will cause diffusion to start, once the simulation is started. We might have equal pressure, but recall that we do not meet Plateau's first rule of equilibrium, hence the foam will move towards equilibrium by changing junction coordinates and cell pressures.

One could also consider a random pressure initialisation or scale the pressure with the number of neighbours per bubble. Bubbles with fewer neighbours tend to have higher pressure than large bubbles with many neighbours, as we learned from von Neumann's law. But generally, after only one iteration of a uniform pressured foam, the pressure will have changed to match the cells' geometrics.

Not much is written about how to initialise the pressure, but digging in to the appended code in [Kermode and Weaire, 1990] reveals that they initialise with equal pressure in all cells. They use relative pressure values, and set the pressure as 0

inside the foam. Of course, the pressure is not 0 Pa inside the foam, but since we only use pressure *differences* in our calculations, any offset does not play any role. Another good reason to initialise the pressure to be equal inside the foam is that we eliminate any additional bias in the initialisation. By adding a constant pressure, we don't construct any pressure bias, and all cells are equal off from the beginning.

3.5 T1 and T2 operations

Two crucial operations in foam evolution are the T1- and T2-processes.

The T1-process occurs if two Plateau junctions get too close to each other, as was the case in the argumentation in Figure 2.3. I.e., if the length of the bubble film between them converged to zero, this would result in a junction with 4 incoming edges, which would violate Plateau's laws, and this instability is therefore never seen, since an instantaneous T1-process would happen in a case. In the discrete case of the implementation for the simulation, [Kermode and Weaire, 1990] uses the criterion for a T1-process, that such a process would occur, if

$$\bar{R}_i \cdot \bar{R}_i = ||\bar{R}_i|| < \bar{R}_i \cdot \bar{\Delta} \tag{3.5}$$

where R_i is the vector between the vertex in question, v_0, and its closest neighbour vertex, v_i. $\bar{\Delta}$ is the change of the position of v_0 in the next time step, and thus the dot product is the length of the movement towards v_i. If this exceeds the distance between them, a T1-process is performed. Figure 3.15 shows such a topological change.

(a) Before T1-process **(b)** After T1-process

Figure 3.15: Illustration of the dual mesh edge flip - the T1-process. As before, the black lines illustrate the primary mesh, whilst the grey lines represent the dual mesh.

This thesis, however, uses a minimum film length threshold defined by the user to detect, when a T1 must be performed. An approach also used by previous authors such as [Brochu and Bridson, 2009] and [Da et al., 2016].

To justify this choice, let me draw the reader's attention to Figure 3.16, which shows two situations of equal surface energies. With the corners fixed, both the red and the green films form the minimum spanning configuration of two junctions with a connecting film [Isenberg, 1978], and there is thus no preference for the one or the other configuration. One can with a small perturbation cause the green configuration to flip to the red situation and vice versa. The argument is, that moving below the threshold h would increase the energy, and thus a flip is performed, since the configuration would never move to a position with higher energy. Following

this reasoning, a user-defined threshold makes sense in a physical sense. Furthermore, one can perform the topological operations before the update cycle on the mesh instead of calculating the movements of the junction, perform the topological processes and then recalculate the movements for the update cycle.

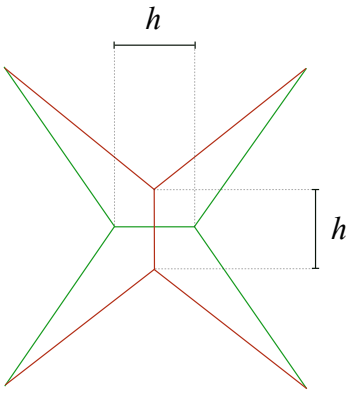


Figure 3.16: Two situations of equal surface energy. There is nu preference for either the red or the green situation. The distance between the junctions is h in both situations.

A T2-process, on the other hand, is the removal of a small three-sided bubble as is shown in Figure 3.17. This is a result of the diffusion and coarsening taking place. As was the case with the implementation of the T1-process, a T2-process will take place, whenever a the area of a bubble gets below a user-defined threshold.

(a) Before T2-process

(b) After T2-process

Figure 3.17: Illustration of the dual mesh face collapse - the T2-process. In the dual mesh, three faces are deleted, and one is instead.

3.6 Physical descriptors of the foam

In the process of simulating the coarsening of the foam, three main physical quantities are important to associate with the foam. These following quantities are also listed in Table 1 and show, where each quantity is stored in the mesh.

3.6.1 Bubble areas

As established, each cell (or bubble. The terms can be used interchangeably) in the foam is associated with a vertex in the dual mesh. Iterating through all such vertices in the foam, one can calculate the area of the cell and attach it to that very vertex as a property (a technical introduction of properties is to be seen in Section 4.2.2).

At each vertex, one has to divide the cell area into sections, calculate the area of these sections and sum them. Figure 3.18 shows an example of a cell. The bubble is enclosed by the black line in the graph. To calculate the area of a segment, of which there are five in this particular example, one calculates the area of one of the red triangles, then one adds the area of the shaded red region on top of the triangle, where the bubble film bends outwards towards the neighbour cell or subtracts the area of the yellowish area, where the bubble bends inwards, if that is the case. Eventually, all segment areas are summed.

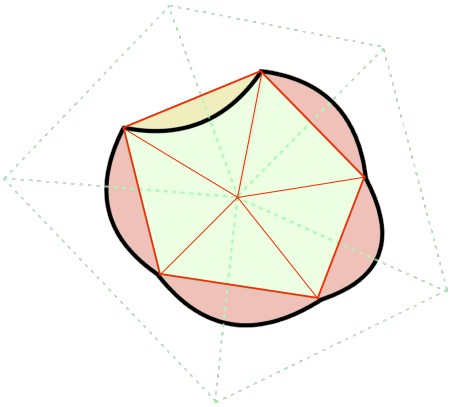


Figure 3.18: An example of a bubble. The dashed green lines show the dual mesh, the solid black lines are the bubble films. The red solid lines mark the areas that are calculated and summed, the red shaded areas the areas that are added and the yellowish are the area that is subtracted.

To calculate the the area of each segment, one needs the three coordinates that form the triangle in question. One of these coordinates is the vertex coordinate, whereas the two others are the coordinates for the Plateau junctions.

Zooming in on one of these segment as in Figure 3.19, one can write the area of this segment as the sum of the green area $A_{triangle}$ and the red area A_{arc}. The area of the red section is

$$A_{arc} = sgn(\Delta p)\left(\frac{\beta}{2\pi}\pi R^2 - \frac{1}{2}R^2\sin\beta\right) = sgn(\Delta p)\frac{1}{2}R^2\left(\beta - \sin\beta\right) \ , \qquad (3.6)$$

where $sgn(\Delta p)$ is the sign of the pressure difference to the neighbouring cell, a positive Δp being a lower pressure in the neighbour cell. The total area of the

segment is given by

$$A_s = A_{triangle} + A_{arc} = \frac{1}{2}|\det(\overline{\boldsymbol{v}_0\boldsymbol{x}_i}, \overline{\boldsymbol{v}_0\boldsymbol{x}_{i+1}})| + sgn(\Delta p)\frac{1}{2}R^2\left(\beta - \sin\beta\right)$$
$$= \frac{1}{2}\left(|\det(\overline{\boldsymbol{v}_0\boldsymbol{x}_i}, \overline{\boldsymbol{v}_0\boldsymbol{x}_{i+1}})| + sgn(\Delta p)R^2\left(\beta - \sin\beta\right)\right) \tag{3.7}$$

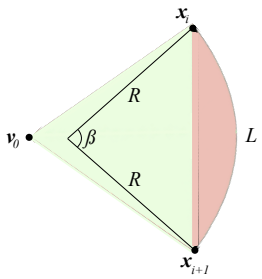


Figure 3.19: One segment of the bubble cell. $\boldsymbol{v}_0$ is the vertex coordinate of the dual mesh belonging to the bubble. R is the curvature radius of the arc, β the angle between the two lines spanning the arc with length R and L the length of the bubble film.

Extending to the whole bubble with N neighbours, the area is given by

$$\boxed{A_{bubble} = \sum_i^N A_{s,i} = \sum_i^N \left[\frac{1}{2}\left(|\det(\overline{\boldsymbol{v}_0\boldsymbol{x}_i}, \overline{\boldsymbol{v}_0\boldsymbol{x}_{i+1}})| + R_i^2\left(\beta_i + sgn(\Delta p_i)\sin\beta_i\right)\right)\right]}$$

3.6.2 Film lengths

The total film length of a bubble L_{bubble} is given by the sum of the individual film lengths to to each neighbour, which are easily calculated by the fraction of the arc to the imaginary circle with curvature radius R

$$L = \frac{\beta}{2\pi} \cdot 2\pi R = \beta R \tag{3.8}$$

and thus

$$\boxed{L_{bubble} = \sum_i^N L_i = \sum_i^N \beta_i R_i}$$

The film length is used in order to calculate the surface energy of the films.

3.6.3 Plateau angles

In order to calculate the Plateau angles in the foam, one has to consider three junction coordinates for each angle, as is shown by $\boldsymbol{x}_j$, $\boldsymbol{x}_i$ and $\boldsymbol{x}_k$ in Figure 3.20.

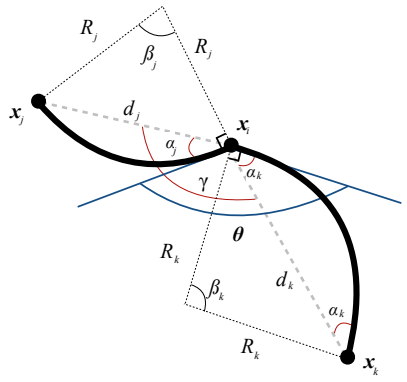


Figure 3.20: An example of a Plateau junction, where θ is the angle between the films as well as the angle between dark blue tangent lines at $\boldsymbol{x}_i$.

To calculate the angle between the bubble films θ, one has to know γ, α_j and α_k in order to find θ (in this case, γ is of course *not* the surface tension, but just a symbol for an angle). θ can be written as a sum of the angles, where in this particular case, α_j would be negative and α_k positive because of the direction of the films' arcs

$$\theta = \gamma + \alpha_j + \alpha_k \, . \tag{3.9}$$

How do we find these three angles? γ is the angle between the two dashed grey lines. Using the inverse cosine, one can find this angle. One needs to check for the case of the angle being larger than 180°. The cross product of two 3-dimensional vectors in the (x, y)-plane can reveal this, if the order is known, which it is, since the mesh data structure is a half-edge data structure iteration in the same direction round the clock each time. Restricting ourselves to the two-dimensional plane, the determinant of the concatenation of the two column vectors does the same job.

$$\gamma = \begin{cases} \cos^{-1}\left(\frac{\overline{\boldsymbol{x}_i\boldsymbol{x}_j}\cdot\overline{\boldsymbol{x}_i\boldsymbol{x}_k}}{||\overline{\boldsymbol{x}_i\boldsymbol{x}_j}||\,||\overline{\boldsymbol{x}_i\boldsymbol{x}_k}||}\right), & \text{if } \det(\overline{\boldsymbol{x}_i\boldsymbol{x}_j}, \overline{\boldsymbol{x}_i\boldsymbol{x}_k}) > 0 \\ 2\pi - \cos^{-1}\left(\frac{\overline{\boldsymbol{x}_i\boldsymbol{x}_j}\cdot\overline{\boldsymbol{x}_i\boldsymbol{x}_k}}{||\overline{\boldsymbol{x}_i\boldsymbol{x}_j}||\,||\overline{\boldsymbol{x}_i\boldsymbol{x}_k}||}\right), & \text{if } \det(\overline{\boldsymbol{x}_i\boldsymbol{x}_j}, \overline{\boldsymbol{x}_i\boldsymbol{x}_k}) < 0 \end{cases} \tag{3.10}$$

To find the α's, let us focus on the lower right bubble film with indices k. The isosceles triangle spanned by the dashed lines has an angle sum of π. The sum is

$$\begin{aligned} \pi &= \beta_k + 2(\frac{\pi}{2} - \alpha_k) \\ &\Downarrow \\ \alpha_k &= \frac{\beta_k}{2} \end{aligned} \tag{3.11}$$

since the dashed lines length R_k and the navy blue tangent line are perpendicular. Hence, to calculate α_k, we need to know β_k. Knowing three sides, one can use the law of cosines to find the angle. Using the identity that $2\sin^{-1}(x) = \cos^{-1}(1-2x^2)$, the expression can be simplified:

$$\beta_k = \cos^{-1}\left(\frac{2R_k^2 - d_k^2}{2R_k^2}\right) = \cos^{-1}\left(1 - 2\left(\frac{d_k}{2R_k}\right)^2\right) = 2\sin^{-1}\left(\frac{d_k}{2R_k}\right) \tag{3.12}$$

Inserting (3.12) into (3.11) and taking the opposite situation with an inward bending film into consideration, too, yields

$$\alpha_k = sgn(\Delta p_k)\sin^{-1}\left(\frac{d_k}{2R_k}\right) . \tag{3.13}$$

Realising that $\|\overline{\boldsymbol{x}_i\boldsymbol{x}_j}\| = d_j$ and that $\|\overline{\boldsymbol{x}_i\boldsymbol{x}_k}\| = d_k$ and plugging (3.13) and (3.10) into (3.9) yields

$$\theta = \begin{cases} \cos^{-1}\left(\frac{\overline{\boldsymbol{x}_i\boldsymbol{x}_j}\cdot\overline{\boldsymbol{x}_i\boldsymbol{x}_k}}{d_j d_k}\right) + sgn(\Delta p_k)\sin^{-1}\left(\frac{d_k}{2R_k}\right) + sgn(\Delta p_j)\sin^{-1}\left(\frac{d_j}{2R_j}\right), & \text{if } \det(\overline{\boldsymbol{x}_i\boldsymbol{x}_j}, \overline{\boldsymbol{x}_i\boldsymbol{x}_k}) > 0 \\ \left[2\pi - \cos^{-1}\left(\frac{\overline{\boldsymbol{x}_i\boldsymbol{x}_j}\cdot\overline{\boldsymbol{x}_i\boldsymbol{x}_k}}{d_j d_k}\right)\right] + sgn(\Delta p_k)\sin^{-1}\left(\frac{d_k}{2R_k}\right) + sgn(\Delta p_j)\sin^{-1}\left(\frac{d_j}{2R_j}\right), & \text{if } \det(\overline{\boldsymbol{x}_i\boldsymbol{x}_j}, \overline{\boldsymbol{x}_i\boldsymbol{x}_k}) < 0 \end{cases} \tag{3.14}$$

4 Software

4.1 Mesh libraries

During the initial phases of the project, the aim was to find a usable mesh library for Python. There were different ones to choose from, all with their advantages and disadvantages. None of them was the obvious choice, since they all lacked in some areas.

Regarding the installation itself, one would be happy, if the packages could be downloaded and installed using comfortable package managers such as `pip`. However, since mesh libraries aren't amongst the best supported packages in Python, and most of them are still to be improved open source projects, the installation often is a bit more tedious than that. And depending on the set-up of one's computer, it can take quite a while to get the libraries up and running. Bindings have to be installed correctly, and the documentation on how to do so was in many cases not particularly comprehensive.

	PyMesh	OpenMesh	CGAL	MatplotLib
Easy to install				✓
Easy to use				✓
Good documentation	✓	(✓)		✓
Delaunay triangulation	✓	✓	✓	✓
Edge flip operation		✓	✓	
Face collapse operation		✓	✓	

Table 3: Overview over the investigated mesh libraries for Python

Table 3 sums up the subjective assessment of the abilities of the different libraries in this research, as they appeared when searching for a mesh library.

Initiallly, I started to try out PyMesh, but it quickly turned out that it did not meet the requirements for the type of mesh, I needed for the project. Basically, three operations are essential in this project: Delaunay triangulation to create the triangle mesh from an initial set of vertices in the dual mesh, edge-flip operations as well as face-collapse operations. PyMesh only met one of these requirements, and hence, the search for another mesh library was on.

After different attempts and discussions with my supervisor, we agreed that it would be a good idea to see, if I could get OpenMesh up and running. OpenMesh is widely used when coding meshes in C++, but it has Python bindings as well. Thus, the documentation for C++ is rather good and can, with a small act of translation, be used when implementing in Python as well. The functionality is comprehensive too, so I gave it a try. It proved to be rather difficult to install and get to work, but it eventually worked out, and it was up and running. To read an installation guide, I created after my experiences with installing OpenMesh, I will refer to Appendix A.

4.2 Using OpenMesh

Once installed, the features of OpenMesh can be used. This section gives a brief introduction to some of the main features encountered in OpenMesh with respect to building the foam mesh.

4.2.1 Building a simple mesh

One of the great things about libraries such as OpenMesh is that one can easily iterate over the vertices, edges and faces of the mesh. These mesh elements are each associated with different variables as shown in Table 1. It is therefore important that these variables are easily accessible. The mesh itself is stored as a so-called TriMesh-object containing all the necessary information. From here on, in code examples, `vh` will be used as a name for vertex handles, `fh` for face handles, `eh` for edge handles and `heh` for halfedge handles. The mesh is created as an empty mesh, and vertices and faces are then added to the mesh:

```
>>> mesh = TriMesh()
>>> vh0 = mesh.add_vertex(TriMesh.Point(0, 0, 0))
>>> vh1 = mesh.add_vertex(TriMesh.Point(1, 0, 0))
>>> vh2 = mesh.add_vertex(TriMesh.Point(0, 1, 0))
>>> vh3 = mesh.add_vertex(TriMesh.Point(1, 1, 0))
>>> fh0 = mesh.add_face(vh0, vh1, vh2)
>>> fh1 = mesh.add_face(vh1, vh3, vh2)
```

where the vertices added in the `add_face`-command are OpenMesh vertex objects. Since OpenMesh is based on a halfedge data structure, the faces must be added adding vertices in the correct order, i.e. in a clockwise direction in this case. This particular mesh is shown in Figure 4.1.

Figure 4.1: Simple example of a mesh consisting of four vertices and two faces.

One can iterate through these vertices in the mesh and print their current identifiers by typing

```
>>> for vh in mesh.vertices():
...     print vh
```

and similarly for faces by iterating through `mesh.faces()`.

Having added the desired number of vertices and faces, the edges are given implicitly.

Regarding circulators, one can also easily visit the neighbours of one of the mesh elements. Say, one for instance wants to iterate over faces surrounding a vertex

```
>>> for fh in mesh.vf(vh0):
...     print fh
```

Other circulators are shown on the webpage of OpenMesh at [RWTH Aachen, c].

4.2.2 Assigning properties

As discussed, one can add values to the mesh associated with vertices, faces and edges. As an example, one can add pressure values to each cell in the foam. One can add such properties to the vertices in the mesh. In OpenMesh for Python, in the case of adding a pressure value equal to 1, it is done as follows:

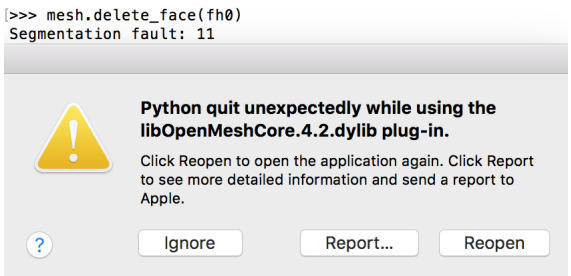
```
>>> pressure = 1
... pressure_ph = VPropHandle()        # Create vertex property handle
... mesh.add_property(pressure_prop_handle, "pressure")
... for vh in mesh.vertices():
...     mesh.set_property(pressure_ph, vh, pressure)
```

4.2.3 Deleting mesh elements

Having built the initial mesh, one of the most important actions when evolving a foam is of course the ability to be able to delete mesh elements, as the evolution of the. In OpenMesh, this seems straight forward. Having for instance a face handle `fh_0` corresponding to a face on the boundary, there is a command

```
>>> mesh.delete_face(fh0)
```

However, this will not go well. Why not? At first, I thought it could have to do with my Python set-up on my laptop. But running simple test programs on other laptops caused the same problem. Python would crash with the error message:



However, when looking through the unit tests of the source code, it turned out that one has to request the attribute before deleting it, as it is also described in the C++ documentation for OpenMesh [RWTH Aachen, b]. For the above example to work, one needs to request the face status before deleting it as well as the vertex status

and edge status, since the deletion in this example results in an isolated vertex with belonging edges:

```
>>> mesh.request_face_status()
>>> mesh.request_vertex_status()
>>> mesh.request_edge_status()
>>> mesh.delete_face(fh0)
>>> mesh.garbage_collection()
```

The garbage collection method is called at the end to delete the face from memory. Otherwise, it would only be flagged as deleted. This deletion changes the mesh in Figure 4.1 to the mesh visualised in Figure 4.2.

Figure 4.2: Simple example of a face deletion in a two faced mesh with four vertices resulting in a one faced mesh with three vertices.

Two main operations in this project are the T1 and T2 operations, i.e. edge flips and face collapses as described in Section 3.5. Looking at Table 1, a bubble collapse corresponds to removing a vertex in the dual mesh. In OpenMesh, such an operation is performed by deleting an outgoing halfedge from that vertex. Only one deletion is necessary. This halfedge deletion then automatically results in the deletion of three faces and the insertion of one face, the deletion of the belonging edges and halfedges as well as the deletion of the middle vertex. Bearing the above in mind, a face collapse is performed as

```
>>> mesh.request_face_status()
>>> mesh.request_vertex_status()
>>> mesh.request_halfedge_status()
>>> mesh.request_edge_status()

>>> for heh in mesh.voh(vh0)
...     if mesh.is_collapse_ok(heh):
...         mesh.collapse(heh)
...         break
... mesh.garbage_collection()
```

Since it does not delete any faces and only changes the adjacent faces' connectivities, an edge flip does not need any requests to be asked, and can be performed straight away

```
>>> mesh.flip(eh)
```

4.3 Visualising the mesh

One important aspect of a physical simulation as the one presented in this thesis is the visualisation of the physical medium. The visualisation serves first of all as a key component in the debugging process of the code. Once the visualisation works smoothly, one can use it as a template for the following work on the numerical solver and the physical correctness.

After different considerations, a visualisation using the 'SVG'-format (Scalable Vector Graphics, XML-based) was chosen. There were several reasons for this. First of all, this format is highly customisable in terms of creating graphics that are not part of pre-existing libraries. Then there are different SVG-libraries for Python as well. Another nice thing is that the SVG-files are seamlessly created without any sudden pop-up windows. In the end, the `svgwrite`-library was chosen. It can handle lines, circles, arcs, different colours and size etc. Unlike the mesh-libraries, the library can be easily installed by a single command line in the terminal using `pip`

4.4 Plotting and fitting

In order to plot and fit data to validate the simulation, I have used the Matplotlib library for Python as well as Root, a library for statistical analysis and data processing, created at CERN. Root also has a Python interface, but needs to be installed separately like OpenMesh and cannot be installed simply by using the `pip`-command.

5 Simulation

Having established the physical equations that describe the foam, the implementation of the foam structure using a mesh data structure as well as describing the necessary software, it is now time to move on to the simulation itself. Up until now, we have been stationary at $t = 0$, and it is time to start the time ticking.

But before we can do so, we need to establish, how we will approach the simulation. From a classical physicist's point of view, one would like to be able to write up the time dependent differential equations that govern the foam (including von Neumann's law of diffusion, Laplace-Young's law), solve these analytically and eventually get a solution to the foam at any point in time. Unfortunately, it is not so straight forward. We are dealing with an N-cell foam, and trying to sit down with a pen to solve this would make one have to spend much more than half a year and probably not come up with a satisfying result at the end anyway.

Instead, we will simulate the foam and build a numerical model, which we can use in order to evolve the system using some suitable numerical method.

Consider the initial foam. This is a foam, which is not in equilibrium. In other words, the foam will want to rearrange itself to a state of lower energy, driven by diffusion and the movement of the junctions in the foam. Thus, this can be viewed as an optimisation problem - we want move the system towards equilibrium by tweaking the variables of the foam.

5.1 The numerical model

For the system to be in equilibrium, Plateau's first rule of equilibrium must hold true. This means that all angles at any junction must equal 120° or $\frac{2\pi}{3}$. Furthermore, for the system to be in a total state of equilibrium, all cell pressures have to equal the world pressure, meaning that no diffusion will take place and that areas won't change.

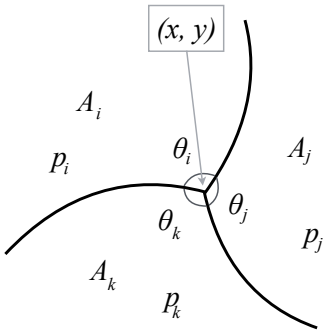


Figure 5.1: One random junction in a foam made up of three sections. Each section is associated with an angle θ, a cell area A and a cell pressure p. Furthermore, the junction is located at position (x, y).

Focussing on one junction as visualised in Figure 5.1, one can write up the optimal solution for a junction in equilibrium as

$$\begin{aligned}
\theta_i(x, y, p_i, p_j, p_k) &= \frac{2\pi}{3} \\
\theta_j(x, y, p_i, p_j, p_k) &= \frac{2\pi}{3} \\
\theta_k(x, y, p_i, p_j, p_k) &= \frac{2\pi}{3} \\
A_i(x, y, p_i, p_j, p_k) &= A_i^{target} \\
A_j(x, y, p_i, p_j, p_k) &= A_j^{target} \\
A_k(x, y, p_i, p_j, p_k) &= A_k^{target}
\end{aligned} \tag{5.1}$$

If these six equations are fulfilled for all junctions in the foam, the foam is in equilibrium an will remain stationary. However, knowing two angles for a junction automatically implies knowing the third angle, namely 360° minus the two known angles. This reduces the problem to having 5 equations. Subtracting the right sides in (5.1), one can write the 5 remaining equations on vector form for each junction as

$$F(\boldsymbol{\alpha}) = \begin{bmatrix} \theta_i(x,y,p_i,p_j,p_k) - \frac{2\pi}{3} \\ \theta_j(x,y,p_i,p_j,p_k) - \frac{2\pi}{3} \\ A_i(x,y,p_i,p_j,p_k) - A_i^{target} \\ A_j(x,y,p_i,p_j,p_k) - A_j^{target} \\ A_k(x,y,p_i,p_j,p_k) - A_k^{target} \end{bmatrix}, \tag{5.2}$$

$\boldsymbol{\alpha}$ being the set of five parameters $\boldsymbol{\alpha} = (x, y, p_i, p_j, p_k)^T$. We seek a solution $\boldsymbol{\alpha}^*$, for which

$$F(\boldsymbol{\alpha}^*) = \overrightarrow{0} \tag{5.3}$$

We have 5 unknown and 5 equations. One is in search of a $\Delta\boldsymbol{\alpha}$ moving the junction to a state of local equilibrium. Using a Newton step, to move one step ahead in time in the direction of $\Delta\boldsymbol{\alpha}$, one changes the vector function F by

$$F(\boldsymbol{\alpha}^{t+\Delta t}) = F(\boldsymbol{\alpha}^t) + \nabla F(\boldsymbol{\alpha}^t)\Delta\boldsymbol{\alpha} \tag{5.4}$$

where $\nabla F(\boldsymbol{\alpha}^t)$ is the Jacobian of $\boldsymbol{\alpha}^t$, i.e. a (5×5) matrix containing the partial derivatives derivatives of F with respect to $\boldsymbol{\alpha}$, i.e. $\frac{\partial F}{\partial \boldsymbol{\alpha}}$. The Jacobian of F is thus defined as

$$\nabla F = \begin{bmatrix} \frac{\partial\theta_i}{\partial x} & \frac{\partial\theta_i}{\partial y} & \frac{\partial\theta_i}{\partial p_i} & \frac{\partial\theta_i}{\partial p_j} & \frac{\partial\theta_i}{\partial p_k} \\ \frac{\partial\theta_j}{\partial x} & \frac{\partial\theta_j}{\partial y} & \frac{\partial\theta_j}{\partial p_i} & \frac{\partial\theta_j}{\partial p_j} & \frac{\partial\theta_j}{\partial p_k} \\ \frac{\partial A_i}{\partial x} & \frac{\partial A_i}{\partial y} & \frac{\partial A_i}{\partial p_i} & \frac{\partial A_i}{\partial p_j} & \frac{\partial A_i}{\partial p_k} \\ \frac{\partial A_j}{\partial x} & \frac{\partial A_j}{\partial y} & \frac{\partial A_j}{\partial p_i} & \frac{\partial A_j}{\partial p_j} & \frac{\partial A_j}{\partial p_k} \\ \frac{\partial A_k}{\partial x} & \frac{\partial A_k}{\partial y} & \frac{\partial A_k}{\partial p_i} & \frac{\partial A_k}{\partial p_j} & \frac{\partial A_k}{\partial p_k} \end{bmatrix} \tag{5.5}$$

Note that the equilibrium terms for the angles $(2\pi/3)$ and areas (A^τ) in F (5.2) vanish in the process of differentiation, as they are constants at the time t and do not dependent on $\Delta\alpha$.

One can try to write up the partial derivatives analytically. The upper left element in (5.5) would be the partial derivative of the angle from (3.14) with respect to x. This equation becomes rather complicated when calculating the dot products and lengths of the vectors. In this particular case, we would need to calculate the derivative of

$$\begin{aligned} \frac{\partial\theta_i}{\partial x} =& \frac{\partial}{\partial x}\Bigg(\cos^{-1}\left(\frac{x^2 - x_k x - x_j x + x_j x_k + y^2 - y_k y - y_j y + y_j y_k}{\sqrt{(x_j - x)^2 + (y_j - y)^2}\cdot\sqrt{(x_k - x)^2 + (y_k - y)^2}} \right) \\ &+ \frac{sgn(\Delta p_k)\sin^{-1}\left(\frac{\sqrt{(x_k-x)^2+(y_k-y)^2}}{4\Delta p_k/\gamma}\right) + sgn(\Delta p_j)\sin^{-1}\left(\frac{\sqrt{(x_j-x)^2+(y_j-y)^2}}{4\Delta p_j/\gamma}\right)}{2} \Bigg) \end{aligned}$$

where (x_k, y_k) denotes the coordinates of $\boldsymbol{x}_k$ and (x_j, y_j) denotes the coordinates of $\boldsymbol{x}_j$ from Figure 3.20, respectively. Differentiating this as well as well as the other 24 entries in the matrix in (5.5) is not straight forward, nor is it necessary. Numerically approximating the derivatives is a common and powerful tool within the field of physics simulations and many other fields and will prove sufficient in our case for finding a search direction.In the foam simulation of [Kermode and Weaire, 1990], which is the main foundation of the one in this thesis, numerical derivatives were also used.

5.1.1 Approximating derivatives

One way of approximating derivatives is to use finite differences. This thesis and will use the central difference approximation given by

$$F'(x) = \frac{F(x+h) - F(x-h)}{2h} \tag{5.6}$$

If $F'''(x)$ exists, the error on this approximation is $\mathcal{O}(h^2)$ [Kincaid and Cheney, 2002]. The approximation of the derivative gives us the slope of the secant intersecting at $x - h$ and $x + h$. The clear advantage of using finite differences is that we need only evaluate the functions for some given parameters and not differentiate them.

For some of the derivatives in the Jacobian, we can simplify the calculations in order to cut computation time. For instance, $\partial A_i/\partial x$ describes the change in area in cell i when moving the junction coordinate in the x-direction. Calculating the area of a bubble cell consists of calculating different sections and adding them at the end as shown in Figure 3.18. As Figure 5.2 shows, only the grey shaded area has to be calculated, as only this area changes when moving the x-coordinate. And since we are dealing with *changes* in area, the junction can also be moved beyond the grey shaded area without a need for calculating the area of the entire cell.

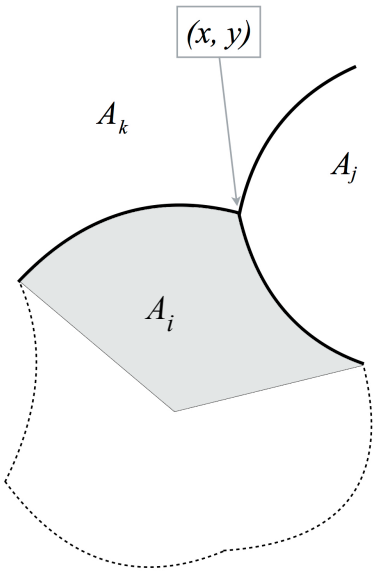


Figure 5.2: An illustration of the same junction as in Figure 5.1, but with the adjacent cell i visualised and the area spanned by the three junctions needed to calculate the change in area, when altering the position of the junction at (x, y)

This approximation also holds for most of the terms involving $\frac{\partial A}{\partial p}$, where it is only necessary to use half of the grey slice facing the cell involving the pressure

change. However, for the terms $\frac{\partial A_i}{\partial p_i}$, $\frac{\partial A_j}{\partial p_j}$ and $\frac{\partial A_k}{\partial p_k}$, one cannot only calculate the area changes in the grey slice, since the rest of the bubble films in the cell will also change their radius of curvature, thus affecting the area of the bubble.

5.1.2 Computing the target area

Being able to define a target area of a cell is crucial in this approach to the relaxation process. Integrating von Neumann's equation (2.14) for cell n gives us the projected change in area in the next time step. Adding this to the current area of the cell gives us the target area. Integrating with respect to time t from zero to Δt yields

$$A^{target} = A - \int_0^{\Delta t} \kappa \sum_j (p - p_j) l_j dt = A - \Delta t \cdot \kappa \cdot \sum_j (p - p_j) l_j \tag{5.7}$$

Now, this integration is a simplification of the real physical foam, since the pressure p and the surrounding cell pressures, the p_j's, are time dependent and will change in the time interval between t and $t + \Delta t$. Since p and the p_j's are not constants, we would not be able to simply pull them out of the integral. That is, we are really dealing with a system of coupled differential equations. However, having a small enough Δt allows us to ignore this.

5.1.3 Solving the system of linear equations

With these technicalities in place, we can move on to solving the system. (5.3) states the optimal solution at which a junction is in a state of equilibrium. If all junctions obey this, the whole system will be in a state of equilibrium. (5.4) described one Newton-step for a junction. Using a *local* approach, our desire is to move towards a point of equilibrium for the junction in question. For this to be the case, we seek a step that makes $F = \vec{0}$

$$F(\boldsymbol{\alpha}^{t+\Delta t}) = F(\boldsymbol{\alpha}^t) + \nabla F(\boldsymbol{\alpha}^t) \Delta \boldsymbol{\alpha} = \vec{0} \tag{5.8}$$

Thus, we want to solve the linear system

$$\nabla F(\boldsymbol{\alpha}^t) \Delta \boldsymbol{\alpha} = -F(\boldsymbol{\alpha}^t) \tag{5.9}$$

for each junction giving us the step direction $\Delta \boldsymbol{\alpha}$

$$\Delta \boldsymbol{\alpha} = -\nabla F(\boldsymbol{\alpha}^t)^{-1} F(\boldsymbol{\alpha}^t) \tag{5.10}$$

This step would bring the junction to a state of local equilibrium. However, we may not want to move the whole length of $\Delta\boldsymbol{\alpha}$, since it might result in geometrical or physical inconsistencies, as the example in Figure 5.3 shows.

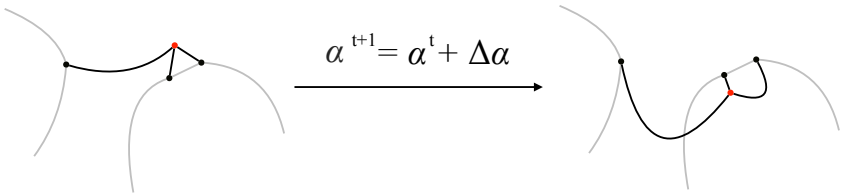


Figure 5.3: An arbitrary example of moving a junction towards a situation, where the first rule of Plateau is fulfilled, thereby making $F(\boldsymbol{\alpha}^t + \Delta\boldsymbol{\alpha}) = \vec{0}$, if the pressures simultaneously are updates as to accommodate the target areas. However, the step is not allowed, since it results in geometrical and physical inconsistencies.

As a result, a backtracking approach is performed by reducing the step length until different criteria are fulfilled with a scaling τ, which will be described in Section 5.1.5.

5.1.4 System of linear equations on the boundary

When dealing with a boundary junction, the size of the problem reduces. Since one area, namely the world area, is constant, $F(\boldsymbol{\alpha})$ has one element which will always equal zero, since the target area and the present area are both ∞. Let this element be the last element in the vector, namely A_k. Similarly, the fifth element of $\Delta\boldsymbol{\alpha}$, Δp_k, will always be zero, by construction. This eliminates the whole last column in the Jacobian (5.5). And since A_k will not change, no matter what Δx's, Δy's or Δp's are thrown at it, so will the bottom row of the Jacobian also be zero. Thus, the problem reduces from a 5×5-problem to a 4×4-problem.

In the implementation, it is always made sure that the vertex v_k holding p_k in the face encapsulating a boundary junction always is the boundary vertex. Thus, the upper left 4×4-corner of the Jacobian can always be clipped, if we deal with such a situation.

5.1.5 Checking for validity of step

Once one has found a step direction, it is now a question of scaling this step with a scalar τ, such that an update of α yields a valid step $\alpha^t + \tau\Delta\alpha$. This also results in different scaling constants for different junctions.

These criteria have been chosen as a result of geometrical and physical considerations. The pseudo-code is sketched in Algorithm 5.

The following bullet points mark the different checks that are performed in the backtracking (in the same order as in Algorithm 5)

- **The junction stays within the three neighbouring cells**. This is often not violated anyway, since the search direction should move the junction towards a situation, where Plateau's first rule of equilibrium holds thereby maintaining a convex shape.
- **Decrease in the surface energy**, which describes the energy in the system. As with other physics simulations, the system is expected to decrease in energy. This is a local adjustment, since the search directions are found locally.
- **Diffusion is restricted** by a user-defined amount. Some simulations tend to be dominated by diffusion. Von Neumann's law restricts the diffusion, which is also incorporated in the calculation of target areas. However, since the model is a local one, it does not take the neighbour cells into account. Hence, a further restriction is added, should the diffusion be too violent in some cases. It, however, proved to only happen on rare occasions.
- **The angles in a Plateau junction must always sum to** 2π. This is a geometrical invariant that may be violated, if films bend across each other thus resulting the angle calculator to double count the part, where the films have crossed. This would result in an angle sum larger than 2π, which is a violation of the physical constraints in the foam.
- **An sufficient decrease in the merit function**. The search direction will always move us in a direction, for which the merit function is decreased, where the merit function is the commonly used sum of squares merit function $\frac{1}{2}F^TF$ [Nocedal and Wright, 2006]. This is obvious, since we solve for a situation giving us a merit function equal to zero. Why do we then have to potentially *decrease* the movement in the search direction? Now, the merit function along τ may not necessarily be monotonous due to the geometrics of the foam as sketched in Figure 5.4. Thus, a linear function with a user-defined negative slope c is defined to define an upper boundary for the fall in the merit function, where the magnitude of c is smaller than the magnitude of the derivative of the merit function at $\tau = 0$. This is an addition to ensure that a certain decrease is achieved - otherwise it makes no sense moving in this direction.

Only when *all* conditions are fulfilled in Algorithm 5, will τ be returned. Instead of all the while loops, one could also make the whole search for τ an optimisation problem on its own, however, this might seem a bit too much of a struggle and a costly affair to do at every iteration in the simulation. Hence, the conditions have been ordered in such a way that if τ has some value, once the algorithm reaches a certain while loop, the above conditions won't be violated, if τ is reduced additionally. For this reason, the sufficient decrease is the last check to be made, since it does not necessarily behave nice and well over the range of τ, as discussed. But it must be said, that most of the times, it is nice and monotonously decreasing.

Updating a junction once with the scaled down step length in most cases results in the junction not reaching a state of equilibrium. And since different junctions most likely experience different values of τ, one can argue that the concept of *time* is different for the different junctions.

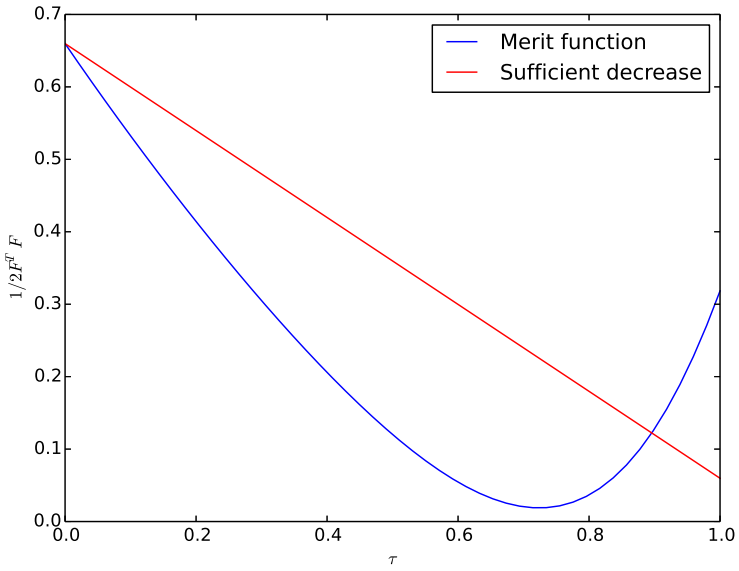


Figure 5.4: The merit function of the search direction $\tau \cdot \Delta\alpha$ as a function of τ for a random junction. The red line symbolises a sufficient decrease restriction given by the user. If the backtracking algorithm would find a τ, where the blue line is above the red, the sufficient decrease addition would reduce τ in such a way that it would move into the regime, where the merit function is lower than the sufficient decrease line. The line has a slope specified by the user.

5.1.6 Local relaxation

Thus, an approach is applied, where each junction is updated iteratively, until it reaches a local equilibrium or has iterated a user-specified maximum number of iterations. Figure 5.5 shows a visual interpretation of such a case.

Figure 5.5: The movement of a junction resulting from updating (x, y)-coordinates. At each step, a new Jacobian J and deviation vector F are calculated. In the update of F, the target angles and target areas are kept fixed.

Thus, instead of updating the parameters with $\tau \cdot \Delta\alpha$, each junction is allowed to be updated k_{max} number of times.

$$\alpha^{t+1} = \alpha^t + \sum_k \tau^k \cdot \Delta\alpha^k \tag{5.11}$$

where k denotes the iteration of the inner loop for a particular junction used in the relaxation process of bringing the junction to a state of equilibrium, and t denotes the iteration of the outer loop for updating the whole foam. Thus, different junctions may need a different number of iterations for local relaxation. If all junctions reach equilibrium, the version of von Neumann's law as presented in (2.15) holds, since Plateau's first rule of of equilibrium holds. And one can generally say that cells with less than 6 sides will shrink, cells with more than 6 sides will grow and cells with 6 sides will remain constant in area.

The calculation of each $\Delta\alpha^k$ follows by changing the indices in (5.10) to mirror the right iteration in the relaxation process

$$\boxed{\Delta\boldsymbol{\alpha}^k = -\nabla F(\boldsymbol{\alpha}^k)^{-1} F(\boldsymbol{\alpha}^k)} \tag{5.12}$$

If the junction reaches a local equilibrium within the user-defined maximum number of k-iterations, k_{max}, one could ask: If it is in a state of equilibrium, and it is the same for the other junctions as well, will the foam not remain stationary from then on? And the answer is that it will, if there is no diffusion. But since we have a model with diffusion and a lower non-changing pressure outside the foam, diffusion will keep the foam moving, since there will be a diffusion towards the outside at all times and diffusion between cells of different pressures. But in the relaxation loop, the diffusion is kept fixed. When calculating $F(\boldsymbol{\alpha}^k)$, the target areas are the same throughout the whole relaxation process.

One of the reasonings for the local relaxation is that the surface tension forces dominate the the diffusion on a short time scale. This, of course, also depends on the permiability of diffusion κ and the surface tension γ, but generally holds true, and was observed in the simulation. One could also have $k_{max} = 1$ thus only moving each junction once, $\alpha^{t+1} = \alpha^t \tau \cdot \Delta\alpha$. This approach was also tried in working on this thesis, but allowing local relaxation for all junctions at time t in a sense provides equal terms for all junction in moving towards a local and temporary state of equilibrium no matter the step reduction as a result of the backtracking or the accuracy of the numerical model at α^t. This approach is also known as the *quasi-static* approach [Kelager, 2009], as it brings the foam into a condition of quasi-static equilibrium.

The point about the accuracy of the numerical model at α^t relates to the whole linear approximation of the step using a Newton step, since the Jacobian is a numerical first order approximation only telling about the effects of position and pressures on junction angles and areas at that very position and point in time. For some junctions, the step direction might reduce F more smoothly than others, since the Jacobian does not change much when updating α slightly. This means that the linear approximation is very precise. For other junctions, this might not be the case to the same extend, which further justifies the introduction of the local relaxation.

As a measure for, when a junction has reached equilibrium, the merit function $\frac{1}{2}F^T F$ as also introduced in Section 5.1.5 is used as a measure of local convergence. In order to update F, it would be interesting to investigate, roughly how

many digits of precision one might lose in precision and thus how large the uncertainty on the resulting F is. This involves calculating the step direction in order to update F. In terms of complexity, the calculation of the inverse of a matrix using conventional Gauss-Jordan elimination is of order $\mathcal{O}(n^3)$, and thus one might expect to lose $5^3 = 125 \approx 10^2$ digits of precision in the calculation of the Jacobian. Multiplying with the Jacobian F to get the search direction adds another $\mathcal{O}(n^3)$ to the complexity, whilst squaring the updated F to get the value of $\frac{1}{2}F^TF$ is of order $\mathcal{O}(n^2)$, which is dominated by $\mathcal{O}(n^3)$. Thus, we have $\mathcal{O}(n^3)\mathcal{O}(n^3) = \mathcal{O}((2n)^3)$. $25^3 = 15625 \approx 10^4$, and running a 64-bit machine with a machine epsilon of $\tilde{1}0^{-16}$, one might thus roughly expect 4 digits of precision resulting of a maximum precision of 10^{-12}. This is then the minimum for ϵ_{relax}, the local relaxation convergence measure for the merit function.

5.1.7 Rank of the Jacobian

There are multiple ways of finding the inverse of $\nabla F(\alpha)$ depending on the condition of it. Finding a straight forward inverse is not always possible, and different methods can be used to solve this.

[Weaire and Hutzler, 2001] sketches an iterative update method for pressures and junction coordiantes. However, with such a small system, NumPy's pseudo-inverse is an inexpensive and accurate way to find the search direction $\Delta\alpha$, provided that the Jacobian has full rank and is well-conditioned.

At first in the process of implementation, I did not realise the false assumption that all area derivatives could be approximated by only considering the area slice as described in Section 5.1.1 and shown in Figure 5.2. Thus, the Jacobian would only have column rank 4 despite being of size 5×5. This, eventually would lead to more unstable solutions of the search direction. Similarly, the condition numbers of the Jacobian matrices for the different junctions would be very large indicating more unstable and inaccurate solutions for the search direction and the problem ill-conditioned. The reason for this shall be found in the relative pressures used to calculate the areas. Consider the following rephrasing of the lower right 3×3 corner of the Jacobian shown in 5.5:

$$\begin{bmatrix} \frac{\partial A_i}{\partial p_i} & \frac{\partial A_i}{\partial p_j} & \frac{\partial A_i}{\partial p_k} \\ \frac{\partial A_j}{\partial p_i} & \frac{\partial A_j}{\partial p_j} & \frac{\partial A_j}{\partial p_k} \\ \frac{\partial A_k}{\partial p_i} & \frac{\partial A_k}{\partial p_j} & \frac{\partial A_k}{\partial p_k} \end{bmatrix} = \begin{bmatrix} \frac{\partial A_i}{\partial p_i} & \frac{\partial A_i}{\partial p_j} & \frac{\partial A_i}{\partial p_k} \\ \frac{\partial A_i}{\partial p_j} & \frac{\partial A_j}{\partial p_j} & \frac{\partial A_k}{\partial p_j} \\ \frac{\partial A_i}{\partial p_k} & \frac{\partial A_k}{\partial p_j} & \frac{\partial A_k}{\partial p_k} \end{bmatrix} \tag{5.13}$$

By using the false assumption by calculating the derivatives depending on changes in the area slices only, one would end up with all row sums and column sums equal to zero, since increasing the pressure in for instance cell i would be the same as lowering the pressure by certain amounts in cells j and k. This would result in rank deficiency *and* a determinant equal to zero making the matrix non-invertible.

The two rows above the 3×3 matrix from (5.13) are also rank deficient, since they describe the angles with respect to pressure changes. And using the same argu-

mentation, increasing the pressure in one cell corresponds to lowering the pressure in the other two cells bordering the junction. But this is not wrong in any way, since the angles of a junction don't care about what changes are made to the other angles at he neighbouring junctions as was the case with the areas. Thus, one needs only to focus on the junction itself when calculating the angle derivatives, whereas one needs to take the whole cells into perspective, when calculating the area derivatives. By taking the whole cell are into account, when calculating the area derivatives, the Jacobian has rank 5 and is easy to invert.

5.1.8 Extending to all junctions - a global approach

The local approach presented above solves a system for each junction. But what about solving a system for the whole system at once? Consider the Jacobian outlined in (5.14). It describes all dependencies in the foam with m junctions and n cells. Pick a junction i. Its angles will depend on the pressure from the three adjacent cells (3 variables) as well of the positions of the junction itself (2 variables) as well as the positions of the three neighbour junctions (6 variables), i.e. 11 dependencies in total. Each of the first $2m$ rows in the global Jacobian represents angle derivatives, whereas the last n rows symbolise area derivatives. The columns symbolise the variables, m x's, m y's and n p's. For the angles, with 11 dependencies, all the rest of the row elements will be zeros. As for the area rows, the number of non-zero elements will depend on the number of neighbours of that particular cell.

$$\nabla F_{global}(\boldsymbol{\alpha}) = \begin{bmatrix} \frac{\partial\theta_{0,0}}{\partial x_0} & \frac{\partial\theta_{0,0}}{\partial x_1} & \cdots & \frac{\partial\theta_{0,0}}{\partial x_m} & \frac{\partial\theta_{0,0}}{\partial y_0} & \frac{\partial\theta_{0,0}}{\partial y_1} & \cdots & \frac{\partial\theta_{0,0}}{\partial y_m} & \frac{\partial\theta_{0,0}}{\partial p_0} & \frac{\partial\theta_{0,0}}{\partial p_1} & \cdots & \frac{\partial\theta_{0,0}}{\partial p_n} \\ \frac{\partial\theta_{0,1}}{\partial x_0} & \frac{\partial\theta_{0,1}}{\partial x_1} & \cdots & \frac{\partial\theta_{0,1}}{\partial x_m} & \frac{\partial\theta_{0,1}}{\partial y_0} & \frac{\partial\theta_{0,1}}{\partial y_1} & \cdots & \frac{\partial\theta_{0,1}}{\partial y_m} & \frac{\partial\theta_{0,1}}{\partial p_0} & \frac{\partial\theta_{0,1}}{\partial p_1} & \cdots & \frac{\partial\theta_{0,1}}{\partial p_n} \\ \frac{\partial\theta_{1,0}}{\partial x_0} & \frac{\partial\theta_{1,0}}{\partial x_1} & \cdots & \frac{\partial\theta_{1,0}}{\partial x_m} & \frac{\partial\theta_{1,0}}{\partial y_0} & \frac{\partial\theta_{1,0}}{\partial y_1} & \cdots & \frac{\partial\theta_{1,0}}{\partial y_m} & \frac{\partial\theta_{1,0}}{\partial p_0} & \frac{\partial\theta_{1,0}}{\partial p_1} & \cdots & \frac{\partial\theta_{1,0}}{\partial p_n} \\ \frac{\partial\theta_{1,1}}{\partial x_0} & \frac{\partial\theta_{1,1}}{\partial x_1} & \cdots & \frac{\partial\theta_{1,1}}{\partial x_m} & \frac{\partial\theta_{1,1}}{\partial y_0} & \frac{\partial\theta_{1,1}}{\partial y_1} & \cdots & \frac{\partial\theta_{1,1}}{\partial y_m} & \frac{\partial\theta_{1,1}}{\partial p_0} & \frac{\partial\theta_{1,1}}{\partial p_1} & \cdots & \frac{\partial\theta_{1,1}}{\partial p_n} \\ \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial\theta_{m,0}}{\partial x_0} & \frac{\partial\theta_{m,0}}{\partial x_1} & \cdots & \frac{\partial\theta_{m,0}}{\partial x_m} & \frac{\partial\theta_{m,0}}{\partial y_0} & \frac{\partial\theta_{m,0}}{\partial y_1} & \cdots & \frac{\partial\theta_{m,0}}{\partial y_m} & \frac{\partial\theta_{m,0}}{\partial p_0} & \frac{\partial\theta_{m,0}}{\partial p_1} & \cdots & \frac{\partial\theta_{m,0}}{\partial p_n} \\ \frac{\partial\theta_{m,1}}{\partial x_0} & \frac{\partial\theta_{m,1}}{\partial x_1} & \cdots & \frac{\partial\theta_{m,1}}{\partial x_m} & \frac{\partial\theta_{m,1}}{\partial y_0} & \frac{\partial\theta_{m,1}}{\partial y_1} & \cdots & \frac{\partial\theta_{m,1}}{\partial y_m} & \frac{\partial\theta_{m,1}}{\partial p_0} & \frac{\partial\theta_{m,1}}{\partial p_1} & \cdots & \frac{\partial\theta_{m,1}}{\partial p_n} \\ \frac{\partial A_0}{\partial x_0} & \frac{\partial A_0}{\partial x_1} & \cdots & \frac{\partial A_0}{\partial x_m} & \frac{\partial A_0}{\partial y_0} & \frac{\partial A_0}{\partial y_1} & \cdots & \frac{\partial A_0}{\partial y_m} & \frac{\partial A_0}{\partial p_0} & \frac{\partial A_0}{\partial p_1} & \cdots & \frac{\partial A_0}{\partial p_n} \\ \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial A_n}{\partial x_0} & \frac{\partial A_n}{\partial x_1} & \cdots & \frac{\partial A_n}{\partial x_m} & \frac{\partial A_n}{\partial y_0} & \frac{\partial A_n}{\partial y_1} & \cdots & \frac{\partial A_n}{\partial y_m} & \frac{\partial A_n}{\partial p_0} & \frac{\partial A_n}{\partial p_1} & \cdots & \frac{\partial A_n}{\partial p_n} \end{bmatrix} \tag{5.14}$$

Similarly, one can write up a global version of (5.2), the right side of the linear equation as a total for the whole foam

$$F_{global}(\boldsymbol{\alpha}) = \begin{bmatrix} \theta_{0,0} - \frac{2\pi}{3} \\ \theta_{0,1} - \frac{2\pi}{3} \\ \theta_{1,0} - \frac{2\pi}{3} \\ \theta_{1,1} - \frac{2\pi}{3} \\ \vdots \\ \theta_{m,0} - \frac{2\pi}{3} \\ \theta_{m,1} - \frac{2\pi}{3} \\ A_0 - A_0^{target} \\ \vdots \\ A_n - A_n^{target} \end{bmatrix} \tag{5.15}$$

Consequently, one can rewrite (5.10) to find a global search direction $\Delta\boldsymbol{\alpha}_{global}$

$$\boxed{\Delta\boldsymbol{\alpha}_{global} = -\nabla F_{global}(\boldsymbol{\alpha}^t)^{-1} F_{global}(\boldsymbol{\alpha}^t)}\,, \tag{5.16}$$

where $\boldsymbol{\alpha}$ has the same dimensions as F_{global} and corresponds to the variables specifying each column in ∇F_{global}.

This global approach is obviously very sensitive to the size of the foam. Inverting a 2000×2000 matrix is very costly in terms of computation, whereas the local approach is independent of the foam size, when it comes to solving the system of linear equations - it will only have to solve more such systems with increasing foam size. Thus, this global approach has its benefits, since it incorporates more dependencies and gives one search direction for the whole system. It turned out that the backtracking would oftentimes stall the simulation, once the foam had been equilibrated to a certain extend. As a result, a local backtracking was performed on the global step direction, as Algorithm 8 outlines in more detail.

It goes without saying that the local relaxation is, by the nature of the global approach, not possible to perform, since all junctions are treated simultaneously. To compare this to the local approach, this corresponds to having $k_{max} = 1$, where variables are updated only once per iteration cycle at a given t.

5.2 Implementation

This thesis will present three different methods for simulating the foam. They are all root finding problems optimising for angles and areas and use the same constraints, but they differ in the way the foam is updated. I have chosen to name the three methods the *local* method, the *quasi-global* method and the *global* method[1]. They will be compared statistically, visually and with respect to the speed that they each coarsen the foam.

[1] A note on vocabulary: The three methods use the same *numerical method* for finding the step direction. The local and quasi-global methods are different from the global method in terms of the *numerical model* and only differ in the way they update the variables. The terms *method* and *approach* will both be used to describe the local, quasi-global and global methods.

- **The local method** performs local relaxation by iteratively calculating a search direction for each junction. Then it performs backtracking to find a valid step size and updates the variables with pressure updates for a given cell normalised with the number of junctions facing this cell. This is repeated until local convergence is reached, or until a certain number of relaxation iterations have been performed.
- **The quasi-global method** performs local relaxation by iteratively calculating a search direction for each junction. Then it performs backtracking to find a valid step size. This is repeated until local convergence is reached, or until a certain number of relaxation iterations have been performed. The updates are stored as a property for the junction. Eventually, after all junctions have been treated, all variables are updated at once, and the pressure updates for a given cell are normalised with the number of junctions facing the cell.
- **The global method** calculates one search direction for the whole foam. It uses a local backtracking on the global search direction. It uses a Jacobian describing dependencies for all areas and angles with the junction coordinates and pressure values in the whole foam. To perform an inversion of such a possibly large Jacobian may prove very costly, but when the problem is well-conditioned, this global approach does take more dependencies into account, which could theoretically result in better results.

In the local and in the quasi-global method, the approach of dividing each pressure update with the number of edges (the same as the number of films) facing the cell was used by [Vedel-Larsen, 2010] in his thesis as well as in the implementation presented in [Kermode and Weaire, 1990]. This scaling ensures an even diffusion contribution from neighbouring cells, no matter the film length. But the film length is already incorporated in the calculation of the target area, and has thus already been taken account for. Initially, I only did this for the quasi-global method, but found it reasonable to also incorporate into the local method, thereby ensuring that each cell pressure is only updated once per iteration cycle, since the sum of the contributions from the cell-facing junctions are normalised.

The drawback of this is that it counteracts the local relaxation, since different junctions may counteract each other. Thus, we cannot reasonably expect Plateau junctions all over the foam at every iteration. It is an act of balance between local relaxation and smooth pressure changes throughout the foam.

Moreover, an iteration i through the foam taking the foam one step forward in time from t to $t + \Delta t$ shall also be known as a *frame* in this thesis.

5.2.1 Parameters used

Table 4 outlines the different parameters used in the simulation. Some are justified by experimenting with the simulation and seeing what made the algorithm most stable. For instance, the surface tension γ had a profound effect, when the simulation hit a very high number of iterations. A too small γ along with a too large κ would make the diffusion dominate the angle requirements and vice versa. The connection between γ, κ and Δt had to be adjusted, but at the end, the ratio be-

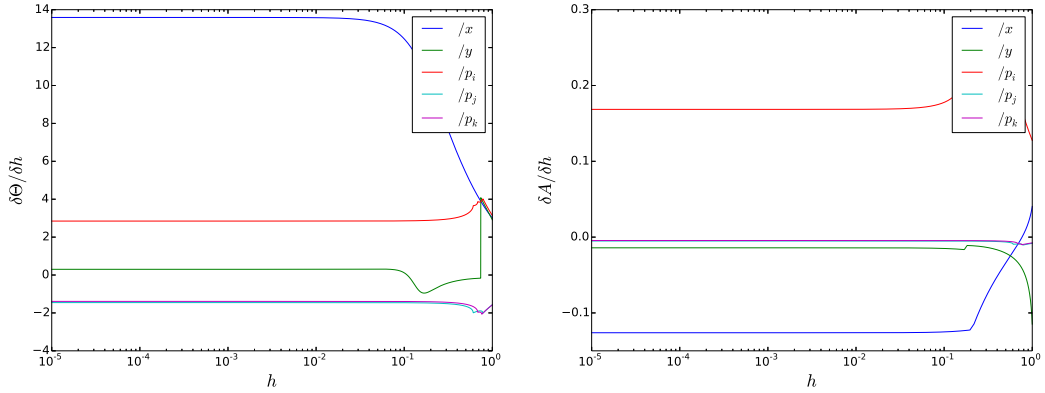
tween these parameters ended up being roughly the same as [Vedel-Larsen, 2010] ended up using for his thesis, but with a smaller Δt for this simulation. It is important to stress that the units in Table 4 are not to be thought of in a literal way. The minimum threshold for the film length δ_{T1}, before a T1 operation is performed, is for instance 6.38 cm in the table, which seems quite large that no films below this length should exist. But these values are arbitrarily tuned to the foam used in the simulation - a foam with width 7.5, and the units could be meters, centimetres or millimetres for that matter, as long as the ratios between the parameters are reasonable. The pressure, for instance, is also not to be thought of as a real physical pressure in terms of units such as Pa, since everywhere in the simulation, where the pressure is used, it is used in terms of pressure *differences*, and then it is only a matter of scaling, too, as it is with the other parameters.

δ_{T1} and δ_{T2} were chosen on the basis of investigating, how the film lengths and cell areas were distributed throughout a simulated foam after a few iterations. The thresholds were chosen as approximately 2 standard deviations subtracted from the mean of the two distributions, respectively, corresponding to about 5 percent of the total number of film lengths and areas. This, obviously was chosen under the assumption that they are normally distributed, which was not the case, but in the process of tuning the parameters, it was used as an initial way of finding thresholds that were later fine-tuned.

Symbol	Meaning	Value	Unit
γ	Surface tension	0.025	m^2/s
κ	Permeability of diffusion	0.5	$\mathrm{N/m}$
Δt	Time step used in calculation of target areas	0.001	s
h	Finite difference step size	0.0001	None
β	Reduction fraction in backtracking algorithm	0.5	None
d_r	Restriction of diffusion in backtracking algorithm	0.2	None
k_{max}	Maximum number of local relaxation iterations	10	None
c	Sufficient decrease damping factor	0.1	None
δ_{T1}	Threshold for film length, before T1 is performed	0.0638	m
δ_{T2}	Threshold for cell area, before T2 is performed	0.0639	m^2

Table 4: Overview of parameters used in the simulation

Figure 5.6 shows the dependencies of the angle derivatives and area derivatives with respect to the finite difference h for the five different variables x, y, p_i, p_j and p_k for some junction inside the foam. One does not want to have an h too small to avoid the limitations of numerical accuracy in the machine, but one still wants an h small enough for it to be representative for the derivative, which is defined in the limit of $h \to 0$. As the plots shows, picking an h somewhere in the regime between 10^{-5} and 10^{-3} seems reasonable.



(a) Angle derivatives

(b) Area derivatives

Figure 5.6: Derivatives in the Jacobian as a function of the finite difference, h. The x-axis is logarithmic.

5.3 Pseudo code of algorithms

input : Face handle for variables at one junction, step direction $\Delta\alpha$, restriction fraction of diffusion diffusionRestriction, update factor β for τ
output: Step length scaling τ

$\tau = 1$

`/* Ensure that junction stays within the borders of the three neighbouring cells */`
while $(\mathtt{x}, \mathtt{y}) + (\Delta\mathtt{x}, \Delta\mathtt{y})$ *is not inside one of the adjacent three cells* **do**
  $\tau = \beta \cdot \tau$
end

`/* Ensure local decrease in surface energy */`
while $\sum_i^3(\mathtt{filmLength}(\alpha_i + \tau \cdot \Delta\alpha_i)) > \sum_i^3(\mathtt{filmLength}(\alpha_i))$ **do**
  $\tau = \beta \cdot \tau$
end

`/* Restrict diffusion */`
while $\frac{p_i^{new} - p_i}{p_i} >$ diffusionRestriction **do**
  $\tau = \beta \cdot \tau$
end

`/* Ensure that angle sum is always` 2π`. A larger value indicates, that films bend across each other resulting in a larger angle sum */`
while $\sum_i \theta_i \neq 2\pi$ **do**
  $\tau = \beta \cdot \tau$
end

`/* Ensure a sufficient decrease in merit function */`
while $F(\alpha + \tau \cdot \Delta\alpha)^T F(\alpha + \tau \cdot \Delta\alpha) > \mathtt{c} \cdot \tau \cdot \frac{\partial}{\partial\tau} F(\alpha)^T F(\alpha)$ **do**
  $\tau = \beta \cdot \tau$
end

return τ

Algorithm 5: Backtracking algorithm for a junction with search direction $\Delta\boldsymbol{\alpha}$. All criteria must be satisfied, before τ is returned.

input : Initial foam mesh to equilibrate, the number of time steps iterations_{max} to update foam, minimumLength of a bubble film inside the foam until it is flipped in a T1 operation, minimumArea of a bubble inside the foam until it is collapsed in a T2 operation, $\mathfrak{p}$ as threshold for when satisfying merit function has been reached, $\mathsf{k}_{\mathsf{max}}$ for maximum number of iterations per junction
output: None (since OpenMesh data structure is mutable)

```
1  for i ← 0 to iterations_max do
2      visualise(mesh)
3      /* Perform topological operations                          */
4      foreach edge e in mesh do
5          if length(e) < minimumLength then
6              performT1 (e)
7          end
8      end
9      foreach vertex v in mesh do
10         if area(v) < minimumArea then
11             performT2 (v)
12         end
13     end
14     /* Coarsen foam by iterating over junctions               */
15     foreach face f in mesh do
16         k = 0
17         α_t = (x, y, p_i, p_j, p_k)^T
18         Δα_t = (0, 0, 0, 0, 0)^T
19         while (1/2) F_k^T F_k > threshold do
20             Δα_k = −inv(J_k) · F_k
21             τ_k = backtrack(mesh_k, f, Δα_k, β)
22             Δα_k = τ_k · Δα_k
23             Δα_t = Δα_t + Δα_k
24             k = k + 1
25             update F_{k+1} with Δα_k
26             if k > k_max then
27                 Exit loop
28             end
29         end
30         Δα_t(3 → 5) = Δα_t(3 → 5)/n_neighbours
31         α_{t+1} = α_t + Δα_t
32     end
33 end
34 visualise(mesh)
```

Algorithm 6: Local algorithm sketching the approach of solving the numerical model of the foam. The cells are updated one after another. Each iteration i corresponds to one time step in the simulation, i.e. one frame.

input : Initial foam mesh to equilibrate, the number of time steps iterations_{max} to update foam, minimumLength of a bubble film inside the foam until it is flipped in a T1 operation, minimumArea of a bubble inside the foam until it is collapsed in a T2 operation, $\flat$ as threshold for when satisfying merit function has been reached, k_{max} for maximum number of iterations per junction
output: None (since OpenMesh data structure is mutable)

for $i \leftarrow 0$ **to** iterations_{max} **do**
    visualise(mesh)

    /* Perform topological operations */
    foreach *edge* e *in* mesh **do**
        if length(e) < minimumLength **then**
            performT1 (e)
        end
    end
    foreach *vertex* v *in* mesh **do**
        if area(v) < minimumArea **then**
            performT2 (v)
        end
    end

    /* Calculate updates of variables in foam */
    foreach *face* f *in* mesh **do**
        $\text{k} = 0$
        $\alpha_t = (\text{x}, \text{y}, \text{p}_\text{i}, \text{p}_\text{j}, \text{p}_\text{k})^T$
        $\Delta\alpha_t = (0, 0, 0, 0, 0)^T$
        while $\frac{1}{2}\text{F}_\text{k}^T\text{F}_k >$ threshold **do**
            $\Delta\alpha_k = -\text{inv}(\text{J}_k) \cdot \text{F}_k$
            $\tau_k = \text{backtrack}(\text{mesh}_k, \text{f}, \Delta\alpha_k, \beta)$
            $\Delta\alpha_k = \tau_k \cdot \Delta\alpha_k$
            $\Delta\alpha_t = \Delta\alpha_t + \Delta\alpha_k$
            $\text{k} = \text{k} + 1$
            update F_{k+1} *with* $\Delta\alpha_k$
            if $\text{k} > \text{k}_{\text{max}}$ **then**
                Exit loop
            end
        end
        $\Delta\alpha_t(3 \rightarrow 5) = \Delta\alpha_t(3 \rightarrow 5)/\text{n}_{\text{neighbours}}$
        save($\Delta\alpha_t$)
    end

    /* Coarsen foam by applying stored updates */
    foreach *face* f *in* mesh **do**
        $\Delta\alpha_t = \text{load}(\Delta\alpha_t(\text{f}))$
        $\alpha_{t+1} = \alpha_t + \Delta\alpha_t$
    end
end
visualise(mesh)

Algorithm 7: Quasi-global algorithm sketching the approach of solving the numerical model of the foam. The cells are updated after all step directions have been calculated. Each iteration i corresponds to one time step in the simulation, i.e. one frame.

```
input : Initial foam mesh to equilibrate, the number of time steps
        iterations_max to update foam, minimumLength of a bubble film inside
        the foam until it is flipped in a T1 operation, minimumArea of a
        bubble inside the foam until it is collapsed in a T2 operation.
output: None (since OpenMesh data structure is mutable)

for i ← 0 to iterations_max do
    visualise(mesh)

    /* Perform topological operations                                  */
    foreach edge e in mesh do
        if length(e) < minimumLength then
            performT1 (e)
        end
    end
    foreach vertex v in mesh do
        if area(v) < minimumArea then
            performT2 (v)
        end
    end

    /* Calculate updates of variables in foam                          */
    Δα = −inv(J_global) · F_global

    /* Find scaling of variables for each junction                     */
    foreach face f in mesh do
        τ = backtrack(mesh, f, Δα, β)
        Δα(f) = τ · Δα
    end

    /* Coarsen foam by updating variables                              */
    foreach face f in mesh do
        Δα_temp = Δα(f)
        (x, y) = (x, y) + Δα(1 → 2)
        (p_i, p_k, p_j) = (p_i, p_j, p_k) + Δα(3 → 5)/n_neighbours
    end
end
visualise(mesh)
```

Algorithm 8: Global algorithm sketching the approach of solving the numerical model of the foam. The backtracking loop takes each junction (face in dual mesh) and performs the same backtracking algorithm as the local methods. It saves updates the corresponding part of $\Delta\boldsymbol{\alpha}$ with resulting τ. Each iteration i corresponds to one time step in the simulation, i.e. one frame.

6 Results

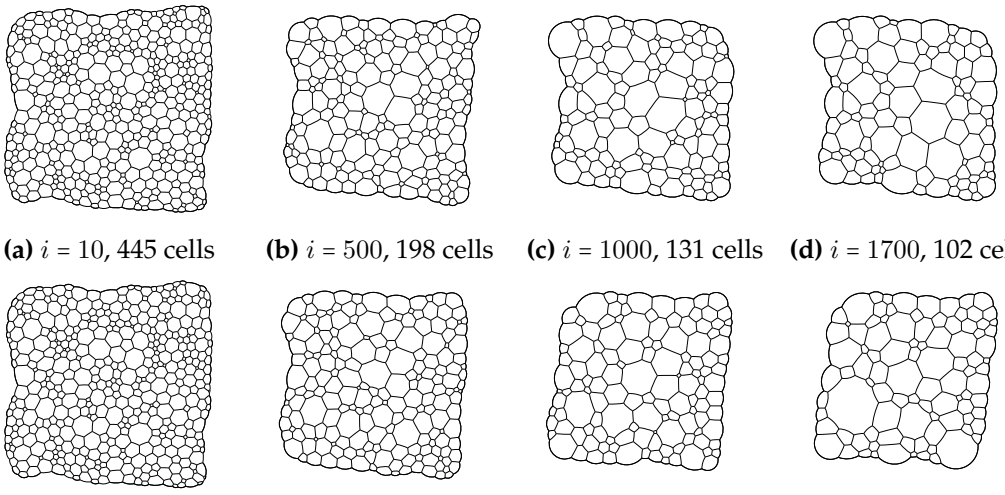
The three different methods all fare differently in terms of performance and results. The inversion of the global Jacobian is very time-consuming and depends a lot on the size of the system. But rather interestingly, the results of this global approach are quite a lot different from the two others.

6.1 The three method - a general comparison

Let us start by taking a look at the visual appearances of the foams. In order to be able to compare the methods on equal terms, the same initial foam has been used in the initial comparisons.

6.1.1 Local relaxation

Using the parameters listed in Table 4, 1700 frames of the same foam were run using the local and quasi-global approaches. The results are visualised in Figure 6.1. The dual mesh has been removed from the figures, but is of course still present as the underlying data structure holding neighbour information and information about, which cells are on the boundary including properties for physical variables.



(a) i = 10, 445 cells **(b)** i = 500, 198 cells **(c)** i = 1000, 131 cells **(d)** i = 1700, 102 cells

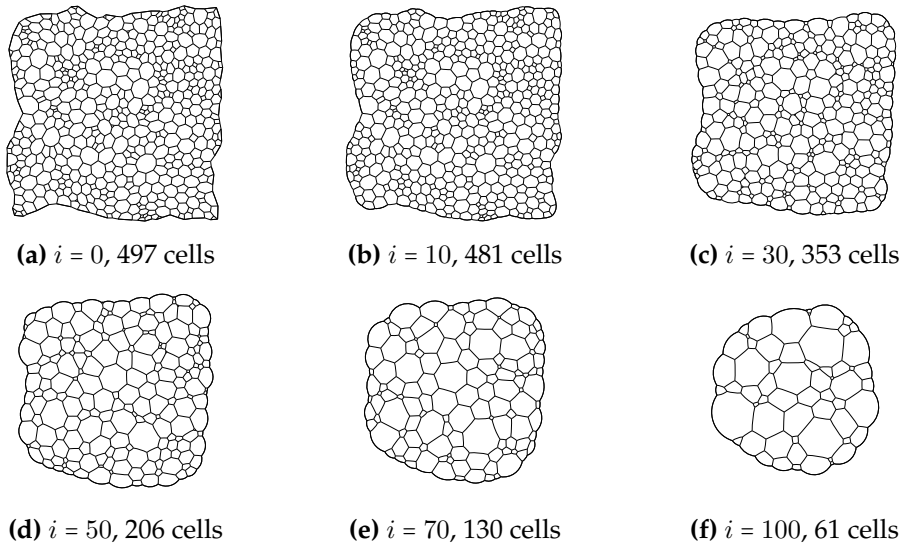
(e) i = 10, 444 cells **(f)** i = 500, 196 cells **(g)** i = 1000, 127 cells **(h)** i = 1700, 95 cells

Figure 6.1: A comparison between the local (top row) and quasi-global (bottom row) approaches with 497 cells at $i = 0$.

A video of the local approach in action with 1700 frames using the same foam as in Figure 6.1 is launched when clicking **here**, whereas the quasi-global version of the same foam is launched when clicking **here**. YouTube has an option of doubling the playback speed, which can sometimes help the human eye identify the long time changes more easily.

6.1.2 Global relaxation

It turned out that the global approach was numerically quite unstable. Adjusting parameters such as γ, κ and Δt would play an important role, as to whether the global Jacobian was invertible or not, even if it had full column rank at all times. The ratio between the three listed parameters greatly affects, how well-conditioned the Jacobian is, since the eigenvalues of the Jacobian can range from very large to very small, when having γ's, κ's and Δt's that do not play well together in this context, since the parameters apart from having physical significances also scale and prioritise the importance of surface tension (governed by γ) with respect to diffusion (governed by κ and Δt). This caused problems when trying to invert the Jacobian, even when using a singular value decomposition. Thus, through adjusting these parameters additionally, the parameters were adjusted to $\gamma = 0.5$ and $\Delta t = 0.005$, while κ was kept fixed. Figure 6.2 shows some of the 100 frames of the same foam as for the local approach.



(a) $i = 0$, 497 cells **(b)** $i = 10$, 481 cells **(c)** $i = 30$, 353 cells

(d) $i = 50$, 206 cells **(e)** $i = 70$, 130 cells **(f)** $i = 100$, 61 cells

Figure 6.2: The global foam over 100 frames. $\gamma = 0.5$ and $\Delta t = 0.005$.

Rather stunningly, the foam evolves much quicker - in a smooth fashion. Turning up γ and Δt will always result in a foam that coarsens more quickly, but with the cost of lower accuracy. Intrigued by these results, I tried using the same parameters with the local and quasi-global methods. But they gave very bad results in terms of smooth movements across the foam, most likely due to the bias of the order in the local method and the blindfolded local updates of the variables in the quasi-global method, where each junction does not know, what the updates for the variables on neighbour junctions will be.

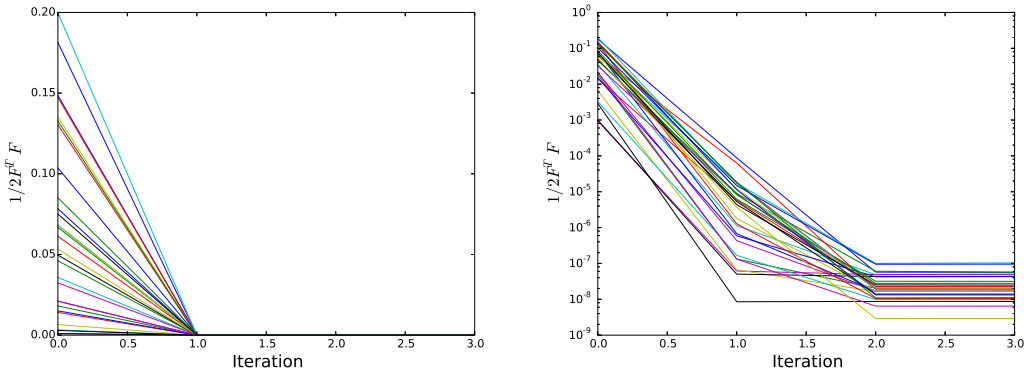
6.1.3 Convergence

When evaluating the convergence of the numerical methods, it really only makes sense to evaluate the relaxation processes. The idea of evaluating the convergence

is to evaluate the numerical method. But in the process of coarsening the foam, the target areas are updated for each frame i, and topological processes are performed. Thus, the objective function in the optimisation problem is updated all the time, and the number of junctions used in calculating the merit function also changes as a result of the topological operations.

By evaluating the relaxation only (the k-loops), the convergence analysis is performed on a section of the simulation, where the objective function is constant.

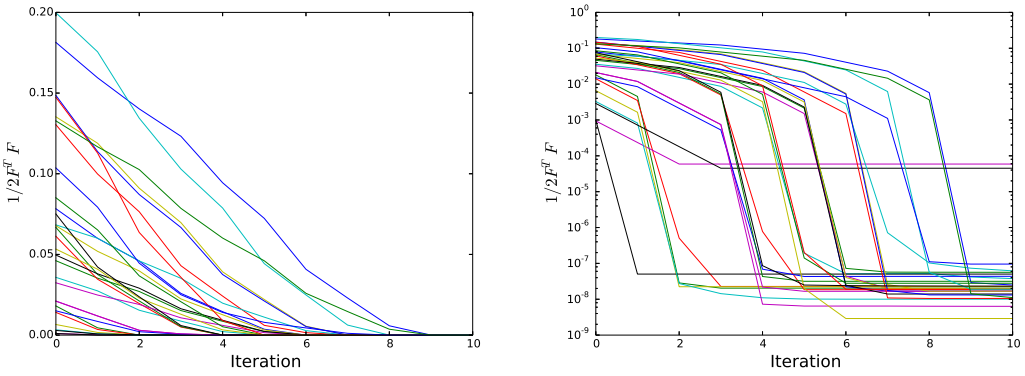
Figure 6.3 shows the convergence of 30 junctions in the local relaxation, looping over the iterator k with no backtracking, corresponding to $\tau = 1$. Obviously, the merit function converges at one iteration, since the system of linear equations is solved exactly. Looking at Figure 6.3a, this indeed seems to be the case. But looking at the same plot with a logarithmic y-scale as plotted in Figure 6.3b, it becomes clear that there are small numerical differences, with the merit function ranging between approximately 10^{-7} to 10^{-4}. But after 2 iterations, they are all below 10^{-6}.



(a) Normal scale **(b)** Logarithmic y-scale

Figure 6.3: The merit function of 30 junctions over 3 relaxation iterations in the foam at $i = 2$ with no backtracking.

Turning on the backtracking, the convergence of the merit function becomes highly dependent on the particular geometry if the junction being relaxed. Each relaxation iteration k will still reduce the merit function, but at different rates, as Figure 6.4a shows. The convergence is of course slower than with no backtracking, with the backtracking ensuring that the foam still maintains correct geometrical shapes and physical properties. Plotted against a logarithmic y-scale, one can see that the different junctions in a sense group into different shapes of convergence, with some converging quickly, whereas others start slowly, but then quickly drop dramatically after some iterations. This plot also justifies having $k_{max} = 10$, since after 10 iterations, all the tested junctions are below 10^{-6} apart from a couple, which seem to have converged around 10^{-4}, where the backtracking keeps the junction variables from being updated any further.



(a) Normal scale **(b)** Logarithmic y-scale

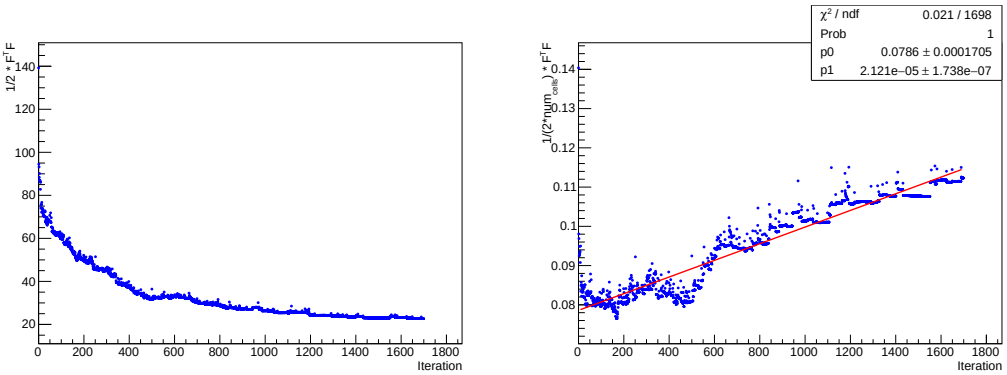
Figure 6.4: The merit function of 30 junctions over 3 relaxation iterations in the foam at $i = 2$ with backtracking.

With local relaxation, the merit functions for the junctions in the foam is thus reduced a lot, since they are allowed to relax to a temporary equilibrium, if the backtracking allows them to do so. In the case of the junctions in Figure 6.4, even the junctions, where the backtracking limits the reduction in the merit functions, they are still reduced to being below 10^{-4}. Summing over all junctions, the total merit function must therefore also be reduced similarly. One could argue that by having a large k_{max} with all junctions being relaxing on equal terms to a temporary equilibrium at each frame, one would at each frame i in general expect a constant normalised sum of the merit function over all the m junctions (normalising with respect to the number of junctions),

$$\sum_{i}^{m} \frac{1}{2m} \cdot F_i^T F_i = c$$

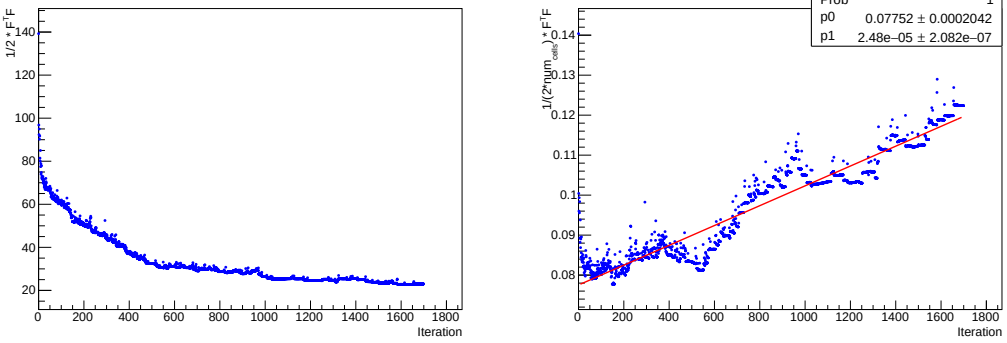
before updating the target areas and performing topological operations.

Figure 6.5 shows the summarised merit function over 1700 frames. The plots on the left are not normalised, while the plots on the right are normalised with the number of junctions at that particular frame i. The non-normalised plots show a clear fall in the merit function. The global approach is a bit still standing in the initial iteration, but plummets to a much lower merit function after about 10 frames. The idea to normalise is to smoothen out the falling number of junctions m caused by the topological operations. A linear fit is performed to the normalised plots. The hypothesis that these plots should be constant seems not to hold. The fit is very poor. The fit is done without errors on the data points, and with the additional oddly shaped data, one can disregard the χ^2 in the statistics box of the plots. The uncertainties on the slope p_1 make it nowhere near probable, that $p_1 = 0$, since in all three plots, this value is several standard deviations from being zero. In some sections, there seems to be a constant trend, but in general, the local and quasi-global seem to increase the normalised merit function over the number of frames, whereas the global approach seems more constant, if ignoring the initial values around 0.13.



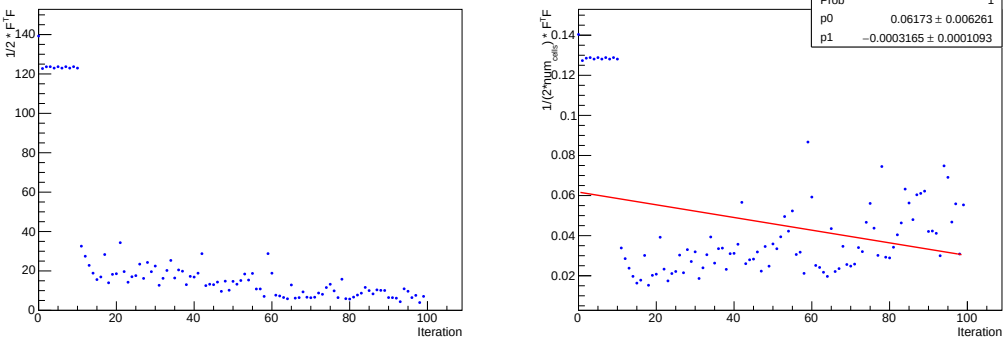
(a) Local, merit function

(b) Local, norm. merit, $R^2 = 0.8977$



(c) Quasi-global, merit function

(d) Quasi-global, norm. merit, $R^2 = 0.8931$



(e) Global, merit function

(f) Global, norm. merit, $R^2 = 0.0789$

Figure 6.5: Summarised merit functions for the three different approaches on the same foam, but over 1700 frames for the local and quasi-global approach and over 100 frames for the global approach. The left column shows the sums of the merits for each junction in the foam, while the right column show the same data, but with each data point normalised with the number of junctions in the foam for the given frame i fitted with a linear fit, $f(x) = p_0 + p_1 \cdot x$.

6.1.4 Cell area scaling

As introduced by (2.19), the average cell area is expected to scale proportionally with time as a power law. Figure 6.6 shows a linear fit to the 1700 frames of the local and quasi-local simulations. The errors on the data points are estimated as

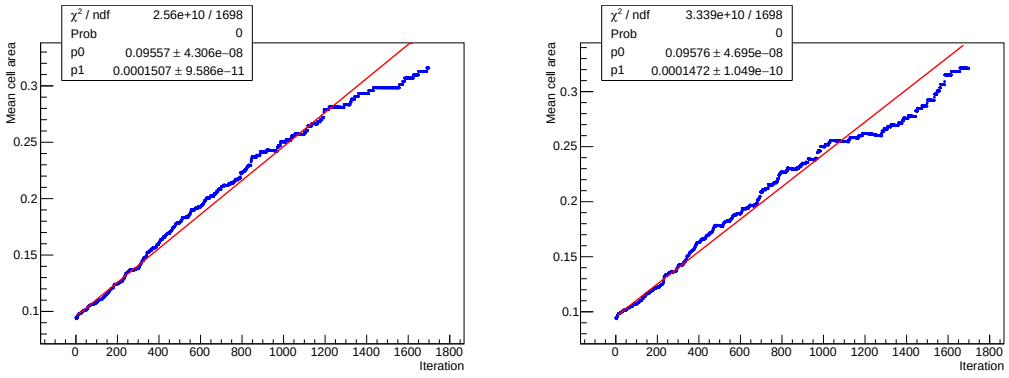
the uncertainty on the mean σ_μ for each frame as

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}} ,$$

where σ is the standard deviation on the mean and N is the number of data points used in the calculation of the mean.

Although the fit statistics are poor in both cases, there seems to be a clear linear trend in the beginning up until about 1000 frames. From there on, there seems to be a drop in the mean cell area compared to what the trend shows in the frames before that. In fact, the local approach seems to be more consistent with being linear than the quasi-global approach.

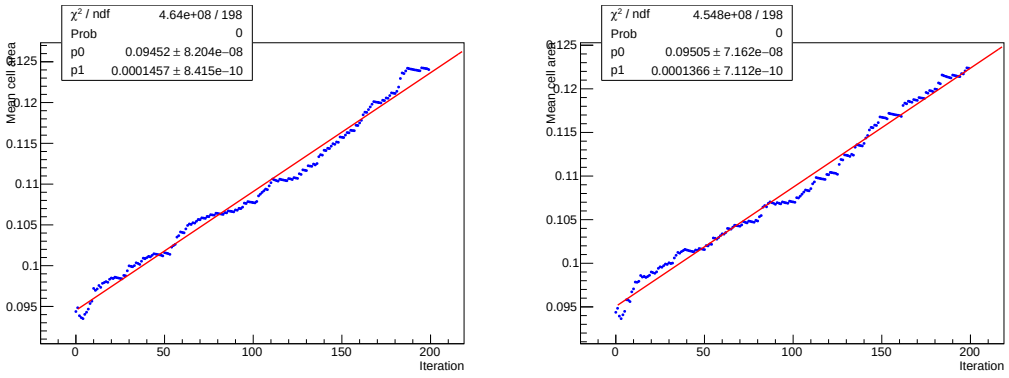
How come? The advantage of the local method is that at each relaxation step for some junction, the surrounding foam is exactly as it seems. The surrounding pressure and the positions of the surrounding junctions are, as they appear. And for the next junction to be relaxed in the process of sweeping through the foam, the situation is the same. As for the quasi-global method, when solving the numerical model at each junction, the surroundings are as they were at the start of the frame, before any junctions were visited. Thus, all junctions are updated on the basis of the same foam, and each junction does not know about the updates of the neighbouring junctions. This way, junctions bordering a certain cell may suggest updates to for instance the cell pressure of that particular cell in totally different directions, which one may see as a disadvantage. The advantage of this, however, is that there is no bias in the order of which the iterator visits the junctions.



(a) Local, $R^2 = 0.9711$ **(b)** Quasi-global, $R^2 = 0.9555$

Figure 6.6: Linear fit to the mean area over 1700 frames of the local and the quasi-global approaches.

When doing the fit on the initial 200 frames, the fit becomes better as shown in Figure 6.7.

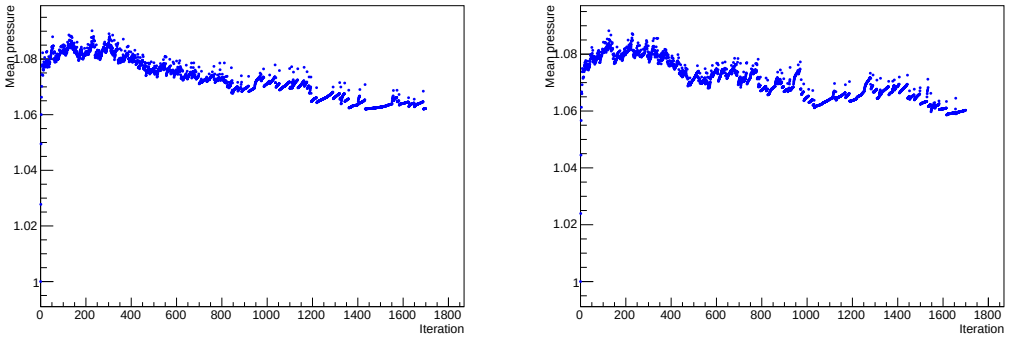


(a) Local, $R^2 = 0.9861$ **(b)** Quasi-global, $R^2 = 0.9880$

Figure 6.7: Linear fit to the mean area over the initial 200 frames of the local and the quasi-global approaches.

One explanation of the poorer proportional behaviour at latter frames may be that the pressure becomes lower on average as shown in Figure 6.8. The pressure is uniformly initialised as 1.0 in the foam and the world pressure outside the foam is set to 0.95. As the coarsening progresses, the diffusion outwards to the surrounding world causes the mean pressure inside the foam to drop ever so slightly, if the size of the foam is not reduced accordingly (again a case of the relationshop between γ and κ). Thus, the target areas used in the numerical model, which are calculated using von Neumann's law (2.14), will become comparably smaller over time, thereby slowing down the coarsening of the foam. As a result, the scaling results in this thesis might not be so surprising after all. One of the limitations of the model used in [Weaire and Hutzler, 2001] was that it used periodic boundary conditions. This gave very nice scaling behaviours, but omitted the outside world.

One way to possibly improve and test the scaling behaviours would be to initialise the pressure inside the foam as the same as the world pressure. Section 6.2 will deal with this.



(a) Local **(b)** Quasi-global

Figure 6.8: Development of the mean of the pressure in the cells over time.

6.1.5 Pressure

In order to get an idea of how the pressures distributes throughout the foam, a gradient colour map is added to the visualisation routine. Blue indicates cells of low pressure, whereas red indicates cells of high pressure. Purple indicates pressures in between.

Initially, the pressure is uniformly distributed as shown in Figure 6.9 shows.

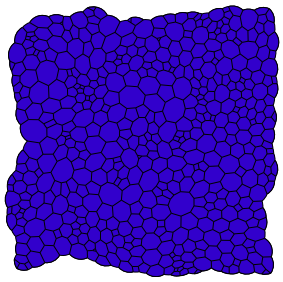
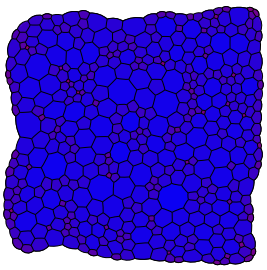
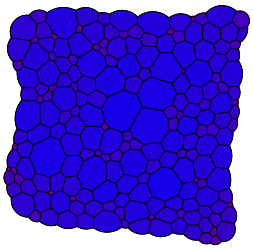


Figure 6.9: Initial configuration and uniform pressure distribution.

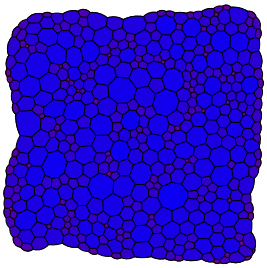
As the coarsening process evolves, the diffusion causes the cell pressures to change. Figure 6.10 shows the local and quasi-global methods at frame 10 and 500, respectively. Larger cells have lower pressure and small cells have high pressure. The small cells with higher pressure will shrink even more, with gas diffusing to neighbour cells. As [Kelager, 2009] also observed, it seems that chains of cells with high pressure seem to form. The same is the case with the low pressure cells.



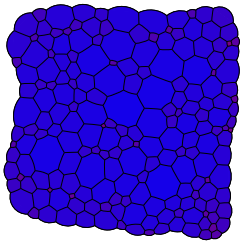
(a) Local, $i = 10$, 445 cells



(b) Local, $i = 500$, 198 cells



(c) Quasi-global, $i = 10$, 444 cells

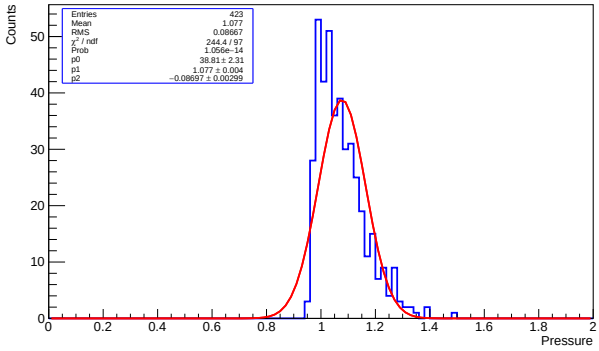


(d) Quasi-global, $i = 500$, 196 cells

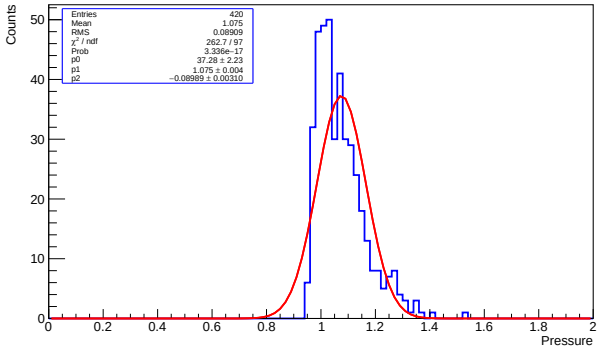
Figure 6.10: A comparison between the pressure distributions in the local and quasi-global method at the same stages and the same foam. The red colour indicates high pressure, where as blue indicates lower pressure.

But how does the pressure distribute, quantitatively speaking? Before looking at the data, let us for a moment think about, which probability density function the pressure would represent, if any? The distribution must be related to von Neumann's law. Assuming that Plateau's first rule of equilibrium holds, (2.15) holds. The area derivatives of 6-sided cells is zeros. There is a certain symmetry, since the area derivative of a 5-sided cell has the same magnitude, but with negative sign, as that of a 7-sided cell, the same with 4- and 8-sided cells and so on, though the symmetry breaks below 2-sided cells, since 1-sided and 0-sided cells don't exist. The second moment μ_2 (2.16) describes the variance of n-sided cells in the foam. And with the symmetry in the vicinity around $n = 6$, one could assume a Gaussian distribution around the mean of pressures in 6-sided bubbles with a variance of μ_2. Figure 6.11 shows Gaussian fits to the pressure distributions at 30 frames for the three methods. The fits do not fit particularly well. First of, there are no pressures below the world pressure, which is 0.95. This breaks with the idea of total symmetry. At first glance, the global fit has the highest probability and the lowest χ^2 to number of degrees of freedom ratio, suggesting a better fit. Also rather interestingly, the pressure values themselves are a lot higher in the global approach, explaining the much more rapid evolution as seen in Figure 6.2.

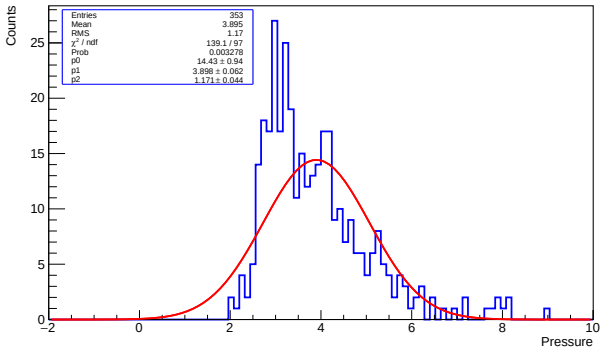
Section 6.3 will redo this analysis with the quasi-global approach on a larger foam with more cells.



(a) Local



(b) Quasi-global



(c) Global

Figure 6.11: Gaussian fits to pressure distributions after 30 frames on the same foam as used in Figures 6.1 and 6.2. Note the much larger pressures in the global case. The Gaussian is parsed to the fitting function as $f(x) = p_0 \cdot e^{-\frac{(x-p_1)^2}{2 \cdot p_2^2}}$.

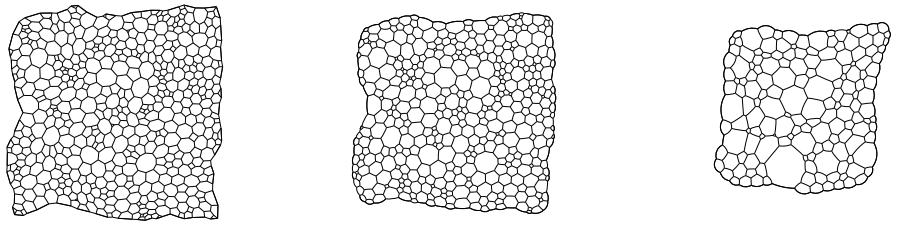
6.2 Experimenting with the boundary - equal pressure in foam and world

So far, the initialisation has used an initial uniform pressure distribution with value 1.0 inside the foam, and a constant world pressure of 0.95. This is a choice of initialisation that one is free to make. But what if the pressure inside the foam was initialised to be uniform and have the same magnitude as the world pressure?

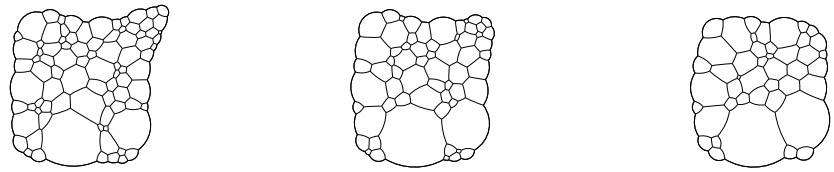
Imagine the previous initialisation with higher pressure inside the foam and lower on the boundary. Intuitively, there are two aspects to consider when trying to predict the contraction or expansion of the foam. One is diffusion, and the other is the pressure difference between the foam and the world and the motion of the foam boundary governed by the ideal gas law.

Imagine the following though experiment: If no diffusion is allowed, and if one keeps a constant temperature, the ideal gas law tells us that the foam will expand and still have the same number of gas atoms inside, correspondingly lowering the pressure to match that of the outside world, if the pressure is higher in the foam than in the world. This corresponds to squeezing a balloon and thereby increasing the pressure. Letting go of the balloon, and it will expand to its original shape with the pressure inside the balloon returning to its original value getting closer to that of the surroundings. But whether it will match the pressure of the surroundings, strongly depends on the surface tension of the rubber of the balloon. Staying in the balloon analogy, one could imagine blowing up a balloon, but only so much that the rubber does not stretch. This way, the pressure inside the balloon and the outside pressure would pretty much match. Blowing more air into the balloon would increase the surface tension, which would work against the expansion of the incoming air and correspondingly increase the pressure. Touching it with a needle thereby breaking the surface tension, and the balloon will bang. How does this correspond to the foam thought experiment? It teaches us the role of the surface tension. This way, the surface tension γ works against any expansion, while the diffusion, governed by the constant of permeability of diffusion κ, lets air pass out to the outside world. These work along each other: The higher the pressure difference between the foam and the world and the larger the value of κ, the more diffusion will occur from foam to world, and the foam will shrink. But the higher the pressure difference, the ideal gas law dictates that the more the foam will want to expand, given a low value of γ. A high value of γ, on the contrary, acts more like a balloon or, to use a different analogy, a corset.

Figure 6.12 shows the evolution with the quasi-global method over the same number of frames and on the same initial foam as in Figure 6.1 (bottom row for the quasi-global method), but with the same pressure outside and inside the foam at $i = 0$. Thus, all films are straight in Figure 6.12a. Even with the lower pressure difference between foam and world, the coarsening is more rapid than Figure 6.12. This might seem surprising, but with the reasoning above, there is an explanation for this.



(a) $i = 0$, 497 cells **(b)** $i = 10$, 423 cells **(c)** $i = 500$, 158 cells



(d) $i = 1000$, 87 cells **(e)** $i = 1300$, 63 cells **(f)** $i = 1700$, 44 cells

Figure 6.12: Quasi-global method on foam with initially 497 cells at $i = 0$. The pressure on the boundary is initialised as the same as inside the foam.

How do the statistics differ from the ones for the quasi-global method in Section 6.1? Figure 6.13 shows a fit to the mean area over 1700 frames, comparable with the case with different initial world pressure and foam pressure for the quasi-global method in Figure 6.6b.

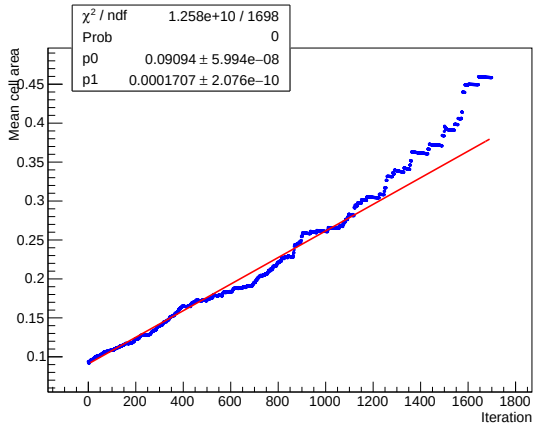


Figure 6.13: Linear fit to mean area of quasi-global method over 1700 frames. The world pressure and foam pressure is initialised to having the same value. $R^2 = 0.9309$

As before, the scaling looks to be quite good within the first approximately 200 frames. After 1000 frames, the fit does not seem fitting for either the case with equal initial world pressure or the case with lower initial world pressure. But whereas Figure 6.6b showed the data from the foam to be below the fit, Figure 6.13 shows an opposite behaviour with the data being above the fit. The explanation to both deviations must be that the boundary begins to affects a relative amount of cells more, as the time progresses. As the coarsening evolves, the mean cell area grows and the foam shrinks, and as a result, the boundary effects will affect the foam

more and more, because the outside world with its constant pressure behaves differently than the bubbles inside. It makes sense, that the cell areas in Figure 6.6b grow slower with time, since as Figure 6.8b shows, the mean pressure gradually decreases. And the lower the pressure differences, the lower the diffusion. The explanation for the opposite behaviour in Figure 6.13 shall be found in the pressure differences. Figure 6.14 shows a histogram of the pressure distribution of the pressure in the foam after 1400 frames, around where the areas grow more rapidly. The world pressure is 1, and as one can see, some cells have a lower pressure than that of the outside world thus resulting in an inward diffusion for low pressure cells on the boundary.

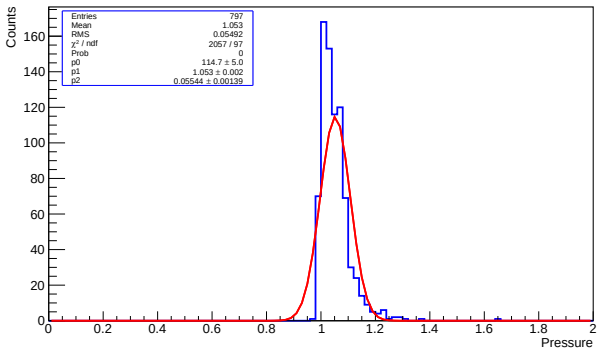
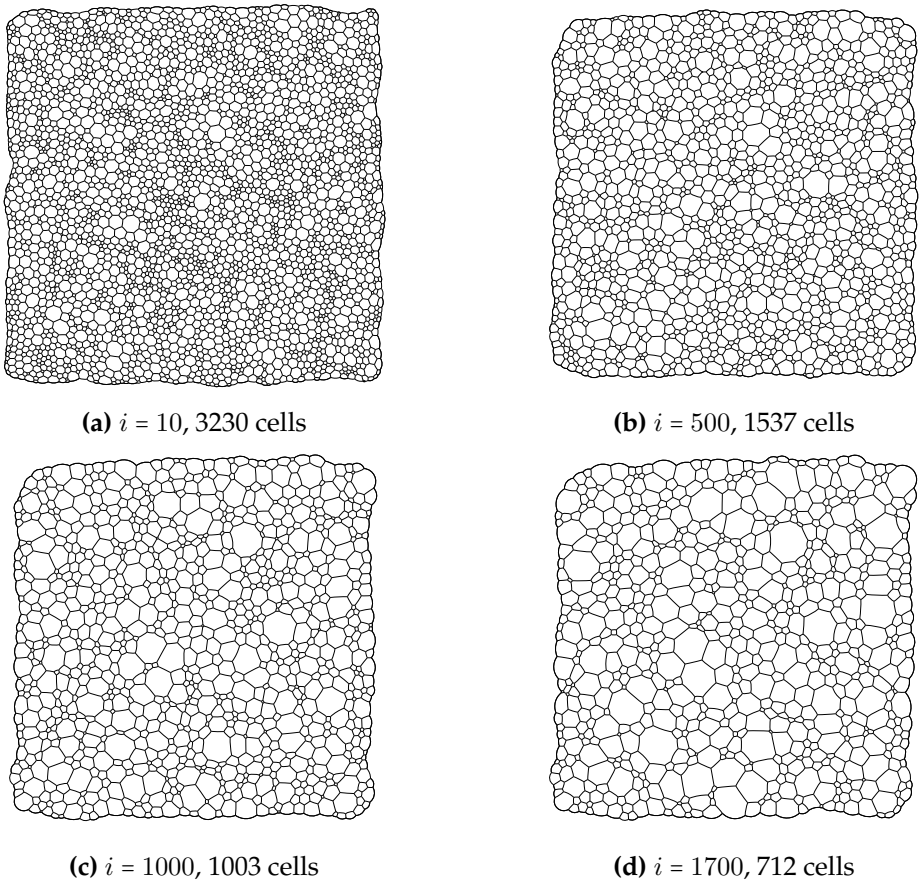


Figure 6.14: Pressure distribution after 1400 frames. The world pressure is 1. Note, how some cells have lower pressure than the outside world.

6.3 Applying the quasi-global method to a larger foam

The three different methods have different advantages each. The local and quasi-global methods are very alike, apart from the way they update the variables. The local approach is biased in the order, it visits and relaxes the junctions. The quasi-global does not depend on the order of the iterator, but the junctions don't know about what updates the other junctions will do. In terms of the statistical results, they are difficult to distinguish. The global method is very sensitive to initial parameters and very slow at the very initial frames, but then works very quickly, and it captures larger amounts of couplings between junctions.

Because of the instability of the global algorithm and the bias of the local, the quasi-global method is chosen to be tested on a larger foam. Again, we initialise the pressure as 0.95 on the boundary and a uniform pressure of 1.0 inside the foam. Figure 6.15 shows some frames of an initial foam of 3362 cells.



(a) i = 10, 3230 cells

(b) i = 500, 1537 cells

(c) i = 1000, 1003 cells

(d) i = 1700, 712 cells

Figure 6.15: The quasi-global method on a larger foam, with 3362 cells at $i = 0$.

6.3.1 Cell area scaling

Having a larger foam, one would expect better scaling behaviours, since the relative amount of boundary cells is smaller. Figure 6.16 shows the mean area over 1700 frames. It definitely looks better than with the smaller foam in Figure 6.6b with a higher R^2 value and a smaller χ^2-value, although this is still very high. But the χ^2-value is not that telling in this sense, since we are not investigating data from a given probability density function and how they distribute as such, but fit a trend line to some simulation data. As before, when the boundary effects begin to take over, the mean cell areas begin to deviate from the fit line. Of course, by "deviate", I realise that these points of course also affect the parameters of the trend line, but they nevertheless exhibit non-linear behaviours in this regime above the approximately 1200 frames.

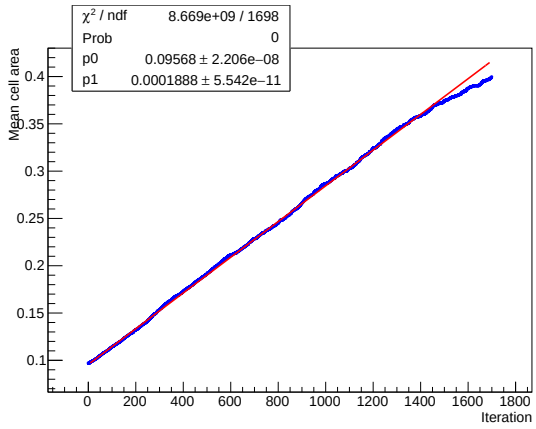


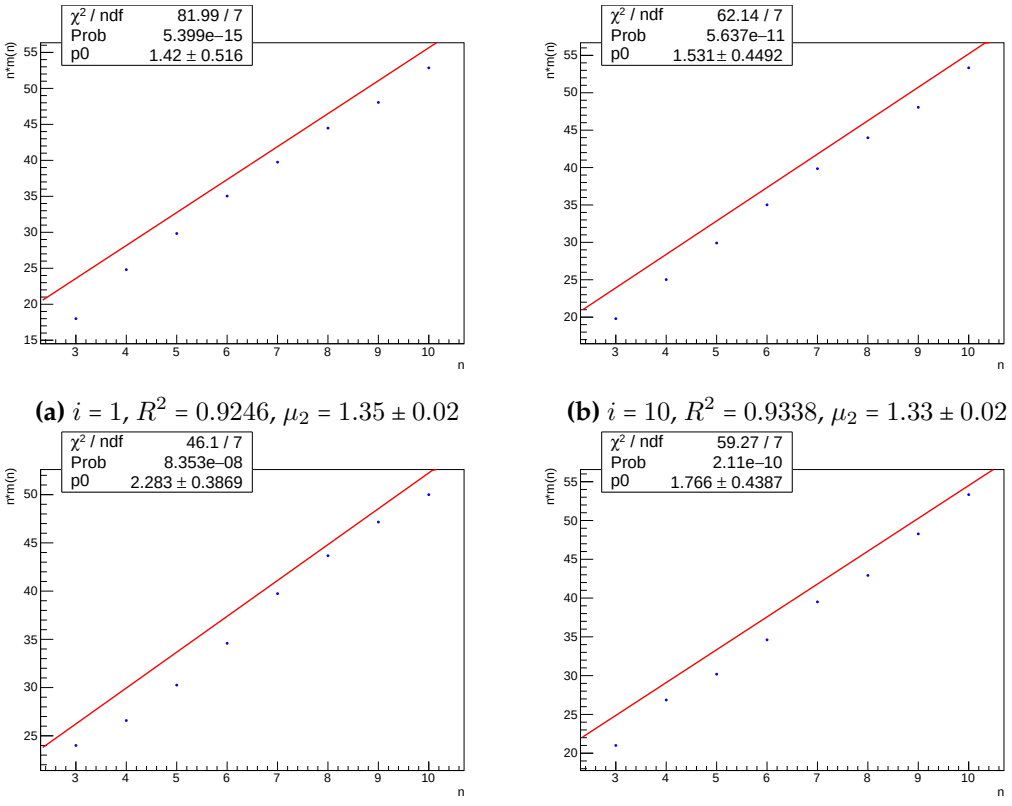
Figure 6.16: Scaling of the mean cell area of a foam with an initial number of 3362 cells. The mean area is expected to scale linearly with time. $R^2 = 0.9976$

6.3.2 The Aboav-Weaire law

The Aboav-Weaire law relates the mean number of sides of neigbouring cells to an n-sided cell and is given in (2.17). The second moment μ_2 is a given parameter for the foam at a certain frame and is calculated using (2.16). Thus, only one parameter, namely a, is an unbound variable that one can fit. It is common to fit $n \cdot m(n)$, since we then get a linear relationship, which is easier to interpret visually. Multiplying (2.17) by n gives

$$n \cdot m(n) = (6 - a)n + 6a + \mu_2 \ .$$

Figure 6.17 shows a fit of Aboaw-Weaire's law for 4 different frames. Each frame has its own μ_2, which has been calculated in each case. The uncertainties on the μ_2's have been calculated by using error propagation on (2.16) with respect to the mean number of sides of the cells in the foam $\bar{n}$ at that given frame.



(a) $i = 1$, $R^2 = 0.9246$, $\mu_2 = 1.35 \pm 0.02$

(b) $i = 10$, $R^2 = 0.9338$, $\mu_2 = 1.33 \pm 0.02$

(c) $i = 500$, $R^2 = 0.9293$, $\mu_2 = 1.39 \pm 0.03$

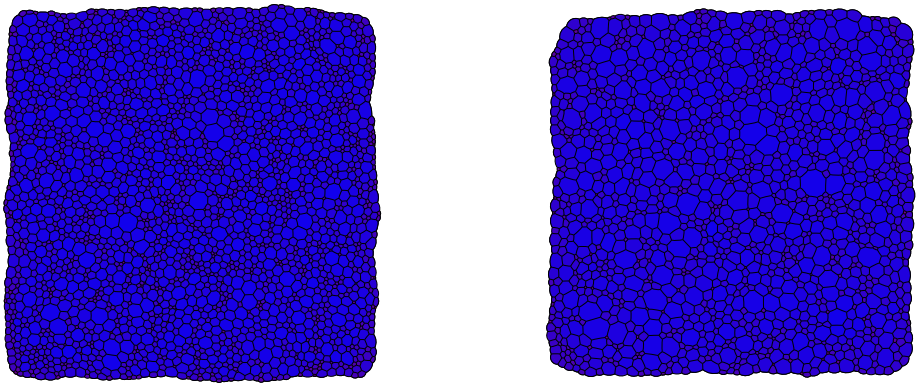
(d) $i = 1000$, $R^2 = 0.9299$, $\mu_2 = 1.58 \pm 0.04$

Figure 6.17: Fits of the Aboav-Weaire law at different frames. The second moment μ_2 is calculated using (2.16), and the errors on the second moment σ_{μ_2} have been calculated using error propagation on the formula with respect to $\bar{n}$.

As one can see, the fit lies above the data points, as it is not possible to fit a line through the points with only one degree of freedom in the shape of a. Also setting μ_2 as a free parameter solves this, but it makes no sense to do so, since μ_2 is a physical parameter. [Vedel-Larsen, 2010] experienced an opposite behaviour with the points lying above the line, but also experienced higher values of μ_2, in general. [Stavans and Glazier, 1989] did extensive studies on real foams and found a constant value of $\mu_2 = 1.4 \pm 0.1$, once the foam had passed the *transient* regime and had entered the *scaling* regime, where it was not dependent on initial conditions. At $i = 500$, the second moment in this simulation fits very well with this value, whereas for the other frames, the values are close, especially when considering the uncertainties. Within 2 standard deviations, one expects 95 percent of the cases to fall within these limits, when repeating the experiment. This simulation enters a scaling regime quite rapidly, and the second moment values in this simulation are close to the experimental values, until late in the coarsening process as seen in Figure 6.17d after 1000 frames. But as we saw for the area scaling, this is also where the boundary effects begin to interfere with the foam statistics.

6.3.3 Pressure

Section 6.1.5 treated the pressure distributions for the three different methods. Figure 6.18 visualises how the pressure distributes across the foam for two different frames of the large foam. Again, one can see, how small high pressure cells form chains, and that the smaller cells with fewer sides generally have higher pressure than the large cells with more sides.



(a) i = 100, 3210 cells **(b)** i = 500, 1537 cells

Figure 6.18: Visualisation of the pressure in the different cells in the foam. The colours range from blue (low pressure) across purple to red (high pressure).

Figure 6.19 sketches the pressure distribution in the foam after 30 frames. Starting at pressure 1.0, it is evident that the pressure on average has risen over the first 30 frames.

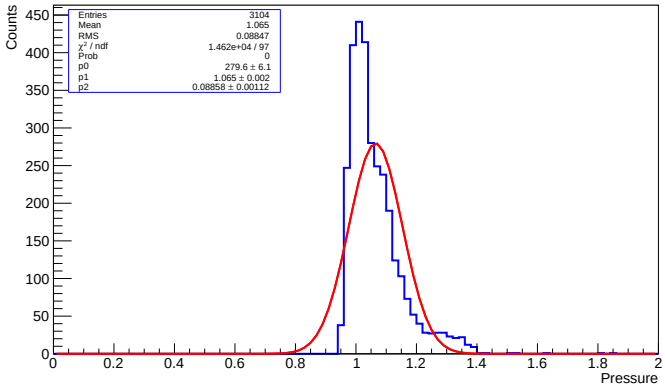


Figure 6.19: Pressure distribution after 30 frames, fitted with a Gaussian distribution function, $f(x) = p_0 \cdot e^{-\frac{(x-p_1)^2}{2 \cdot p_2^2}}$.

6.3.4 Number of cells

The number of cells gradually slows as the coarsening progresses. [Herdtle and Aref, 1992] suggests a power law relation between time and cell count, but in this

case a better fit was achieved with an exponential.

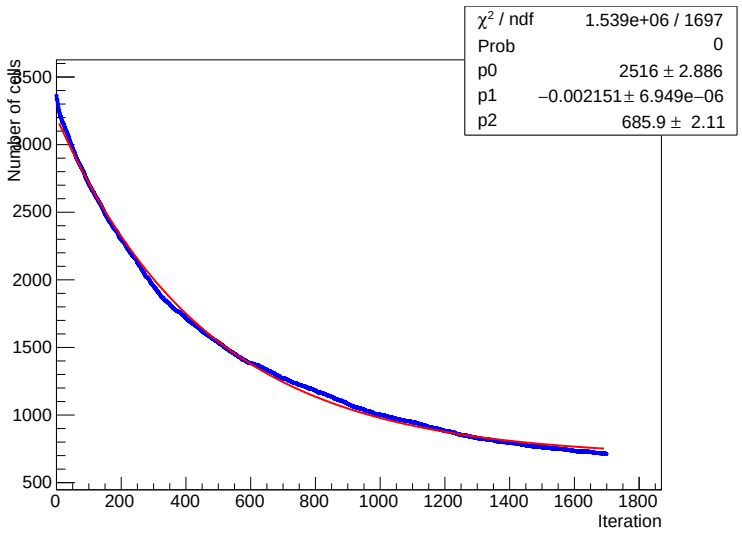


Figure 6.20: The number of cells as function of frame number fitted with an exponential, $f(x) = p_0 \cdot e^{-(x-p_1)^2/(2p_2^2)}$. $R^2 = 0.9978$

6.3.5 Surface energy

The surface energy is proportional to the film length in the foam, with the constant of proportionality being 2γ, with γ being the surface tension. If the second moment is more or less constant with time, one would expect the same scaling behaviours of the surface energy as that for the number of cells, since less cells equal less surface length. Hence, an exponential fit is made to the total surface energy as function of time as shown in Figure 6.21.

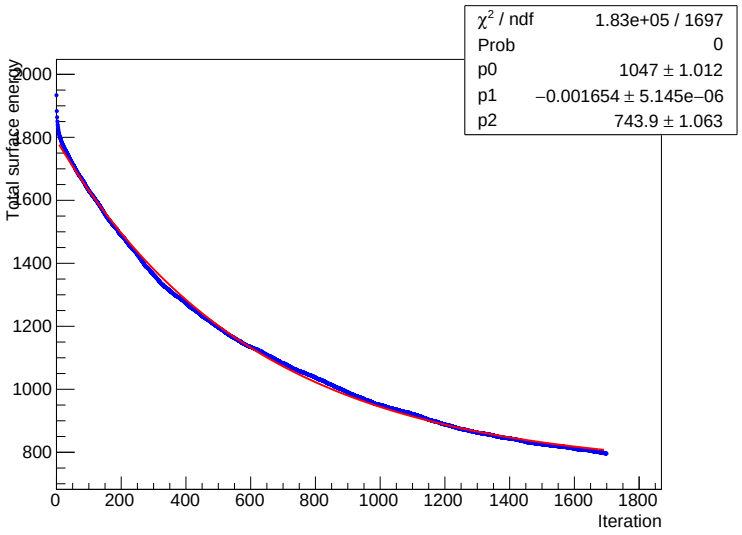


Figure 6.21: The surface energy as function of frame number fitted with an exponential, $f(x) = p_0 \cdot e^{-(x-p_1)^2/(2p_2^2)}$. $R^2 = 0.9985$

7 Conclusion

In this thesis, a successful implementation of a two-dimensional vertex model of dry foam has been implemented. The theoretical concepts have been explained in detail and partially derived to give an understanding of the underlying physics behind the behaviour of foams. An improved dual mesh representation of the boundary has been introduced thus making the simulation more stable. Combined with a more comprehensive foam initialisation, which after the construction of a Voronoi network through a Delaunay triangulation, additionally equilibrates the foam to its initial form, the simulation has been less sensitive to initial conditions than previous authors have experiences, by giving the user more control over the simulation.

Three different approaches were successfully introduced, the local, quasi-global and global method. The latter showed interesting features, but was too sensitive to initial conditions and parameters, where as the two others were very similar, but with their own advantages.

Statistical analyses have been made in order to validate the physics of the foam simulations. Some data fitted very well over several hundred iterations, for instance the area scaling over time, until the boundary effects became too large. The Aboav-Weaire law showed slight discrepancies, although the the variation in number of sides per cell, the second moment, agreed very well with real foam experiments.

Generally, it seemed that the free boundary, which was not implemented in most of the previous work cited in this thesis, affected the statistics somewhat. Not that the statistics were wrong, but it was harder to compare to other simulations using periodic boundary conditions. Where the boundary effects were smaller, scaling behaviours agreed well with theory.

The speed of the simulation has not been the main focus in this thesis, since the physical correctness and the robustness of the simulation was prioritised. All the simulations were run on my own laptop, but especially the quasi-global method is very well suited for paralellisation and multithreading with potential massive cuts on runtime.

7.1 Future work

Using this thesis as the foundation for future work, there would be many interesting extensions and improvements to make:

- Parallelise the existing code of the quasi-global method to make it run faster. This would allow the user to sample much larger foams at much shorter times. This would allow for larger statistical analyses, enabling for an even better understanding of the simulation and the dynamics of foams in the simulation.
- Go from a dry foam to a wet foam. This introduces the concept of drainage as well as additional physical descriptors such as the border pressure, the

pressure in the liquid between three films at a Plateau junction. The films are still circular in the two-dimensional case.

- Extension from two to three dimensions. This is not a simple matter, as the surface films in a three-dimensional foam are not spherical. [Weaire and Hutzler, 2001] mentions a foam evolver using a tessellation method representing the surfaces as a refined triangular grid. This enables discretised, but accurate models of the film surfaces.
- An improved global method for two-dimensional dry foams. The global method in this thesis showed interesting features. It captures more physics, but lacks in the numerics. If an improvement of this approach could be done, it would open up for a whole new mindset in the field of foam simulations. One could for instance combine the local and global approaches: one could combine slicing up a very large foam into sub-foams built on a global model and do parallel calculations on these. This would reduce the sizes of the global Jacobian matrices corresponding to the number of sub domains, while still keeping the global approach on the sub-foams. The sizes of the sub domains could be depending on the amount of disorder, for instance the second moment μ_2, in that particular part of the foam.

References

[Berg et al., 2008] Berg, M. d., Cheong, O., Kreveld, M. v., and Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition.

[Brochu and Bridson, 2009] Brochu, T. and Bridson, R. (2009). Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing*, pages 2472–2493.

[Da et al., 2016] Da, F., Hahn, D., Batty, C., Wojtan, C., and Grinspun, E. (2016). Surface-only liquids. *ACM Trans. on Graphics (SIGGRAPH 2016)*.

[Desbrun et al., 2008] Desbrun, M., Kanso, E., and Tong, Y. (2008). *Discrete Differential Forms for Computational Modeling*, pages 287–324. Birkhäuser Basel, Basel.

[Elcott and Schröder, 2005] Elcott, S. and Schröder, P. (2005). Building your own dec at home. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA. ACM.

[Hales, 2001] Hales, T. C. (2001). The honeycomb conjecture. *Discrete & Computational Geometry*, 25(1):1–22.

[Herdtle and Aref, 1992] Herdtle, T. and Aref, H. (1992). Numerical experiments on two-dimensional foam. *Journal of Fluid Mechanics*, 241:233–260.

[Isenberg, 1978] Isenberg, C. (1978). *The Science of Soap Films and Soap Bubbles*. Dover books explaining science. Dover Publications.

[Janiaud et al., 2007] Janiaud, E., Weaire, D., and Hutzler, S. (2007). A simple continuum model for the dynamics of a quasi-two dimensional foam. *Colloids and Surfaces A: Physicochemical and Engineering Aspects*, 309:125–131.

[Kelager, 2009] Kelager, M. (2009). Vertex-based simulation of dry foam (master's thesis).

[Kermode and Weaire, 1990] Kermode, J. and Weaire, D. (1990). 2d-froth: a program for the investigation of 2-dimensional froths. *Computer Physics Communications*, 60:75–109.

[Kincaid and Cheney, 2002] Kincaid, D. and Cheney, E. (2002). *Numerical Analysis: Mathematics of Scientific Computing*. Pure and applied undergraduate texts. American Mathematical Society.

[Kirillov and Turaev, 2007] Kirillov, A. and Turaev, D. (2007). Foam-like structure of the universe. *Physics Letters B*, 656(1–3):1 – 8.

[K.Nakashima et al., 1989] K.Nakashima, T.Nagai, and K.Kawasaki (1989). Scaling behavior of two-dimensional domain growth: Computer simulation of vertex models. *Journal of Statistical Physics*, 57:759–787.

[Neatorama, 2013] Neatorama (2013). Neatorama, image of honeycomb. `http://static.neatorama.com/images/2013-05/honeycomb-bee-hexagon.jpg`. [Online; accessed 03-February-2017].

[Nocedal and Wright, 2006] Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization, Second Edition*. Springer.

[RWTH Aachen, a] RWTH Aachen, C. O. Compiling OpenMesh. `https://www.openmesh.org/Daily-Builds/Doc/a00030.html`. [Online; accessed 30-September-2016].

[RWTH Aachen, b] RWTH Aachen, O. D. G. E. Deleting geometry elements. `https://www.openmesh.org/Daily-Builds/Doc/a00060.html`. [Online; accessed 05-November-2016].

[RWTH Aachen, c] RWTH Aachen, O. P. T. Python Tutorial. `https://www.openmesh.org/Daily-Builds/Doc/a00036.html`. [Online; accessed 30-September-2016].

[Spencer et al., 2017] Spencer, M. A., Jabeen, Z., and Lubensky, D. K. (2017). Vertex stability and topological transitions in vertex models of foams and epithelia. *The European Physical Journal E*, 40(1):2.

[Stavans and Glazier, 1989] Stavans, J. and Glazier, J. A. (1989). Soap froth revisited: Dynamic scaling in the two-dimensional froth. *Phys. Rev. Lett.*, 62:1318–1321.

[Vasileiou et al., 2015] Vasileiou, V., Granot, J., Piran, T., and Amelino-Camelia, G. (2015). A planck-scale limit on spacetime fuzziness and stochastic lorentz invariance violation. *Nature Physics*, 11(4):344 – 346.

[Vedel-Larsen, 2010] Vedel-Larsen, B. K. (2010). Quasi-static simulation of foam in the vertex model, master's thesis. University of Copenhagen.

[Weaire and Hutzler, 2001] Weaire, D. and Hutzler, S. (2001). *The Physics of Foams*. Clarendon Press.

[Weaire et al., 2002] Weaire, D., Hutzler, S., Cox, S., Kern, N., Alonso, M., and Drenckhan, W. (2002). The fluid dynamics of foams. *Journal of Physics: Condensed Matter*, 15(1):S65.

[Weaire and Kermode, 1983] Weaire, D. and Kermode, J. P. (1983). Computer simulation of a two-dimensional soap froth. *Philosophical Magazine Part B*, 48(3):245–259.

[Yu et al., 2015] Yu, Y., Wu, Q.-b., Chen, M.-h., and Suleman, M. (2015). Robust construction of minimal surface from general initial mesh. *Applied Mathematics-A Journal of Chinese Universities*, 30(2):227–244.

Appendix A Installing OpenMesh for Python

It took me a while to get OpenMesh up and running for Python. There is a documentation on OpenMesh's webpage [RWTH Aachen, c] [RWTH Aachen, a], but it is rather limited. As a result, here is a little guide as to how I got it to work. First off, one needs to install Boost.Python, a C++ library used to bind C++ libraries to Python. Since I am a Mac user, I used one of their package managers to install Boost.Python. I use Homebrew, but one could use Macports or other similar package managers as well. In the case of more installations of Python on one's laptop, it is important that Boost.Python is linked to the correct version of Python. In my case, I installed it against the pre-installed version of Python (/usr/bin/python. Usually, one simply types "python"), but one can write any executable of Python to the right of the equals sign

```
$ brew install boost-python --with-python=/usr/bin/python
```

With Macports, it would look like

```
$ sudo port install boost +universal +python27
```

Assuming that this step installed Boost.Python correctly, it is time to install OpenMesh. If one has Subversion working on one's laptop, it is easy to check out and thus download the libraries. Navigate to the desired folder, where the OpenMesh files should be located and type

```
$ svn co http://www.openmesh.org/svnrepo/OpenMesh/trunk OpenMesh
```

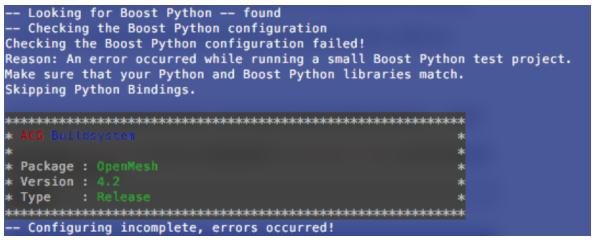
Now, navigate to the OpenMesh folder and create a folder, could be named "build". Navigate to this

```
$ cd OpenMesh
$ mkdir build
$ cd build
```

Now, run cmake again remembering to specify the correct version of Python. Once again, one should change "/usr/bin/python", if one wants to build OpenMesh against another version of Python.

```
$ cmake .. -DPYTHON_EXECUTABLE:FILEPATH=/usr/bin/python$
```

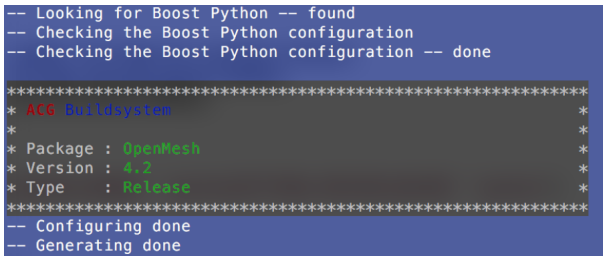
Now, one may come accross the following error message:



This could be caused by many things, but if one typed the correct Python executable in the `cmake`-command and linked Boost.Python to the same version of Python, it could be caused, if one runs Python 3.x. Then, the OpenMesh-type has to be specified in the `cmake`-command

```
$ cmake .. -DOPENMESH_PYTHON_VERSION=3
```

If this solved the problem, the error message should have changed to



Assuming all possible disasters thus far have now been dealed with, one shold type

```
$ make
```

Now, the guide on the webpage [RWTH Aachen, a] included fewer steps than shown above, and it stopped after this step without the usual `make install` step, but this proved crucial for me. Thus, type

```
$ make install
```

To build OpenMesh's documentation, write

```
$ make doc
```

Now, open the Python that OpenMesh is build against and add the folder, where the `openmesh.so` file is located in the build folder:

```
$ /usr/bin/python
>>> import sys
>>> sys.path.append("/Users/me/bin/OpenMesh/build/Build/python")
```

Fingers crossed. This worked for me. To test it, type (in the same Python session as the above)

```
>>> from openmesh import *
>>> mesh = TriMesh()
```

This should not produce any errors. If indeed it does not, OpenMesh has been built correctly.