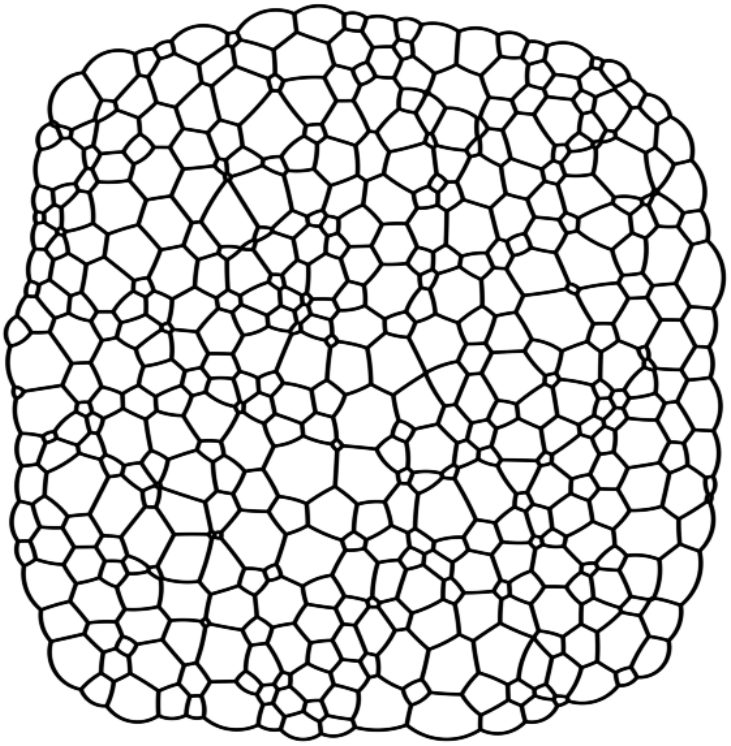
Quasi-static Simulation of Foam in the Vertex Model

Bue Krogh Vedel-Larsen
buekvl@gmail.com



Master of Science Thesis

University of Copenhagen, Denmark
Department of Computer Science

October 29, 2010

Abstract

In the field of computer physics, foams have proved to be a challenging phenomenon to simulate. The fragile nature of foam combined with heterogeneous materials interacting in an inherently unstable and continuously evolving mass requires highly specialized numerical methods. This thesis presents the Vertex Model, a quasi-static soap-film model of two-dimensional dry foam that focuses on capturing the physical behavior of foams with low liquid content. A detailed examination of the Vertex Model is made and statistics, gathered from a CUDA implementation of the model, are presented and compared to experimental data to demonstrate the validity of the model. The thesis contributes a novel approach to the computational mesh in the form of the dual of the foam, as well as two hard constraints on the quasi-static relaxation process. It is further discussed how the Vertex Model can be parallelized and how modern GPU's can be used to render foams in a fast and flexible manner.

Contents

Chapter 1

Introduction

In the field of computer physics, foam is a subject that has not received much attention. While foam models certainly exists, foam has never received the interest of liquids, rigid- and soft-body dynamics, and smoke has enjoyed. This is surprising given the proliferation of foams in nature, from the foam-like behavior of biological cells, to foam as a material in the building industry. We encounter foams constantly: Soap foam when we wash our hands, milky froth in a cappuccino served to-go in a styrofoam cup, foam padding on our furniture, and foam forming when we pour a glass of beer.

In this thesis I will examine one of the existing models of foam and attempt to improve certain aspects of it. I will briefly cover the physics of foam, then make a detailed description of how we can make a mathematical model, the Vertex Model, that captures the behavior of foam which we can then turn into a computational model, ready for computer simulation. The presented model focuses on a physical correct modeling of the behavior of foam, while the visual aspects, with light interference patterns and scattering, have been ignored. The philosophy has been to create a foam model that is physical correct first and then we can add external dynamics and better visuals afterwards.

The model presented here will be strictly two dimensional. Intuitively we can image that we are observing a foam that is caught in the very thin gap between two glass plates that are almost touching. This condenses the physical theory involved into a form that are directly observable by physicists in a controlled experiment. Therefore two dimensional foams are comparatively better described than three dimensional foams. At the same time, restricting the model to a two dimensional plane greatly simplifies the mathematics involved.

1.1 Previous work

In physics, foams have been extensively studied and documented. Giving a complete treatment of the field would be beyond the scope of this thesis, so I have focused my research on the advances in computer simulation of foam, a much smaller subject area. I have seperated the following into two main categories: Foam simulation and foam rendering. Simulation are (physics-based) methods for generating foams that can be compared to experimental measurements of real foam samples. The work in this thesis falls into this category. The second

category are foam models which focuses on the visual aspects of cells and foams. Many of these have no direct physical foundation but mimicks the observed behavior of foam.

1.1.1 Physics-based foam simulation

In [26] and [27], Weaire and Kermode presented a computer simulation of dry foam which is the basis of the method presented in this thesis. Their model included all of the theoretical foundation which this method builds on, such as the Plateau laws, the Laplace-Young law, von Neumann diffusion, and T1 and T2 topological processes. However, they used periodic boundaries and they showed no foams with more than 100 cells. Interestingly, their prime motivation was grain growth in polycrystalline metals and glasses.

In [28], Weaire and Rivier surveyed and described many different space-filling random two dimensional patterns, among these soap froth. It was shown how soap froth is just one in a class of cell based patterns that all follow some similar statistics. Several useful theoretical tools was mentioned in passing, including graph duality, cell network surface energy, and Aboav's law for the relation between the number of sides in a cell and the mean number of sides in cells surrounding it.

Wejchert *et al.* [29] sought to investigate the asymptotic behavior of foam as time $t \to \infty$, but they realised that the simulation created by Weaire and Kermode [26] was too time consuming. They therefore created a Monte Carlo method on a fixed hexagonal grid to simulate grain growth by minimizing the surface energy of the system.

Kawasaki *et al.* [12] were the first to present a vertex-based model to simulate grain growth in polycrystaline metals. Their model used a straight-line approximation of films and a velocity/friction model for the movement of junctions. Using their model they simulated foams of up to 24.000 cells.

Brakke [3] created a computer program, which he called the Surface Evolver, for minimizing the energy of surfaces subject to constraints. He worked with surface integrals in a soap-film model of two dimensional surfaces embedded in three dimensional space, but could handle arbitrary dimensions. An example of its use shows a simulation of grain growth in a metal as a two dimensional Voronoi network of cells with periodic boundaries, a problem that is closely related to dry foam. However, the focus of the Surface Evolver is, in the authors own words, breadth of application, not optimal treatment of specific problems; it can not replace the more specialized simulations as presented in this thesis.

Aste [1] investigated evolution of froths under geometrical and topological constraints in the statistical model. He found that, while froths can reach a stable equilibrium state, they will only do so at very high temperatures ($\sim 10^{16}$K for soap froth). At "normal" conditions froths will remain unstable and evolve, increasing the size of large cells at the expense of small cells, which will shrink and disappear.

Fuchizaki *et al.* [10] presented a three dimensional vertex-based model for grain growth, based on the Kawasaki [12] model. To ameliorate the problem of accurately modeling the behavior of three dimensional film surfaces, virtual vertices was used to piece-wise approximate curved surfaces.

Marsh *et al.* [15] was the first to present a vertex-based model that used film curvature to more accurately model the behavior of junctions. They demon-

strated a method for calculating film turning angles that is very close to the one used in this thesis, however, they still used the velocity/friction model of Kawasaki *et al.* [12].

Neubert and Schreckenberg [18] presented a summary and comparison of six models for simulating soap foam, including a vertex-based model and five topological event-driven models. The vertex-based model was based on straight-line films.

In [30] Weygand *et al.* updated the Kawasaki [12] model by adding virtual vertices between junctions. In this way they could increase the accuracy of the straight-line model and could remove some of the simplifications of the model used by Kawasaki *et al.*

In [31] Weygand *et al.* extended the Kawasaki [12] model from two dimensions to three dimensions, by adopting the Fuchizaki [10] model, and investigated the differences between two dimensional and three dimensional simulations. It was found that, while similar, there was key differences and that two dimensional simulation slices can not, in general, be used to replace a three dimensional simulation.

Icart and Arqués [11] focused on the interaction of light with foam, but as part of their work they presented an model for soap froth simulation based on minimizing the distance between pairs of spherical cells, subject to the radii of the cells. As foam dynamics such as coarsening and topological rearrangment was not handled, it can be argued that the model was not of foam but of cell clusters. Though the cells where three dimensional, they only simulated a single layer of cells. See also Section 1.1.2.

A different approach to cell simulation was presented by Ďurikovič [23] where individual cells was simulated as a mass/spring system. By creating explicit models for cell coalescence he was able to form clusters of up to six cells. The model did not handle foam dynamics such as topological rearrangement and was better suited to simulating and visualizing single cells and small clusters.

In [32] Zheng *et al.* presented a simulation method based on fluid simulation over a grid, with the Navier-Stokes equations in a continuous multiphase fluid simulation. A Regional Level Set was used to capture the films of cells, though much smaller than the grid size. A semi-implicit surface tension model was used to render the simulation unconditionally stable. They conjectured that the model would be robust enough to handle foams as well as bubble clusters.

Kelager [13] presented a comprehensive treatment of the Vertex Model for simulating dry foam. Apart from covering several theoretical and practical issues in implementing the vertex model, the main contribution was the introduction of a free surface boundary condition through the World Bubble construct. This thesis can be viewed as a continuation of the work performed by Kelager.

Barrales Mora [2] built upon the Kawasaki [12] model and the Weygand [30] model to develop a vertex-based simulation for grain growth and used this to investigate the behavior of grain growth in a magnetic field.

1.1.2 Foam rendering

Icart and Arqués [11] presented a model based on light interference with the films of soap froth. Using this technique they created realistic color interference patterns on cell surfaces, both planar and spherical.

Cleary *et al.* [5] used a particle based Smoothed Particle Hydrodynamics (SPH) model to simulate the behavior of bubbles forming and rising in a liquid. Bubbles was approximated with spherical particles that were forced upwards through a buoyancy force and to avoid overlapping they added a spring-based repulsive force between bubbles. While simplistic, their approach captured many of the traits of foam.

Kück *et al.* [14] presented a method where wet foams could be simulated without an explicit representation of the interior of bubbles. Instead the foam was approximated as a mass of spheres held together by the interaction of replusive, attrative, and external forces. A ray-tracing algorithm was used to model the interaction of light within the foam to reproduce the visual appearance of a mass of bubbles.

1.2 A few words on vocabulary

The subject of this text is a model of the behavior of *foam*. In reading the litterature concerning the matter of foams, one will often come across the word *froth* used to describe the same phenomenon, e.i. gas or heterogenous matter seperated into bubbles or cells. Indeed, there seems to be little separation between foam and froth. Merriam-Webster Online[1] defines foam as "a light frothy mass of fine bubbles formed in or on the surface of a liquid or from a liquid," while froth is defined as "bubbles formed in or on a liquid." Weaire and Hutzler [25] notes that froth is generally uncomplementary which can explain why foam is preferred by many.

Moving closer to the foam, it consists of *cells* or *bubbles*, with cell the more generic term. Since my main focus is on liquid (dry) foams, I will use cell and bubble interchangeably, though bubble is not an appropriate term when dealing with heterogenous, non-gassious materials. Cells are matter trapped in a thin *film* or *membrane* of some other matter, such that the film forms the interface between the different materials or states. Some uses the term *boundary* instead, but I prefer the term film, as boundary is easily confusable with other aspects of the simulation and membrane usually denotes stretched biological tissue. Where two films intersects, a *(Plateau) border* is formed. In three dimensional (3D) foams, films are two dimensional (2D) surfaces, and borders are one dimensional (1D) lines that forms in the intersection of films. In 2D foams, films are 1D and so films and borders collapses into the same 1D line, so I shall make no distinction between the two. Where three or more borders intersects a *junction* is formed.

In the following chapters I shall delve deeper into what exactly foams, cells, films, and junctions are.

1.3 The findings of this thesis

This thesis builds in large part on the work begun by Kelager in his thesis "Vertex-Based Simulation of Dry Foam" [13]. Over the cause of the writing, almost all parts of Kelagers work was scrutinized and rewritten into a much improved implementation of the foam simulation. Apart from the topics covered

[1] http://www.merriam-webster.com

here, many other minor additions and improvements will be detailed throughout the thesis.

Starting from the bottom, the simulation was rewritten to use the dual of the foam as the computational mesh. This significantly changes many of the algorithms used and results in a simpler, more straight-forward and more stable implementation. Section 3.2 details the new computational mesh. Rewritting the computational mesh also allowed me to modify the model to include two-sided cells, a pathological case which no previous implementation has fully included. Section 3.6 has a thorough discussion of two-sided cells and how they are incorporated in the simulation.

In Kelagers implementation a simplistic, non-physical constraint called the "In-Bubble test" was used to ensure that the foam simulation did not enter an invalid state when junctions were relaxed. I have completely removed this constraint and examines two alternative, improved constraints: The Energy Decay Constraint and the Orientation Invariant Constraint. The first is a physically based energy measure, where we, by ensuring that the energy of the foam (locally) decreases, constantly keep the foam in a valid state. The second constraint is based on the geometry of the foam and limits the movement of junctions to always obey an rotational ordering invariant. The two constraints are detailed in Section 3.5. I show that, while both functional, the Energy Decay Constraint is too strong and the Orientation Invariant Constraint is not strong enough.

A final major improvement to the underlying model is that the algorithm used by Kelager to control when topological changes was allowed to happen to the computational mesh has been reworked. Kelagers method, while usable, was simplistic and insufficient. In Section 4.3 a thorough discussion is made of the implications of the algorithm for controlling topological operations and an improved algorithm is presented. Like the energy measure, the improved algorithm leads to an improved stability of the simulation.

In a novel approach, I have from the start incorporated parallelism in the simulator. The result is an implementation that, though not fully parallel, performs the computationally heavy relaxation process in parallel with a significant performance gain. Section 4.4 discusses the problems associated with parallelism and my solutions to making as large a part of the simulation as possible parallel. A CUDA implementation is used to demonstrate the viability of the method.

Using the demonstration implementation, I measure several essential statistics of different foams samples and compare them to statistics gathered experimentally from real foams. In Section 6 I show that the Vertex Model produces foams that are comparable to real foams.

Lastly, from a presentation point of view, I have developed a method for rendering the simulated foam using a programmable Graphics Processing Unit (GPU). While the rendered image is still a simplistic black-and-white line drawing, the speed and visual quality of the rendering is much improved. Chapter 5 is dedicated to the rendering of foams.

In this thesis my focus has been on the correctness of the foam model and the parallelism of the algorithm. As such, certain parts from Kelagers work has not been implemented. Most notable, all interactive elements have been removed, as has shear dynamics. The argument is that, while these dynamics are interesting from a presentation point of view, they are ultimately secondary until the simulation is completely correct, stable, and fast. I will claim that the removed elements can be added to my finished implementation with relative

ease and that they will benefit from the improved simulation.

Chapter 2

The Physics of Foam

In this chapter I will give an overview of the physics of foam as it applies to the Vertex Model of dry foam, though I will not attempt to account for all of the details of the topic. For an excellent introduction to the field of foam physics, Weaire and Hutzlers book [25] is easy and enjoyable to read.

A foam is a heterogeneous mass where two (or more) materials interact to form cells of one material between films of another material. A well-known example is soap foam where bubbles of air are trapped between films of water, with soap used as a surfactant[1] to decrease the surface tension of the water. Another example is the formation of crystals in polycrystalline metal alloys, known as grain growth. The model presented in this thesis most closely matches the behavior of soap foam and is often known as the soap foam model (or soap froth model) in the litterature.

When two cells touch, a thin (on the order of nanometers [25]) film of liquid is formed that separates the two cells and where three films intersects a Plateau border is formed. In a soap foam, the Plateau borders are capillary tubes, draining liquid away from the films [11]. The greater the amount of liquid in the Plateau borders, the greater the cross-section area of the borders are and we have a *wet foam*. As the amount of liquid tends to zero the Plateau border shrinks until it is of a negligable size in a *dry foam*. In this thesis I will assume a perfectly dry foam and we can therefore consider the Plateau borders infinitesimally thin and disregard them. In the following I will make a more precise definition of how the liquid volume of a foam is measured.

In the foams we will be considering in this thesis, cell scale is on the order of millimeters to centimeters. A dry foam on this scale is almost meta-stable and, if not subjected to external forces, will only break down slowly over a time scale of tens of minutes to hours [22, 25], in a process known as *coarsening* which will be discussed in Section 2.4. A wet foam will behave slightly different as the capillary suction of Plateau borders causes liquid to drain away from the films. A foam in a near meta-stable state is said to be in *equilibrium* as the internal forces caused by surface tension and pressure are balanced. A major part of this thesis, Chapter 3, is an investigation into a method for bringing a foam into an equilibrium state. A foam in equilibrium is immobile and thus the energy stored in the foam is entirely potential and proportional to the surface energy of the

[1] surface active agent

films in the foam. In Section 2.5 I will cover how to estimate the surface energy of a foam. Unless subjected to external forces we expect the energy of a foam sample to decrease monotonically over time as the foam evolves.

As a foam is subjected to gravity, the capillary tubes of the Plateau borders will drain liquid in the direction of gravity in a process known as *drainage*. The effect is that the top of the foam become dry while the bottom remains wet. We can therefore observe all the stages of a foam from wet to dry in a single foam sample. In this thesis I will make the assumption that the foam samples we are considering are a single, infinitly thin layer of cells on a plane perpendicular to the force of gravity. Gravity will therefore be uniform over the entire foam and we can disregard gravity and drainage in the model.

We will, unless otherwise specified, only be considering disordered, random foams. Ordered foams are uninteresting from a simulation point of view, as there will be no dynamics and the foam will be static. We will also only be considering mature foam samples and not cover adding gas into the foam. We will, in other words, not cover inflation of cells.

In a dry foam, under the given assumptions, the films of the foam will form a topological network where cells are enclosed by films and cells naturally pack themselves such that three cells forms a stable junction. We say that a foam is a *space-filling cellular structure* [28] because a foam will fill all available space with no holes. In Section 2.2 I will go into more detail on the rules which governs the formation of foam shapes. As the foam is subjected to internal and external dynamics, the shape of the foam will change as the topological network of films is reordered. In Section 2.3 I will introduce two topological operations which will form the basis for all foam dynamics. The most important internal foam dynamic of a dry foam is *diffusion*, which I will cover in Section 2.4. Finally, in Sections 2.5 and 2.6 I will cover how to approximate the energy of a foam and a few essential foam statistics.

2.1 Dry foam in two dimensions

A soap foam consists of some amount of gas and some amount of liquid. Formally we can formulate the amount of gas as the gas volume ratio

$$\Phi_g = N_V \overline{V}_c,$$

where

$$V_c = \frac{4}{3}\pi \left(\frac{d}{2}\right)^3$$

is an idealised volume of a cell c with effective diameter d where we assume cells to be spherical. The effective diameter takes into account that the film enclosing the cell can be of varying thickness and it may not be well-defined where one cell ends and another starts when cells are touching. N_V is the number of cells per unit volume.

Knowing the gas volume we can calculate the liquid volume ratio as $\Phi_l = 1 - \Phi_g$. In this thesis I will limit my attention to dry foam. A dry foam is one in which the ratio of liquid volume to gas volume is low ($\Phi_g \to 1$), while in a wet foam the ratio is high [25]. As the liquid quantity increases, the cells in the

foam are pushed apart until, in the extreme, the cells becomes free-flowing. The difference can be readily observed in carbonated beverages, where CO_2 bubbles rises freely and individually in the liquid but forms a closely packed head of foam on top.

I will also limit this thesis to foams in two dimensions (2D) on a flat plane. Informally, this matches the behaviour of foam trapped between two clear glass plates. As such, the foam has no thickness, only width and breadth. In the litterature, the physics of 2D foams are better understood than three dimensional foams, in large part simply because 2D foams are easier to observe and quantify in physical experiments.

2.2 The geometry of foams in equilibrium

Cells are formed when surface tension causes a small quantum of gas to be captured inside a thin film of liquid. We shall assume that the thickness of the film is negliable compared to the area of the cell, so that the thickness can be ignored and films treated as 1D lines. When subjected to stress, the bulk modulus of a foam is much greater than its shear modulus to a degree where we can consider the bulk modulus to be infinite [25]. We can therefore assume that the gas is an incompressible ideal gas, so that we may disregard viscous forces and inertial effects [26, 25]. Intuitively, when we apply pressure to a foam it is more likely to deform and move away perpendicularly to the direction of pressure in a shear effect, than it is to compress.

The pressure difference between cell-cell and cell-environment will cause films to curve with an arc radius given by the *Laplace-Young law* [26, 15, 25]

$$r = \frac{2\gamma}{\Delta p} \ , \tag{2.2.1}$$

where $\Delta p = p_i - p_j$ is the pressure difference of cells i and j and γ is the surface tension of the liquid medium. As $\Delta p \to 0$, the film separating the cells approaches straight as $r \to \infty$.

It is important to note that the preceding model of film curvature is an approximation: In an arbitrary polycrystalline foam the film curvature will, in general, not be the same at the two junctions and thus the films will not be circular [15]. However, in a soap foam model we can reasonably make the assumption that the thin, liquid films and the even pressure differences across the whole cell will cause the junctions to arrange themselves such that the curvutature becomes even and the films circular to minimise the energy over the film.

Joseph Plateau formulated three important laws about the behavior of foams in equilibrium from experimental observation [25]:

Equilibrium rule A1 For a dry foam, the films can intersect only three at a time, and must do so at 120°. In two dimensions, this applies to the lines which define the cell boundaries.

Equilibrium rule A2 Again for dry foams, 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 thetrahedral vertex

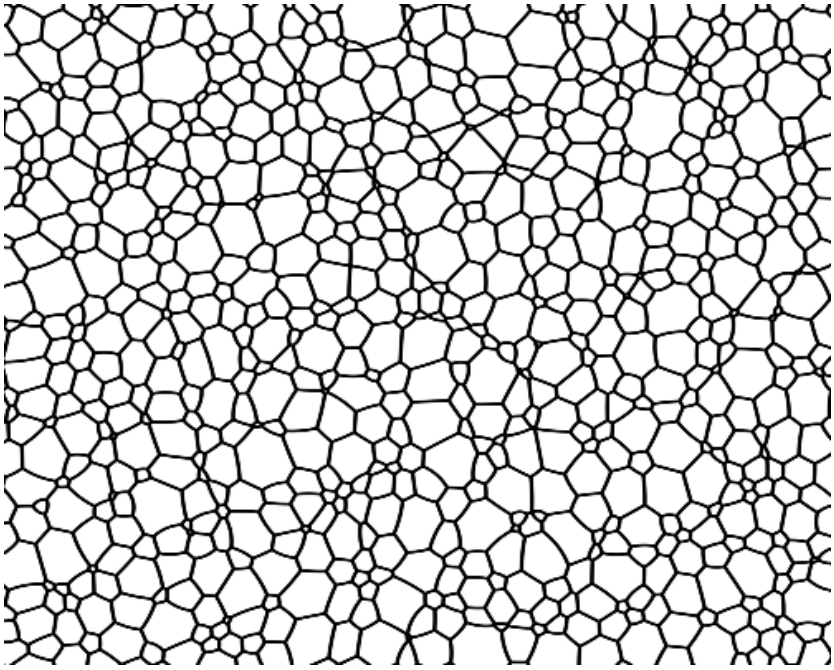


Figure 2.1: A section of a foam sample in equilibrium. Note how all junctions are formed in the intersection of exactely three films and how films curve to follow pressure differences.

is perfectly symmetric. Its angles all have the value $\phi = \cos^{-1}\left(-\frac{1}{3}\right)$, sometimes called the Maraldi angle.

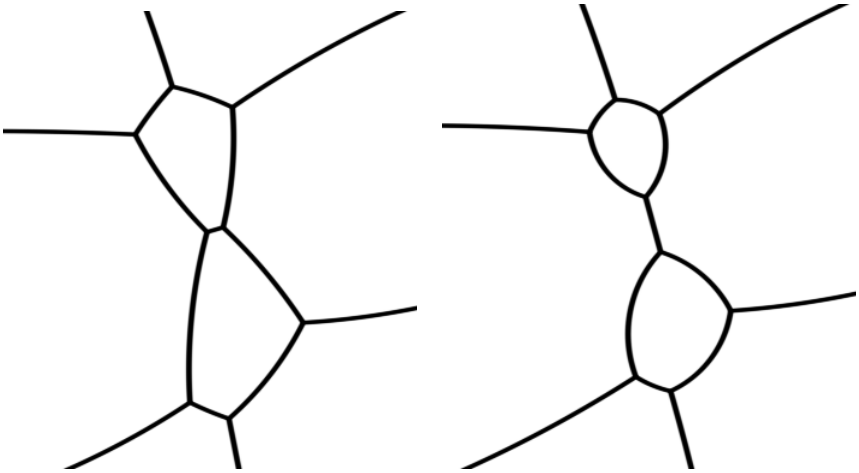
Equilibrium rule B Where a Plateau border joins an adjacent film, the surface is joined smothly, that is, the surface normal is the same on both sides of the intersection.

These are equilibrium rules in the sense that a foam that does not fulfill these rules is unstable and will collaps into a form where the rules hold. In the following we are only concerned with rule A1. Rule A2 only holds for 3D foams, while Rule B is specifically for wet foam. A1 can also be formulated as the *coordination number*, *cardinality* or *degree* of all junctions must be 3 [28].

Figure 2.1 shows a foam in equilibrium where all junctions are formed in the intersection of exactly three films. It is immediately apparent that not all film turning angles are 120°, but in an arbitrary foam sample, such as the one the shown section is a part of, it is not possible for all films to turn 120°. However, as the foam is in equilibrium we have reached a state where the foam is as close as possible to satisfy this. Only in an infinite, perfectly honeycomb-shaped foam, where each cell has six films, can the Plateau rule be satisfied completely.

2.3 Topological operations

As a foam move and evolve, topological changes in the network of films will cause the foam to change shape. All these changes can be rooted back to two



(a) (b)

Figure 2.2: A film is flipped in a T1 operation.

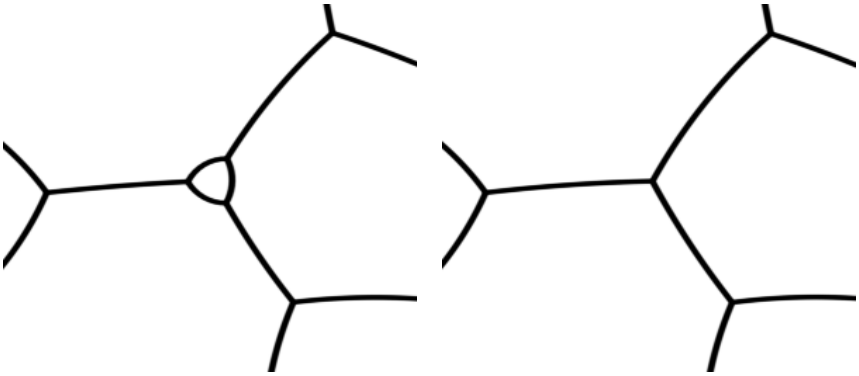
fundamental topological operations [26, 28, 25]. The first operation, which we shall call the T1 operation, happens when four cells change their combined connectivity as shown in Figure 2.2, often as a result of pressure differences pushing two junctions together, to form a single junction where four films intersect. This is an unstable state and the foam will collapse into a lower-energy state by separating the four-film junction into two three-film junctions. The second operation, the T2 operation, happens when three cells meet, causing a fourth cell to collapse as shown in Figure 2.3, typically as a result of coarsening (see below). All other topological processes can be recreated as a sequence of T1 and T2 operations.

T1 operations are violent, near-instantaneous changes to the foam topology, but in the brief time in which the four-film junction seperates into two three-sided junctions, none of our earlier assumptions about the exclusion of viscous forces and inertial effects are necessarely true [26]. However, for our use we can safely treat T1 processes as instantaneous.

Note how the Plateau A1 rule tie in with the observed behavior of T1 operations: Junctions that form at the intersection of three films are the only stable configuration; if more films intersects at a junction, the foam will collapse into a lower-energy state in a series of one or more topological changes.

2.4 Diffusion

Over time, the thin films between cells allows gas to move from one cell to another in a process known as *diffusion* [26, 25]. The pressure difference between two neighboring cells causes gas to be pressed through the shared film from the high-pressure cell to the low-pressure cell, causing the high-pressure cell to shrink and the low-pressure cell to grow. The same process will cause border cells to loose gas to the surrounding environment. Eventually, as shrinking cells loose volume, they will undergo a T2 process and disappear, leading to a macro-behavior where the foam appears to evolve from a detailed state with



(a) (b)

Figure 2.3: A cell is collapsed in a T2 operation.

many small cells, to a coarse state with a few large cells. This evolution process is known as *coarsening*.

Diffusion between two given cells sharing a film can be modeled in 2D as [25, 13]

$$\frac{\partial A}{\partial t} = -\kappa l(p_i - p_j) \ , \tag{2.4.1}$$

where $\frac{\partial A}{\partial t}$ is the area rate of gas transfer, l is the length of the shared film (in 3D the film area would be used,) $\kappa \in \mathbb{R}^+$ is the diffusion constant of the liquid which controls the transfer rate, and $p_{i,j}$ is the pressure of cell i and j respectively. (2.4.1) is derived from Fick's law of diffusion which states that the flux of material goes from regions of low concentration to regions of high concentration.

By combining (2.2.1) and (2.4.1) we get

$$\frac{\partial A}{\partial t} = -2\gamma\kappa\frac{l}{r} \ .$$

for a single film and

$$\frac{\mathrm{d}A}{\mathrm{d}t} = -2\gamma\kappa\sum_i \frac{l_i}{r_i} \tag{2.4.2}$$

for a cell, where i runs over the films of the cell and l_i and r_i^{-1} is the length and radius of the i-th film. For a cell with n films, where the tangents of films turn $\frac{\pi}{3}$ degree on account of Plateau's law, it holds that [25]

$$\sum_i \frac{l_i}{r_i} + n\frac{\pi}{3} = 2\pi \ . \tag{2.4.3}$$

Rearranging (2.4.3) yields the sum rule

$$\sum_i \frac{l_i}{r_i} = 2\pi\left(1 - \frac{n}{6}\right) \ . \tag{2.4.4}$$

Finally, by inserting (2.4.4) into (2.4.2) and rearranging we arrive at

$$\frac{\mathrm{d}A}{\mathrm{d}t} = \frac{2\pi}{3}\gamma\kappa\,(n-6) \ , \tag{2.4.5}$$

which is *von Neumann's law of diffusion.* Von Neumann's law elegantly states that, irrespective of pressure differences or film lengths and curvature, cells with more than six sides will grow and cells with less than six sides will shrink. Cells with exactely six sides are meta-stable and will neither grow nor shrink. From this and the Plateau rules we can realise that a perfect, infinite honeycomb structure, where all cells are hexagonal and all films turn $\frac{2\pi}{3}$, is a perfectly stable structure which will never change.

Von Neumann's law is a approximation that captures the macro-behavior of foam over time based on statistical measurements. In the following I will explore some other foam statistics.

2.5 Surface energy

A foam can be seen as a network of cells, where each cell has a constant, predetermined area, and the foam is in an state of equilibrium when the total surface energy has been minimized [28]. The surface energy of a foam is proportional the the area of the films in the foam. In 2D this becomes the length l of the films:

$$E_{\text{film}} = 2\gamma l \tag{2.5.1}$$

As a result we can estimate the total surface energy as

$$E_{\text{total}} = \sum E_{\text{film}} = 2\gamma \sum_{i \in E} l_i \ , \tag{2.5.2}$$

where E is the set of all films in the foam and l_i is the length of the i-th film. From this we can estimate the energy of a single cell as

$$E_{\text{cell}} = 2\gamma \sum_{i} l_i \ , \tag{2.5.3}$$

where i runs over the n films in the cell. As with any physical system, the energy in the system must be in balance. Therefore we must garantee that all changes made to the foam preserves or decreases E_{total}. In a later section we shall use this result to impose constraints on the process of equilibrating a foam.

Recall from 2.2 that Plateau's A1 rule dictates that the turning angle between any two adjacent films intersecting in a junction must be 120°. Equation (2.5.1) clearifies why this must be true. Let us consider a foam with uniform pressure. In such a foam all films will have radius $r \to \infty$ and so we can consider them straight. Examining a single junction $\mathbf{x}$, we see that the three films intersecting at the junction spans a triangle with the incident junctions $\mathbf{x}_i$, $\mathbf{x}_j$, and $\mathbf{x}_k$ at the corners, as illustrated in Figure 2.4. The energy of the three films $\mathbf{x} \to \mathbf{x}_i$, $\mathbf{x} \to \mathbf{x}_j$, and $\mathbf{x} \to \mathbf{x}_k$ are given by (2.5.1) and the combined energy of the junction $\mathbf{x}$ is

$$E_{\mathbf{x}} = E_{\text{film}}^{\mathbf{x}\to\mathbf{x}_i} + E_{\text{film}}^{\mathbf{x}\to\mathbf{x}_j} + E_{\text{film}}^{\mathbf{x}\to\mathbf{x}_k} \ .$$

It follows directly from the triangular inequality that $E_{\mathbf{x}}$ is least when $\mathbf{x}$ is inside $\triangle \mathbf{x}_i\mathbf{x}_j\mathbf{x}_k$ and the turning angles between the three films are all 120°. In other words, the foam is in a least-energy state when Plateau's A1 rule is satisfied for all junctions in the foam. This is the global energy minima of the foam.

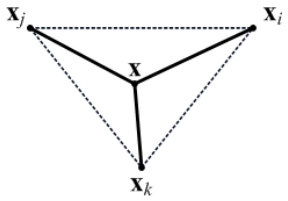


Figure 2.4: The three-junction neighborhood of a junction $\mathbf{x}$. The incident junction $\mathbf{x}_i$, $\mathbf{x}_j$, and $\mathbf{x}_k$ spans a triangle.

As the pressure in the foam becomes non-uniform and the radius r of films becomes finite, films will start to curve and we can, in general, no longer fulfill Plateau's A1 rule for all junctions. The foam will therefore reach an equilibrium state where E_{total} is minimized but not, in general, in the global minima.

2.6 Foam statistics

For a dry foam in 2D the mean number of films in a cell is

$$\overline{n} = 6 \tag{2.6.1}$$

which is known as *Euler's law* [26, 25] (or Euler's theorem.) It simply states that, on average, in an infinite foam sample, cells will have six sides. This results follows from Euler's equation

$$|F| - |E| + |V| = \chi$$

where $|F|$ is the number of faces, $|E|$ is the number of edges, and $|V|$ is the number of vertices in any 2D cellular structure. $\chi \in \mathbb{Z}$ is of order 1 and depends on the space in which the structure lives. For our use, a plane, $\chi = 1$. In a foam, as a result of Plateau's law, $|E| = \frac{3}{2}|V|$ and from this (2.6.1) follows by substitution.

We expect the distribution of films in a cell $p(n)$ to be symmetric and roughly gaussian, with mean $\overline{n} = 6$, and the second moment

$$\mu_2 = \sum_{i \in F} (n - \overline{n})^2 \, p(n) = \sum_{i \in F} (n - 6)^2 \, p(n) \ ,$$

where F is the set of cells, allows us to reason about the distribution [26, 25]. Finally, the *Aboav-Weaire law*

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

relates the number of films in a cell to the average number of films in surrounding cells [28, 22].

The Aboav-Weaire law relates to an observed behavior of foam, where small cells will cluster around large cells. This creates distinct patterns and the Aboav-Weaire law measures whether the foam exhibits these patterns or not.

2.7 The dry foam model

We can now summaries the preceding into a model for dry foam:

- Pressure differences between cells causes films to curve with a radius given by the Laplace-Young law (2.2.1).
- The Plateau laws limits the formation of foams to three-film junctions and film turning angles will tend to 120° to minimize potential energy in the foam.
- Topological changes are discrete occurances which will cause the foam to change shape to minimize the potential energy of the foam.
- The potential energy is proportional to the surface energy of the foam, which in 2D is the sum of the length of films.
- Over time, gas diffusion between cells will cause the foam to evolve as gas moves from small cells to large cells according to von Neumann's law (2.4.5).

In the next chapter I will develop this physical model into a mathematical model and then into a computational model which can be implemented on a computer.

Chapter 3

A Vertex-Based Model of Dry Foam

In the previous chapter we explored a physical model of foam. In this chapter I will transform that model into an algorithm for generating and evolving foams. The algorithm I will be using is known as a vertex-based model or simply the Vertex Model [15, 12]. I shall begin this chapter by giving a high-level description of the Vertex Model, then delve into the details.

In the Vertex Model we use a computational mesh to represent the foam. In each time step of the simulation we can move the junctions of the foam (for example through rheological processes or external forces), change the area of cells (for example through diffusion), etc., after which a quasi-static relaxation process is performed to bring the foam into a new, stable equilibrium state. Through-out all of these processes we must ensure that the Plateau laws are not violated.

Iterative simulations are generally the best choice for foam simulation, since the topological changes the foam must undergo during the course of the simulation can not be defined analytically without making gross assumptions [15]. In an iterative method, on the other hand, we can make running adjustments to the foam. Section 3.1 discusses an iterative method for relaxing a foam into an equilibrium state.

The quality and success of the Vertex Model is dependent on the underlying computational mesh. In Section 3.2 a novel approach to the computational mesh is detailed and it is shown how the topological operations are carried out on the computational mesh. Following that we cover in detail how to compute the essential properties needed by the relaxation process and boundary conditions are touched upon. Finally the complete model is presented in Section 3.7.

3.1 The quasi-static simulation

A foam in equilibrium must satisfy two properties: Each cell must have a given area and each junction must satisfy the Plateau laws [28], e.i. the incident angles must be 120°. To reach such an equilibrium there are two facets that can be adjusted: The coordinates of junctions and the pressure in cells.

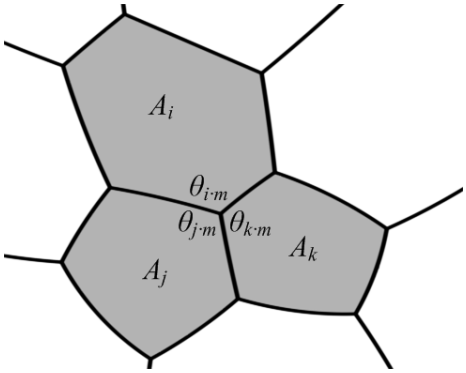


Figure 3.1: The cell area A and film turning angle θ of three cells surrounding a particular junction m.

The process of relaxing a foam can be posed as a root search problem where we must minimize the deviation of areas and angles [25]. Given n cells and m junctions, let

$$F(\mathbf{t}') = \mathbf{0} \ , \tag{3.1.1}$$

where $\mathbf{t}' = [p_1 \cdots p_n, x_1 \cdots x_m, y_1 \cdots y_m]^{\mathrm{T}}$ is the set of cell pressures and junction coordinates, be a function that measures the deviation of the foam from the state of equilibrium. We do not have an analytical solution to F, but we can employ Newton's method to iteratively find a root [19].

Before we can solve F we need to have an optimal foam state, such that we can define this state as the equilibrium state. We say that the foam is in equilibrium when it is in this optimal state. We therefore introduce the *cell area* A_i and *cell target area* A_i^τ, which are, respectively, the area and the theoretical optimal area of cell i. If the cell is free to evolve without other constraints, it will change it's pressure and junction positions to have area $A_i = A_i^\tau$. We also needs to satisfy the Plateau law, so we introduce the *film turning angle* $\theta_{i\cdot j}$ which is the angle between two films of cell i, intersecting in a junction j. We define $\theta_{i\cdot j} = \frac{2\pi}{3}$ if cell i has no films intersecting in junction j. Plateau's law dictates that if a cell is free to evolve without other constraints, $\theta_{i\cdot j} = \frac{2\pi}{3}$. Figure 3.1 shows cell area A and film turning angle θ. In the following I will omit the junction index from the film turning angle if the relevant junction is clear from context.

We can pose the preceding as constraints:

$$A_i^\tau - A_i = 0 \tag{3.1.2}$$

$$\frac{2\pi}{3} - \theta_{i\cdot j} = 0 \tag{3.1.3}$$

For now we will assume a method for calculating A_i and $\theta_{i\cdot j}$ exists. Using these

constraints we can define

$$F(\mathbf{t}') = \begin{bmatrix} A_1^\tau - A_1 \\ \vdots \\ A_n^\tau - A_n \\ \frac{2\pi}{3} - \theta_{1\cdot 1} \\ \vdots \\ \frac{2\pi}{3} - \theta_{n\cdot 1} \\ \frac{2\pi}{3} - \theta_{1\cdot 2} \\ \vdots \\ \frac{2\pi}{3} - \theta_{n\cdot m} \end{bmatrix} = \mathbf{0} \ ,$$

where n is the total number of cells and m is the total number of junctions. F will be of length $n + n \cdot m$. The Jacobian of F is

$$\nabla F(\mathbf{t}') = \begin{bmatrix} \frac{\partial A_1}{\partial p_1} & \cdots & \frac{\partial A_1}{\partial p_n} & \frac{\partial A_1}{\partial x_1} & \cdots & \frac{\partial A_1}{\partial x_m} & \frac{\partial A_1}{\partial y_1} & \cdots & \frac{\partial A_1}{\partial y_m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial A_n}{\partial p_1} & \cdots & \frac{\partial A_n}{\partial p_n} & \frac{\partial A_n}{\partial x_1} & \cdots & \frac{\partial A_n}{\partial x_m} & \frac{\partial A_n}{\partial y_1} & \cdots & \frac{\partial A_n}{\partial y_m} \\ \frac{\partial \theta_{1\cdot 1}}{\partial p_1} & \cdots & \frac{\partial \theta_{1\cdot 1}}{\partial p_n} & \frac{\partial \theta_{1\cdot 1}}{\partial x_1} & \cdots & \frac{\partial \theta_{1\cdot 1}}{\partial x_m} & \frac{\partial \theta_{1\cdot 1}}{\partial y_1} & \cdots & \frac{\partial \theta_{1\cdot 1}}{\partial y_m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \theta_{n\cdot 1}}{\partial p_1} & \cdots & \frac{\partial \theta_{n\cdot 1}}{\partial p_n} & \frac{\partial \theta_{n\cdot 1}}{\partial x_1} & \cdots & \frac{\partial \theta_{n\cdot 1}}{\partial x_m} & \frac{\partial \theta_{n\cdot 1}}{\partial y_1} & \cdots & \frac{\partial \theta_{n\cdot 1}}{\partial y_m} \\ \frac{\partial \theta_{1\cdot 2}}{\partial p_1} & \cdots & \frac{\partial \theta_{1\cdot 2}}{\partial p_n} & \frac{\partial \theta_{1\cdot 2}}{\partial x_1} & \cdots & \frac{\partial \theta_{1\cdot 2}}{\partial x_m} & \frac{\partial \theta_{1\cdot 2}}{\partial y_1} & \cdots & \frac{\partial \theta_{1\cdot 2}}{\partial y_m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \theta_{n\cdot m}}{\partial p_1} & \cdots & \frac{\partial \theta_{n\cdot m}}{\partial p_n} & \frac{\partial \theta_{n\cdot m}}{\partial x_1} & \cdots & \frac{\partial \theta_{n\cdot m}}{\partial x_m} & \frac{\partial \theta_{n\cdot m}}{\partial y_1} & \cdots & \frac{\partial \theta_{n\cdot m}}{\partial y_m} \end{bmatrix} ,$$

where p_i is the pressure of cell i and $\{x_j, y_j\}$ are the coordinates of junction j. ∇F is of size $(n + 2m) \times (n + n \cdot m)$.

We can now pose our problem as a Newton system $\nabla F(\mathbf{t}')\mathbf{z}' = -F(\mathbf{t}')$ with update step ${\mathbf{t}'}^{k+1} = {\mathbf{t}'}^k + {\mathbf{z}'}^k$. Here

$$\mathbf{z}' = \begin{bmatrix} \Delta p_1 & \cdots & \Delta p_n & \Delta x_1 & \cdots & \Delta x_m & \Delta y_1 & \cdots & \Delta y_m \end{bmatrix}^{\mathrm{T}}$$

is a vector of changes to apply to the pressure of cells and coordinates junctions. When the iterative method has brough us sufficiently close to the root we use that foam state as the output of the current step of the quasi-static simulation. We can measure how close to the root we are either as $\|\mathbf{z}'\|_\infty$, the largest change that must be applied to the foam, or $\|F\|_\infty$, the largest deviation from the optimal shape. The second method, $\|F\|_\infty$, can be problematic however, as the deviation is not, in general, garanteed to tend to zero. Only in an infinite, perfectly honey-comb shaped foam will we get $F = \mathbf{0}$.

Working with F and ∇F quickly becomes infeasible. Computing ∇F is on the order of $O(n^2 + m^2)$, a task that will take a very long time as n and m increases significantly. However, if we accept a slightly subobtimal solution and that we may have to compute more iterations, we can reduce the scope of the problem significantly. If we examine a foam sample, it becomes clear that the influence a given cell has on a given junction dwindle rapidly the further the cell is from the junction. Taking this to the extreme, by only gathering the

influence of the three cells touching the intersection (see Figure 3.1) we can replace F with one function per junction:

$$f_j^*(\mathbf{t}) = \begin{bmatrix} A_i^\tau - A_i \\ A_j^\tau - A_j \\ A_k^\tau - A_k \\ \frac{2\pi}{3} - \theta_i \\ \frac{2\pi}{3} - \theta_j \\ \frac{2\pi}{3} - \theta_k \end{bmatrix} = \mathbf{0} \tag{3.1.4}$$

where $\mathbf{t} = [p_i, p_j, p_k, x, y]^{\mathrm{T}}$, $p_{i,j,k}$ is the pressures of the three cells intersecting in the junction, and $\{x, y\}$ is the coordinates of the junction. We simply cherry-pick the three cells that contributes the largest change to the junction, which will inevitably be the three cells which share the junction. This is equivalent to considering ∇F a sparse block-diagonal matrix, with zero in all elements except on the diagonal. The per-junction Jacobian is then

$$\nabla f_j^*(\mathbf{t}) = \begin{bmatrix} \frac{\partial A_i}{\partial p_i} & \frac{\partial A_i}{\partial p_j} & \frac{\partial A_i}{\partial p_k} & \frac{\partial A_i}{\partial x} & \frac{\partial A_i}{\partial y} \\ \frac{\partial A_j}{\partial p_i} & \frac{\partial A_j}{\partial p_j} & \frac{\partial A_j}{\partial p_k} & \frac{\partial A_j}{\partial x} & \frac{\partial A_j}{\partial y} \\ \frac{\partial A_k}{\partial p_i} & \frac{\partial A_k}{\partial p_j} & \frac{\partial A_k}{\partial p_k} & \frac{\partial A_k}{\partial x} & \frac{\partial A_k}{\partial y} \\ \frac{\partial \theta_i}{\partial p_i} & \frac{\partial \theta_i}{\partial p_j} & \frac{\partial \theta_i}{\partial p_k} & \frac{\partial \theta_i}{\partial x} & \frac{\partial \theta_i}{\partial y} \\ \frac{\partial \theta_j}{\partial p_i} & \frac{\partial \theta_j}{\partial p_j} & \frac{\partial \theta_j}{\partial p_k} & \frac{\partial \theta_j}{\partial x} & \frac{\partial \theta_j}{\partial y} \\ \frac{\partial \theta_k}{\partial p_i} & \frac{\partial \theta_k}{\partial p_j} & \frac{\partial \theta_k}{\partial p_k} & \frac{\partial \theta_k}{\partial x} & \frac{\partial \theta_k}{\partial y} \end{bmatrix}. \tag{3.1.5}$$

Computing one ∇f_j^* per junction is now reduced to a task on the order of $O(m)$. However, we must be clear of the cost of replacing F with f_j^*: We can no longer hope to find the globally stable foam shape, but we can find a locally stable shape.

There is an observations that can be made about ∇f_j^* which we can put to use later: ∇f_j^* is dense and have no zero entries. We can realize this by simplely observing that any change in cell pressure or junction position will result in a positive or negative change in area and angle; they cannot remain unchanged (the world cell, which I will cover later, is an exception to this which must be handled as a special case.)

3.1.1 Finite difference

We do not have an analytical expression of ∇f_j^*, so instead we use a finite difference method [8] to approximate the partial differentials. For example, we can approximate

$$\frac{\partial A_i}{\partial p_j} \approx \frac{A_i^+ - A_i^-}{2h}$$

as a central difference, where A_i^+ and A_i^- is the area of cell i calculated with the pressure of cell p_j increased and decreased respectively by $h \in \mathbb{R}^+$. Similarly, in

$$\frac{\partial A_i}{\partial x} \approx \frac{A_i^+ - A_i^-}{2h},$$

A_i^+ and A_i^- is the area of cell i calculated with the x coordinate of the junction we are equilibrating increased and decreased by h.

3.1.2 Solving the linear system

We can now solve the linear system

$$\nabla f_j^* \mathbf{z} = -f_j^* \ , \tag{3.1.6}$$

where $\mathbf{z} = [\Delta p_i, \Delta p_j, \Delta p_k, \Delta x, \Delta y]^{\mathrm{T}}$, to get the changes in cell pressure and junction position to apply to equilibrate this particular junction.

The system in (3.1.6) is over-constrained with 6 equations and 5 unknowns. By observing the system we realise that this is because the three angles are dependent: Given two angles we can calculate the third. Formally

$$\theta_a = 2\pi - (\theta_b + \theta_c) \qquad \text{for } a, b, c \in \{i, j, k\} \text{ where } a \neq b \neq c \ .$$

We can therefore omit one of the equations, to get a system of five equations and five unknowns:

$$f_j = \begin{bmatrix} A_i^\tau - A_i \\ A_j^\tau - A_j \\ A_k^\tau - A_k \\ \frac{2\pi}{3} - \theta_i \\ \frac{2\pi}{3} - \theta_j \end{bmatrix} = \mathbf{0} \ ,$$

and

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

which allows us to solve

$$\nabla f_j \mathbf{z} = -f_j \ . \tag{3.1.7}$$

There exists many methods for solving systems of the form $\mathbf{Ax} = \mathbf{b}$, however, given that the system is limited in size ($\mathbf{A}$ is of size 5×5,) I have found that a simple Gauss-Jordan reduction [17] is a sufficient, and indeed a very fast, way to find a solution. The solution is simplified further by our earlier realization that the Jacobian ∇f_j is dense and non-zero, allowing us forego rearranging the matrix during reduction. Appendix B.1 lists an implementation example.

There is a slight chance that ∇f_j may be singular. However, this requires a completely symmetric junction which is unlikely to occure naturally. I have therefore chosen to disregard this possibility in favor of performance.

3.1.3 Updating the foam

After solving the system of equations for a given junction, the result is a set of changes to apply to the neighborhood of that junction in our Newton update step:

$$\mathbf{t}^{k+1} = \mathbf{t}^k + \mathbf{z}^k \ .$$

The junctions coordinate $\mathbf{x} = \{x, y\}$ must be changed to

$$\mathbf{x} \leftarrow \{x + \Delta x, y + \Delta y\}$$

and the pressure in each of the three surrounding cells must be updated:

$$\begin{aligned} p_i &\leftarrow p_i + \Delta p_i \\ p_j &\leftarrow p_j + \Delta p_j \\ p_k &\leftarrow p_k + \Delta p_k \end{aligned}$$

However, here we must realise a problem with updating junctions separately and in isolation: With no information about the contribution from other junctions in a cell, all junctions may result in an increase or decrease in pressure of the cell leading to a sudden large increase/decrease in cell pressure. This is undesirable both physically, visually, and computationally. The solution is to normalize the Δp contribution by the number of films in a cell. In other words, the pressure update becomes

$$\begin{aligned} p_i &\leftarrow p_i + \frac{\Delta p_i}{n_i} \\ p_j &\leftarrow p_j + \frac{\Delta p_j}{n_j} \\ p_k &\leftarrow p_k + \frac{\Delta p_k}{n_k} \end{aligned}$$

where $n_{i,j,k}$ is the number of films in cell i, j, and k respectively. This was suggested by Weaire and Kermode in [26], but was not used by Kelager in [13].

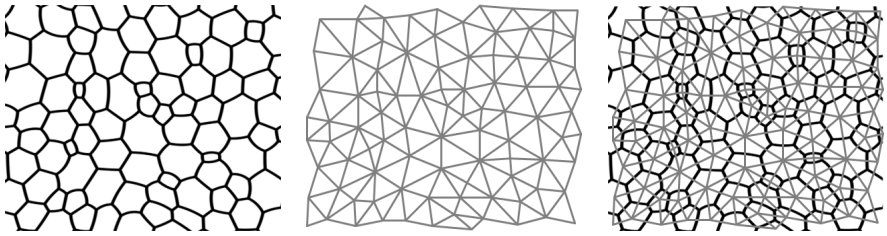
For a physical explanation of this normalization step, consider a cell with n films. The total amount of gas transported into or out of this cell is proportional to the number of films. Therefore the amount of gas transported over any single film is an n-th part of the total amount of transported gas.

3.2 Computational mesh

Before we can simulate a foam, we must define a computational mesh to work on. The obvious choice is to use a polygonal mesh, with cells as polygons, films as edges, and junctions as vertices. This has indeed also been the mesh used in former implementations such as Weaire and Kermodes [26] and Kelagers [13]. It is easy to conceptualize and there is an almost direct mapping between computational mesh and visual rendering. However, as shown by this author in [24], using the *dual* of the foam as the computational mesh has some clear advantages. In the following I will define the dual foam mesh and demonstrate how the foam simulation can benefit from it.

3.2.1 Discretizing a foam

To discretize a foam to create the dual foam mesh is equivalent to finding the dual of the foam when considered as a polygonal mesh. In the following I will refer to the polygonal (curved-edge) foam mesh as the *primal*.



(a) (b) (c)

Figure 3.2: A section of a foam sample showing the (a) primal mesh, (b) dual mesh, and (c) the dual superimposed on the primal. While the primal mesh consists of irregular n-gons, the dual is guaranteed to be a triangle mesh.

To establish a common vocabulary I will describe the dual as a *simplical complex*. For an introduction to simplical complexes see for example [6, 7]. I will adopt the notation that given an n-dimensional simplical complex, σ^k, where $0 \leq k < n$, are the simplices of the simplical complex and $p_k : \mathbb{Z} \to \mathbb{R}^N$ is a map from a k-simplex to a tuple of scalars. The set of 2-simplices are $\mathbf{F}$ (triangles), the set of 1-simplices are $\mathbf{E}$ (edges), and the set of 0-simplices are $\mathbf{V}$ (vertices). I will not need simplical complex of higher dimension than three.

The dual of a mesh can be found by placing a σ^0 in each face of the primal and then connect adjacent σ^0's with σ^1's as illustrated in Figure 3.2 to form σ^2's [24]. Note how each edge in the primal is mirrored by a σ^1 in the dual. Indeed it holds that $|\mathbf{F}_{\text{primal}}| = |\mathbf{V}_{\text{dual}}|$, $|\mathbf{E}_{\text{primal}}| = |\mathbf{E}_{\text{dual}}|$, $|\mathbf{V}_{\text{primal}}| = |\mathbf{F}_{\text{dual}}|$. (Here I have abused notation slightly. The primal is not a simplical complex as faces have more than three indices.) From this it follows that no topological information is lost.

From the above, it should be clear that in the dual mesh, cells are vertices and junctions are triangles, where each vertex in the triangle is one of the three cells surrounding the junction. From this we can derive the first advantage of the dual mesh: The Plateau rule that a junction must be formed in the intersection of exactely three cells are implicitely true. This may appear trivial, but the consequence is that we are incapable of constructing a foam that violates the Plateau rule, which again removes the burden of having to explicitely check that we do not accidentially create an invalid foam.

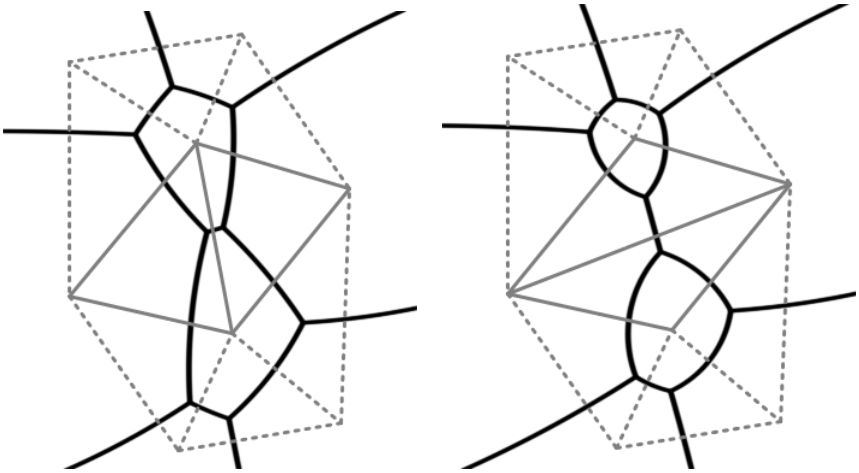
For the dual foam mesh to work we must abandon our usual preconception that positions should be stored at vertices and face properties in faces. In the dual mesh, junction coordinates are stored in σ^2 and cell pressure in σ^0. Formally, we maintain two maps

$$p_0 : [0, n) \to \{p\} \in \mathbb{R}$$

and

$$p_2 : [0, m) \to \{x, y\} \in \mathbb{R}^2$$

where n, m is the total number of σ^0 and σ^2 respectively and p is a cell pressure. In contrast to common computer graphics usage of meshes, we can say that the dual foam mesh ceases to have a geometric interpretation and becomes a purely topological construction; it can be helpful to consider the dual foam mesh as a



(a) (b)

Figure 3.3: A film is flipped in a T1 operation and the resulting junctions are equilibrated. The dual mesh is shown as gray lines with the two σ^2 participating the the operation highlighted.

graph with junctions and cells as connected leafs. Should we ever need to, we can return to the primal simply by finding the dual of the dual foam mesh.

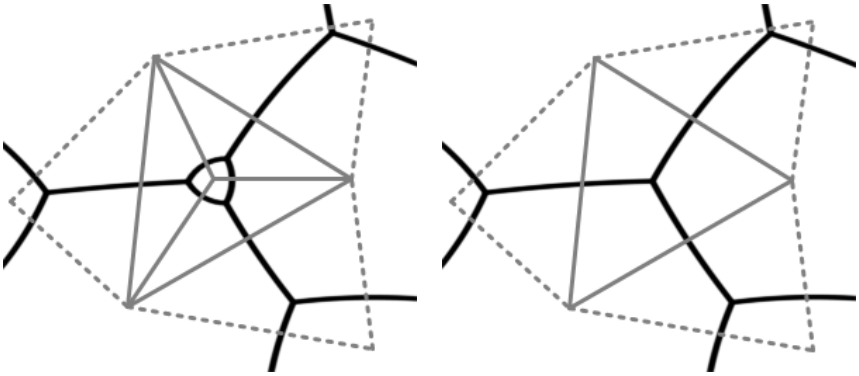
3.2.2 Topological changes

In the primal, topological changes are complicated operations as documented by Kelager [13]. However, in the dual, the operations are considerably simpler. Figure 3.3 demonstrates the T1 operation in the dual mesh. Note how the pre-T1 mesh topology in (a) is two σ^2 faces, joined on the shared film, with two free σ^0 faces, giving us four σ^0 faces, two of them connected by a σ^1. During the T1 process we simply change which two σ^0 are connected, conceptually flipping the σ^1 90°, after which we get the topology illustrated in Figure 3.3 (b).

The T2 process, as shown in Figure 3.4, involves three σ^2 faces, giving us four σ^0 faces, one of which is discarded in the process. A single σ^2 is then constructed from the remaining three σ^0.

The two cases given are unique and the only topological configuration the dual computational mesh can have for the given operation to be possible. Likewise the post-operation topological configurations are also unique. Note how in both operations the boundary σ^0 are fixed, only the connectivity is changed: The operations are deterministic, making them trivial to implement.

The last necessary piece of information we need to perform the topological operations are how to calculate new coordinates for the junctions involved. Recall from Section 2.3 that the operations are near-instantaneous and that we can not make any assumptions about the state or dynamics of the foam during the operation. If we turn this argument around we can freely choose what happens in the foam during the operation and are therefore free to create a scheme for calculating junction coordinates. The only consideration is that after the operation the foam should be in a form that can be equilibrated. I have chosen



(a) (b)

Figure 3.4: A cell is collapsed in a T2 operation and the resulting junction is equilibrated. The dual mesh is shown as gray lines with the three σ^2 participating the the operation highlighted.

a purely geometric scheme where the new coordinates are calculated based on the old coordinates. The T2 process simply calculates the average of the three old coordinates to produce the new coordinate:

$$\mathbf{x}_{\text{new}} = \frac{1}{3}\left(\mathbf{x}_i + \mathbf{x}_j + \mathbf{x}_k\right) \ .$$

The T1 process flips a film 90°. If the film connects junctions $\mathbf{x}_i$ and $\mathbf{x}_j$ then a vector

$$\mathbf{e}_{\text{perp}} = \widehat{\mathbf{x}_j - \mathbf{x}_i} \ ,$$

where $\widehat{\mathbf{e}} = \widehat{\{x, y\}} = \{y, -x\}$, is perpendicular to the film. The two new junction coordinates can then be calculated as

$$\mathbf{x}_a = \frac{\mathbf{x}_i + \mathbf{x}_j}{2} + \frac{\mathbf{e}_{\text{perp}}}{2} \ ,$$
$$\mathbf{x}_b = \frac{\mathbf{x}_i + \mathbf{x}_j}{2} - \frac{\mathbf{e}_{\text{perp}}}{2} \ .$$

Note that the distance from $\mathbf{x}_a$ to $\mathbf{x}_b$ is the same as from $\mathbf{x}_i$ to $\mathbf{x}_j$. The described scheme closely matches the one used by Kelager in [13].

While we can perform T1 operations on any film and T2 operations on any three-sided cell (with some limitations, see Section 3.6,) in the quasi-static simulation we need a fixed condition for when each operation should be performed. In a real foam, T1 operations happen when two junctions have joined, or in computational terms, when $\|\mathbf{x}_j - \mathbf{x}_i\| = 0$. However, in computer physics, such a hard limit is not usable, the finite precision of the floating point representation makes a precise limit impossible. We therefore introduce a T1 threshold value such that we perform a T1 operation when $\|\mathbf{x}_j - \mathbf{x}_i\| \leq \delta_{\text{T1}}$, where $\delta_{\text{T1}} \in \mathbb{R}^+$ is close to zero [15]. Likewise, T2 operations happens when all the gas has diffused out of a cell. In computational terms, $A_{\text{cell}} = 0$. Again, this hard limit is unusable, so we introduce a T2 threshold value so that we perform a T2 operation

when $A_{\text{cell}} \leq \delta_{\text{T2}}$, where $\delta_{\text{T2}} \in \mathbb{R}^+$ is close to zero. The precise value of δ_{T1} and δ_{T2} depends on the scale of the foam we are simulating.

In [13] Kelager treated T1 and T2 operations as two discrete occurances and, as Kelagers work was the basis for this thesis, so have I. However, in many simulations T1 and T2 operations are linked, such that T2 operations are performed when a T1 operation is performed on a film in a three-sided cell [30, 12]. This simplifies the algorithm but, as we shall see in Section 3.6, in some cases it limits the simulation.

In the examples given in this thesis, δ_{T1} and δ_{T2} was chosen to match the desired behavior of cells disappearing when "tiny" compared to the full foam sample. However, if an analytical model is desired, Weygand *et al.* [30] suggested using the average cell area as a comparative measure:

$$\delta_{\text{T1}} = \alpha \left(\frac{2A_{\text{total}}}{\pi m} \right)^{\frac{1}{2}} ,$$

where A_{total} is the total area of the foam, m is the number of junctions, and $\alpha \in \mathbb{R}^+$ is a user-supplied constant[1]. Weygand *et al.* used the model where T2 operations are a consequence of T1 operations and in that case

$$\delta_{\text{T2}} = \delta_{\text{T1}}^2$$

since a T2 operation will happen when the length of one side of the three-sided cell is less than δ_{T1} and two sides less than $2\delta_{\text{T1}}$.

3.3 Angles and areas

To equilibrate a junction, we will need to calculate the angles seperating the three films intersecting and the areas of the three cells adjacent to the junction. In the 2D plane, these values are relatively easy to calculate using trigonometry.

For now, let us consider pairs of cells that share a film. The shared film is given by two junctions $\mathbf{x}_0$ and $\mathbf{x}_1$ and a signed radius of curvature r as computed from Equation (2.2.1). Let $\mathbf{x}_t$ be a point equidistant from $\mathbf{x}_0$ and $\mathbf{x}_1$ such that $\triangle \mathbf{x}_0\mathbf{x}_1\mathbf{x}_t$ is an isosceles triangle with two sides of length r and one side of length $d = \|\mathbf{x}_1 - \mathbf{x}_0\|$, as illustrated in Figure 3.5. The angle ω is given by

$$\sin\left(\frac{\omega}{2}\right) = \frac{d}{2|r|} \Rightarrow \omega = 2\sin^{-1}\left(\frac{d}{2|r|}\right) \qquad \text{for } \frac{d}{2|r|} \in [-1;1] \ . \tag{3.3.1}$$

The input range of $\sin^{-1}$ dictates that $0 \leq d \leq 2|r|$, which can not be garanteed in a random foam. Therefore we must ensure that d does not exceed $2|r|$ when performing this computation, for example by clamping the value. The angle $\beta = \frac{\pi - \omega}{2}$ is simply the remaining angle in the triangle. Finally, because vectors $\mathbf{x}_t - \mathbf{x}_0$ and $\mathbf{x}_t - \mathbf{x}_1$ must be normal to the circle with radius $|r|$ and center in $\mathbf{x}_t$, it follows that $\alpha + \beta = \frac{\pi}{2}$ and therefore $\alpha = \frac{\omega}{2}$. However, as r arose from a pressure difference, r may be negative. We must therefore preserve the sign of r when calculating α before we can use it for both cells in the pair we are

[1] Weygand *et al.* divided α by $n_{\text{virtual}} + 1$ which was the number of virtual vertices inserted between junctions. For our use $n_{\text{virtual}} = 0$.

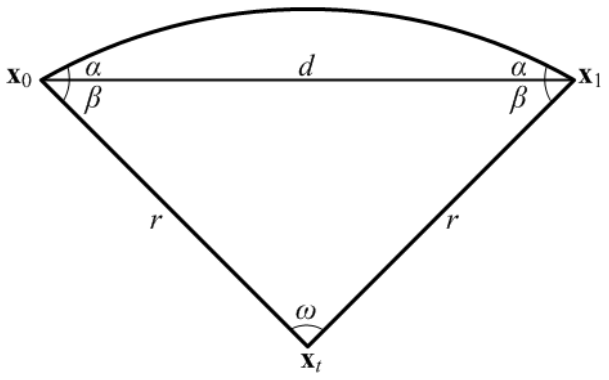


Figure 3.5: The necessary values for calculating the angles ω and α.

considering:

$$\alpha = \operatorname{sgn}(r)\frac{\omega}{2} = \operatorname{sgn}(r)\sin^{-1}\left(\frac{d}{2|r|}\right) \qquad \text{for } \frac{d}{2|r|} \in [-1;1] \tag{3.3.2}$$

where

$$\operatorname{sgn}(x) = \begin{cases} 1 & \text{for } x \geq 0 \\ -1 & \text{for } x < 0 \end{cases} .$$

Now, given α we can calculate the angle between two films intersecting at a junction. Let $\mathbf{x}_i$ be a junction and $\mathbf{x}_j$ and $\mathbf{x}_k$ be two adjacent junctions as illustrated in Figure 3.6. Given the straight line edges $\mathbf{e}_j = \mathbf{x}_j - \mathbf{x}_i$ and $\mathbf{e}_k = \mathbf{x}_k - \mathbf{x}_i$, with arc radius r_j and r_k respectively, we can calculate the angle

$$\psi_i = \cos^{-1}\left(\frac{\mathbf{e}_j \cdot \mathbf{e}_k}{\|\mathbf{e}_j\|\|\mathbf{e}_k\|}\right) \tag{3.3.3}$$

and from this the angle separating the two edges

$$\phi_i = \begin{cases} \psi_i & \text{for } \det(\mathbf{e}_j, \mathbf{e}_k) \geq 0 \\ 2\pi - \psi_i & \text{for } \det(\mathbf{e}_j, \mathbf{e}_k) < 0 \end{cases} , \tag{3.3.4}$$

where $\det(\mathbf{a}, \mathbf{b}) = \mathbf{a}^x\mathbf{b}^y - \mathbf{a}^y\mathbf{b}^x$. (3.3.4) covers the case where the angle separating the two films are greater than 180°, which can happen during the equilibration process when a cell becomes concave, in which case (3.3.3) is not sufficient [13].

By (3.3.2) we get the arc angles

$$\alpha_j = \operatorname{sgn}(r_j)\sin^{-1}\left(\frac{\|\mathbf{e}_j\|}{2|r_j|}\right) \tag{3.3.5}$$

and

$$\alpha_k = \operatorname{sgn}(r_k)\sin^{-1}\left(\frac{\|\mathbf{e}_k\|}{2|r_k|}\right) . \tag{3.3.6}$$

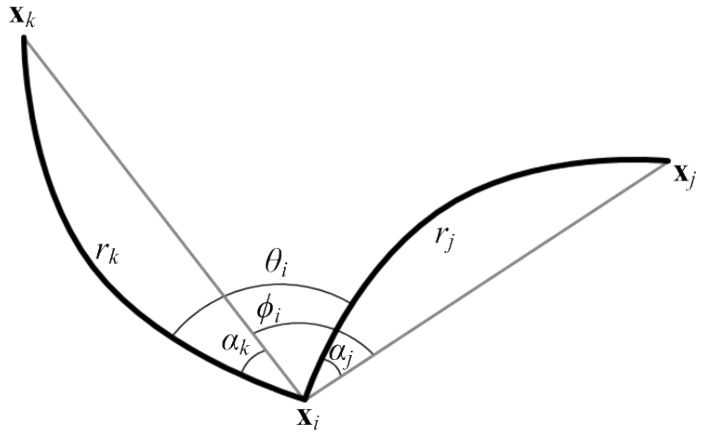


Figure 3.6: The necessary values for calculating the angle θ_i.

Combining (3.3.4), (3.3.5), and (3.3.6) we can calculate the full angle

$$\theta_i = \phi_i - \alpha_j + \alpha_k \ . \tag{3.3.7}$$

Note the difference in sign: In Figure 3.6 $\operatorname{sgn}(r_j) = \operatorname{sgn}(r_k)$, but the arc caused by r_j decreases θ_i while the arc caused by r_k increases θ_i. Therefore we must subtract α_j and add α_k and, since α is signed, we get correct addition of the angle.

Now that we can calculate the incident angles at a junction, let us turn to calculating the area of a cell. We do this by first calculating the area of the cell as a straight line polygon, then adding the (signed) area of the arc segments on the edges. The area of a single triangle $\triangle \mathbf{x}_i\mathbf{x}_j\mathbf{x}_k$ is

$$\begin{aligned} A_\triangle &= \frac{1}{2}\det(\mathbf{x}_j - \mathbf{x}_i, \mathbf{x}_k - \mathbf{x}_i) \\ &= \frac{1}{2}\left((\mathbf{x}_j^x - \mathbf{x}_i^x)(\mathbf{x}_k^y - \mathbf{x}_i^y) - (\mathbf{x}_j^y - \mathbf{x}_i^y)(\mathbf{x}_k^x - \mathbf{x}_i^x)\right) \ . \end{aligned} \tag{3.3.8}$$

The area of a cell with V junctions, for convenience and without loss of generality indexed as $\mathbf{x}_{0\cdots V-1}$, is then

$$A_{\text{poly}} = \frac{1}{2}\sum_{k=1}^{V-2}\det(\mathbf{x}_k - \mathbf{x}_0, \mathbf{x}_{k+1} - \mathbf{x}_0) \ . \tag{3.3.9}$$

Please note that this triangular tesseleation of the cell is in no way related to the computational mesh. Rather, A_{poly} is a tesselation of the primal mesh.

To calculate the arc areas, consider again the isosceles triangle $\triangle \mathbf{x}_0\mathbf{x}_1\mathbf{x}_t$ defined earlier and the circle with radius $|r|$ centered in $\mathbf{x}_t$. The arc area $A_\frown$ is the area of the sector of the circle defined by angle ω minus the area of $\triangle \mathbf{x}_0\mathbf{x}_1\mathbf{x}_t$:

$$\begin{aligned} A_\frown &= \operatorname{sgn}(r)\frac{1}{2}r^2\omega - \operatorname{sgn}(r)\frac{1}{2}r^2\sin\omega \\ &= \operatorname{sgn}(r)\frac{1}{2}r^2\left(\omega - \sin\omega\right) \ . \end{aligned} \tag{3.3.10}$$

Again, to preserve the symmetry between cell pairs, the arc area must be signed. The total arc area of a cell with E films, for convenience and without loss of generality indexed as $r_{0\cdots E-1}$, is then

$$A_{\text{arc}} = \sum_{k=0}^{E-1} \text{sgn}(r) \frac{1}{2} r_k^2 \left(\omega_k - \sin \omega_k\right) \tag{3.3.11}$$

Finally, the total cell area is

$$A = A_{\text{poly}} + A_{\text{arc}} \ . \tag{3.3.12}$$

3.4 Boundary conditions

Until this point we have worked under the implicit assumption that the foam sample we are simulating is infinite in all directions. However, to simulate a foam on a finite computer we must impose some constraints on the boundary. The simplest boundary condition is to fix the boundary junctions, which is akin to sealing the foam in a hermetically sealed container. The fixed boundary condition is generally undesirable as the fixed junctions doesn't allow the foam to evolve freely and thus can introduce statistical errors and instabilities. It most be noted that fixed boundaries are common in real world observations of foam. For example, Stavans and Glazier [22] reports sealing a foam in a plexiglass box, but that they disregarded cells touching the walls when conducting measurements.

Under periodic boundaries the foam is allowed to evolve freely, but it wraps around on all axis, creating an infinitely replicating foam that fills all available space. Periodic boundaries have been used in many foam simulations as it statistically emulates an infinite foam sample [25, 26]. If the goal of the simulation is to collect statistics for different foams then periodic boundaries are the best choice.

In [13] Kelager introduced a free surface boundary condition, a new concept in foam simulation. By introducing a special cell, which he named the world (or ghost) cell, a free surface can be constructed, creating a finite, freely evolving foam sample. The world cell emulates the effects of the environment surrounding the foam by having infinite area and unit pressure. All boundary junctions have the world cell as one of the three cells surrounding it and in the simulation the world cell is treated as any other cell except that it never changes pressure and it is not rendered in the visualization. The free surface boundary condition is the best choice if the goal of the simulation is visualization. In this thesis I have used the free surface exclusively and in Chapter 6 I will examine the effects of the free surface on the statistics of foam samples.

To use the free surface boundary condition we must accommodate for the world cell when equilibrating a junction using (3.1.7). If one of the three cells surrounding the junction is the world cell, we replace the corresponding row and column in ∇f_j and f_j with zero. For example, if cell j is the world cell then the

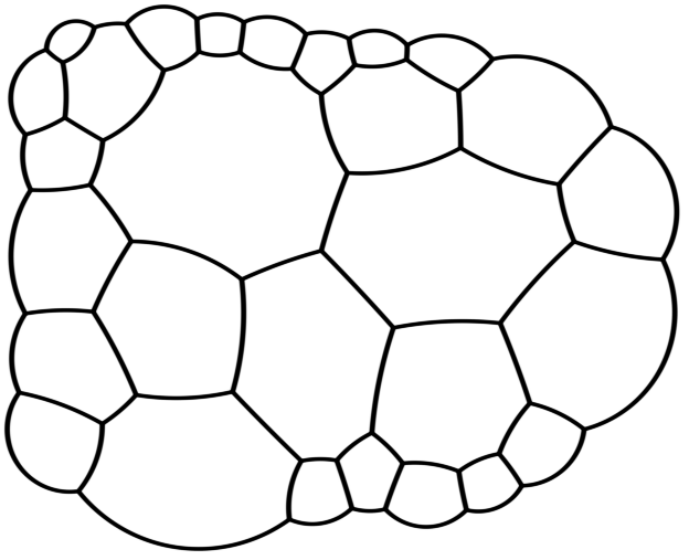


Figure 3.7: A small foam sample with the free surface boundary condition. Note how all boundary cells have a higher pressure than the surrounding environment, causing all boundary films to curve out of the foam sample. This closely matches the observed physical behavior of real foam.

system becomes

$$\begin{bmatrix} \frac{\partial A_i}{\partial p_i} & 0 & \frac{\partial A_i}{\partial p_k} & \frac{\partial A_i}{\partial x} & \frac{\partial A_i}{\partial y} \\ 0 & 0 & 0 & 0 & 0 \\ \frac{\partial A_k}{\partial p_i} & 0 & \frac{\partial A_k}{\partial p_k} & \frac{\partial A_k}{\partial x} & \frac{\partial A_k}{\partial y} \\ \frac{\partial \theta_i}{\partial p_i} & 0 & \frac{\partial \theta_i}{\partial p_k} & \frac{\partial \theta_i}{\partial x} & \frac{\partial \theta_i}{\partial y} \\ \frac{\partial \theta_k}{\partial p_i} & 0 & \frac{\partial \theta_k}{\partial p_k} & \frac{\partial \theta_k}{\partial x} & \frac{\partial \theta_k}{\partial y} \end{bmatrix} \mathbf{z} = \begin{bmatrix} A_i^\tau - A_i \\ 0 \\ A_k^\tau - A_k \\ \frac{2\pi}{3} - \theta_i \\ \frac{2\pi}{3} - \theta_k \end{bmatrix}, \tag{3.4.1}$$

effectively reducing the system to 4 equations and 4 unknowns. Note how the last two rows are changed from (3.1.7) so that the row with values from cell j is the one that is left out. The result of using (3.4.1) is that junctions on the the free surface boundary is free to move while boundary cells will tend to have higher pressure than the world cell, causing boundary films to be convex. Figure 3.7 shows a typical foam sample with a free surface.

3.5 Hard constraints

In the equilibration process we have already covered two constraints, described in Equations (3.1.2) and (3.1.3). I shall refer these as *soft constraints*: The relaxation process will find the solution that best satisfy them, but they can get worse if no better solution can be found. As the number of iterations increase we will converge towards the best possible solution. In the following I will introduce two *hard constraints*. These are invariants in the simulation that can not be violated.

Kelager used one hard constraint in [13], the *In Bubble* constraint, which was an intersection test between junctions and a straight edge triangulation of

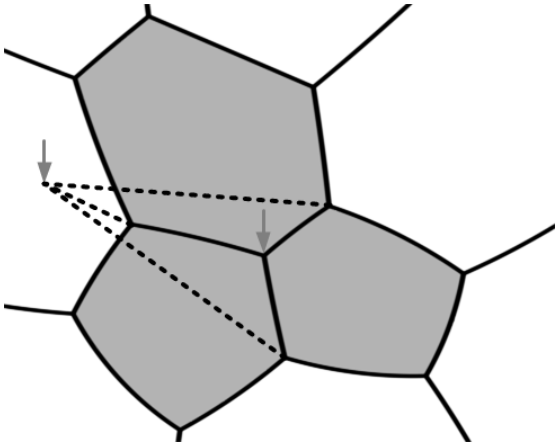


Figure 3.8: The three-cell neighborhood of a junction. Moving the junction outside of the envelope of the three cells will cause the local energy of the junction to increase as the films become longer.

the three surrounding cells. Junction positions must be subject to some form of constraint on the envolope, shown in Figure 3.8, in which the junction is free to move. This was observed by Kelager, who used the In Bubble intersection test to ensure that junctions did not move outside the area enclosed by the three surrounding cells. There is a number of problems with this approach: 1) It is not physically founded, but a stop-gap measure to counter an unintentional behavior. 2) To simplify implementation, only a straight edge polygonal tesselation of cells where used in the intersection test, which makes the constraint unsuitable for boundary cells which will typically have a film with a small radius seperating the foam from the environment (see for example Figure 3.7.) 3) Two-sided cells (described below) can not be handled at all using the intersection test, as they can not be tesselated. 4) Even with the intersection test, junctions could still make large sudden jumps if the neighboring cells where large.

3.5.1 Orientation Invariant Constraint

The first constraint I will introduce ensures that the intrinsic geometric orientation of junctions are never inverted. Figure 3.9 shows two foam configurations that results in a change in the intrinsic geometric orientation of a junction. In (a)-(b) a junction moves from a valid to an invalid state. Note how the three films in the junction changes orientation. In (c)-(d) a junction moves to bring an incident junction into an invalid state. To create a definition of these invalid states in our model, consider the six permutations of films in Figure 3.10. We define that, when visiting the three films of a junction, we shall always visit them in the order $0 \rightarrow 1 \rightarrow 2$ and, geometrically, we shall always visit them in a counter-clockwise order. This is equivalent to the intrinsic orientation of a triangle in a triangulated mesh and this indeed matches the order of films in a junction in the dual computational mesh. Formally, given three films f_0, f_1, and f_2 intersecting in a junction, the angle θ_0 between films f_0 and f_1, and the angle θ_1 between films f_0 and f_2, with the angles calculated with Equation (3.3.7), it must always be true that $\theta_0 < \theta_1$. This is the *orientation invariant*

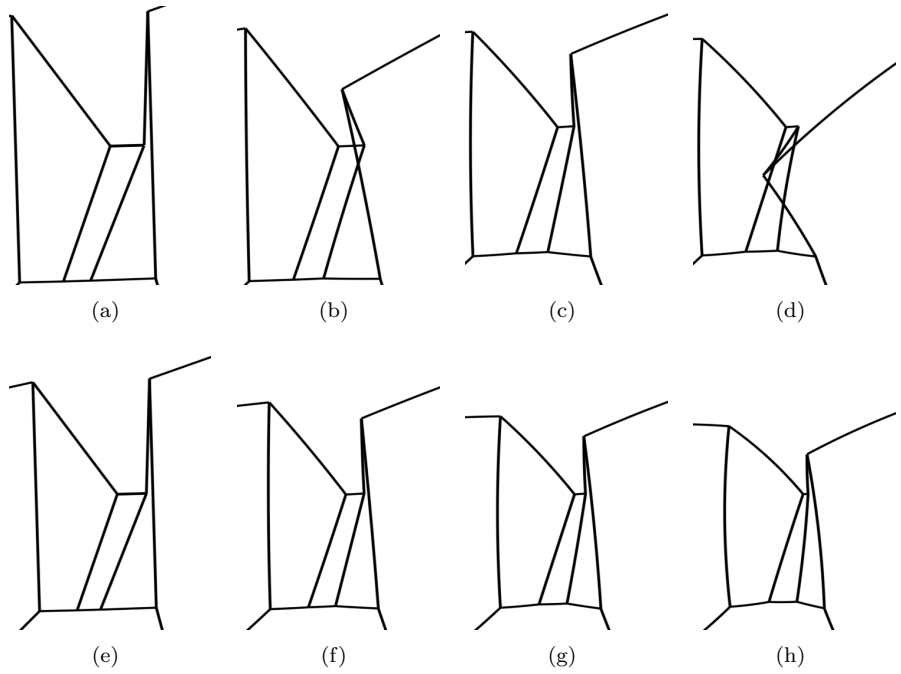


Figure 3.9: Without constraint on the relaxation process invalid geometry might arise. The images are taken from inside a single time step, from different iterations of the quasi-static equilibration process. (a)-(b) Locally inverted angles. (c)-(d) Secondary inverted angles. (e)-(h) With the Orientation Invariant Constraint enabled no inverted angles are formed.

of the junction. If the orientation invariant is violated, we say that the junction has become inverted. From this we can formulate a constraint on the movement of junctions:

Axiom (Orientation Invariant Constraint)**.** *A process which changes the coordinates of a junction must maintain the orientation invariant for that junction and the three incident junctions.*

Figure 3.9 (e)-(h) depictes the same foam but with the Orientation Invariant Constraint enabled. Now the junction movements are constrainted and no junctions become inverted. It is, of course, vital that the foam is initialized to be in a form where the orientation invarient is true for all junctions. The contraint can not be used to cause inverted junctions to become valid.

To realize why junctions can become inverted and why the Orientation Invariant Constraint is necessary, consider again Figure 3.10. We can calculate the turning angle between each pair of incident films with (3.3.7), but in doing so we must calculate the straigt edge angle between the films with (3.3.4) and it is this angle that is problematic. As long as the orientation invariant is true, ϕ will be the shortest positive angle between the two film, but if the junction becomes inverted (3.3.4) will give the positive long angle, not the negative shortest angle that would be necessary for the relaxation process to "untangle" the inverted junction. In other words, (3.3.7) is only sufficient as long as junctions can not become inverted, which is why the OIC is introduced.

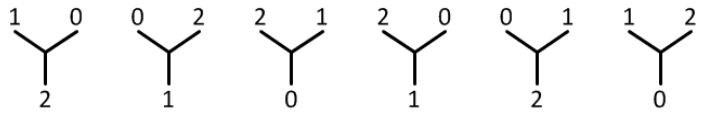


Figure 3.10: The six permutations of three films intersecting in a junction. Note how the first three and last three are rotations of the same ordering. Here we define the counter-clockwise orientation (the first three) to be the orientation we expect.

The Orientation Invariant Constraint fulfils the same function as Kelagers In Bubble constraint by keeping junctions from moving outside the three-cell envelope, but remains functional even with curved films. Like the In Bubble constraint, the Orientation Invariant Constraint is not physically founded, but a geometric solution to the problem.

3.5.2 Energy Decay Constraint

While the Orientation Invariant Constraint solves the problem of junctions becoming inverted and moving outside the envelope of the three surrounding cells, it does not prevent large jumps in junction position. In the following I will show a physical constraint that solves this problem by limiting the movement to always cause a decrease in potential surface energy. From Section 2.5 we know how the energy of a foam can be estimated and that the total energy of the foam can not increase over time.

The length of any given film in the foam can be calculated as

$$l = \omega r \ , \tag{3.5.1}$$

where ω is calculated using (3.3.1) and r is the arc radius of the film. From Equation (2.5.3) we can then compute the energy of a single film as

$$E_{\text{film}} = 2\gamma\omega r \tag{3.5.2}$$

and the energy of a cell as

$$E_{\text{cell}} = 2\gamma \sum_i \omega_i r_i \ , \tag{3.5.3}$$

where i runs over the films of the cell. Lastely, the total energy of a foam is

$$E_{\text{total}} = 2\gamma \sum_{i \in E} \omega_i r_i \ , \tag{3.5.4}$$

where E is the set of all films in the foam. Now, given an energy measure, we can formulate a new constraint on the equilibration process:

Axiom (Energy Decay Constraint)**.** *A process which changes the geometric or topological configuration of a foam must always result in a state of lower energy:*

$$E_{total}^{after} \leq E_{total}^{before} \ .$$

In a novel approach[2], we can use this constraint to improve the stability of the foam simulation: Only changes which lowers or preserves the energy of the foam network are allowed, resulting in either no change or a net gain in equilibration after each iteration. Without such constraints, the local relaxation of a junction can result in a violent change to the junctions coordinates or the pressure of the surrounding cells. By limiting the changes to obey the Energy Decay Constraint we can remove these violent changes, which will result in an increased stability of the simulation.

The Energy Decay Constraint axiom deals with the total energy of the foam. However, as with the relaxation process, where we only hope to reach a local equilibrium, our goal is only a local decrease in energy which we hope will result in a global decrease in energy. When we can limit our attention to local changes we can consider each step in the equilibration process separately.

Coarsening

The diffusion process, as governed by Equation (2.4.5), does not directly affect the foam, but only the cell target area. Only during the equilibration process does the cell target area translate to a change in the foam. We need therefore not examine the energy during coarsening, as it will not change during this part of the simulation.

Topological changes

First, let us examine the topological changes, as described in Section 3.2.2. It is important to first realise that the topological operations are purely mechanical, forced movement of the involved junctions. After a T1 or T2 process, the junctions involved will not be in equilibrium.

I will claim that we can not measure the energy of the foam directly following a topological change and get a meaningful result. Rather we must perform the topological operation *and then relaxe the involved junctions into an equilibrated state* before it makes sense to measure the energy. To understand this, we must realise that the topological operations are not physically based, but rather mechanical approximations to observed behavior. What happens during the topological operation can be more or less arbitrarily defined by the implementation of the simulation, as long as the resulting foam does not violate the Plateau laws. It is only later, after equilibration, that the foam assumes a physically based shape.

The result of this discussion is that we can not use the Energy Decay Constraint to allow or discard topological changes. We must perform all changes and then later, during equilibration, examine whether the energy decreases or not. This is an inherent defficiency in the Vertex Model: Because we do not have a proper physical model for the behavior of foam during topological changes we must approximate it and, as a result, we can not apply physical reasoning to the immediate result of topological changes.

[2]To our knowledge, no one has explicitly used energy as a constraint in a foam simulation.

Equilibration process

It is in the equilibration process, where we attempt to find a stable foam shape, that the Energy Decay Constraint can be made to make a difference. After calculating the Jacobian ∇f_j and solving (3.1.7), we get a vector of changes that can be applied to neighborhood of a junction as described in Section 3.1.3. There are two discrete changes that must be performed: Moving junctions and changing cell pressure. I will examine each in turn.

From Figure 3.8 it is appearent that any large change in junction position will cause the local energy of the junction to increase as the length of the three films meeting in the junction increases. This follows directly from the triangular inequality: Given an arbitrary point, the combined distance from the three corners of a triangle will be least when the point is inside the triangle. From this follows that if we move a junction from inside the triangle defined by the three incident junctions to outside this triangle, the combined film length will increase. By only allowing junction movement that causes the energy to decrease, large sudden jumps are disallowed. At the same time we can remove the computationally costly intersection test.

The other aspect of the equilibration process is cell pressure updates. At first glance it seems reasonable that we can apply the Energy Decay Constraint to cell pressures in the same fashion as with junction position, that is, the local energy of a cell must decrease when the pressure is updated. However, this is unfortunately not possible: The diffusion process will cause some cells to shrink and others to expand. Globally and statistically the energy of the foam will decrease, but when only examining local changes to cells, we can not use the Energy Decay Constraint. We can simply not determine locally if the expanding cell is counteracted by contracting cells elsewhere in the foam. A possible solution to this would be to make the Energy Decay Constraint dependent on the valency (number of films) of the cell: If the valency is less than six then the energy must decrease, if the valency is greater than six then the energy must increase, else the energy must remain constant. I have however not tested this.

3.5.3 Imposing the constraints

The purpose of the two presented constraints is to limit the movement of junctions. As they represents invariants that must be true before and after the movement, they should be enforced on each junction as it is updated with the method from Section 3.1.3. If the constraints are violated the change should be rolled back. Later, when we gather everything into a single algorithm in Section 3.7, we will see in more details where in the process the constraints come in.

3.6 Handling two-sided cells

An aspect of the Vertex Model which has so far not been well described is how to handle the case where a topological change (T1 or T2) causes a three-sided cell to collapse into a two-sided cell. Two-sided cells are pathological and Weaire and Kermode noticed that in a real foam, two-sided cells have extremely short lives [26]. Two-sided cells can only form as floating on the film of another, larger

cell and random fluctuations in the foam will quickly cause the unbalanced two-sided cell to slide along the film it floats on until it comes to rest against another cell at which point the cell becomes three-sided again. Figure 3.11 illustrates the typical life cycle of a two-sided bubble.

Weaire and Kermode described two-sided cells in [26], but concluded that they will not form in a real foam and that if they do form, they are meta-stable, but short lived. They showed that the surface energy in the foam will increase if two-sided cells are formed and used this as proof that two-sided cells can not occur naturally. They backed this up by experimental observations of real foam samples that showed no two-sided cells. In the Vertex Model, two-sided cells will form as a result of the purely mechanical process of T1 and T2 operations, however, as I demonstrated in Section 3.5.2, we can not garantee that the local energy of a foam will decrease during T1 and T2 operations. We can therefore not prevent two-sided cells by observing energy changes.

Weygand *et al.* touched upon two-sided cells in their discussion of topological operations in [30]. They introduced a new topological operation, T3, which was the collapse of a two-sided cell into an unbroken film. They claimed that a T2 process could be decomposed into a T1 operation followed by a T3 operation. However, they made no mention of handling two-sided cells as part of their simulation, which must lead to the conclusion that they eliminated two-sided cells immidiately upon formation. In the following I will derive a method that makes T3 operations unnecessary.

The preceding discussion leads us to two options: We can either totally disallow two-sided cells from forming, or we can make the model robust enough to gracefully support two-sided cells. The first option is simple: Whenever we are about to perform a T1 or T2 operation we measure the valency (the number of films) of the cells which will loose a film (two cells for a T1, three cells for a T2.) If the valency is less than four we abort the topological operation. In this way, no two-sided films can ever form. However, this is not a tenable solution; preventing cells from collapsing in T2 operations can lead to very small cells, impeding the evolution of the foam and giving rise to numerical instabilities [26]. For example, in Figure 3.11, if the small cell in (a) is not allowed to collapse, then the cells will continue to shrink (as a result of diffusion) but never disappear, impeding the behaviour of the larger cells.

The second option is therefore the most desirable. There are three aspects of the model we must examine to determine if two-sided cells will cause problems: The von Neumann diffusion, the Jacobian ∇f_j calculation, and the computational mesh. The von Neumann diffusion is trivial to verify. Equation (2.4.5) will cause two-sided cells to quickly shrink, which is desirable so that they quickly collapse. There are two sides to the Jacobian calculation: Calculating film turning angles with (3.3.7) and cell areas with (3.3.12). When calculating the film turning angle, the angle $\phi_i = 0$ as the two straight line edges becomes parallel in (3.3.4), however the arc angles α_j and α_k are not affected. In other words, the angle calculation can handle two-sided cells unchanged. When calculating cell area, we can not construct a full triangle inside the two-sided cell so $A_{\text{poly}} = 0$, however, the arc angle A_{arc} is not affected. Therefore, the cell area can be calculated for two-sided cells. From these results we can conclude that the presented methods for calculating angles and areas are fully compatible with two-sided cells.

The last issue is whether the computational mesh will support two-sided

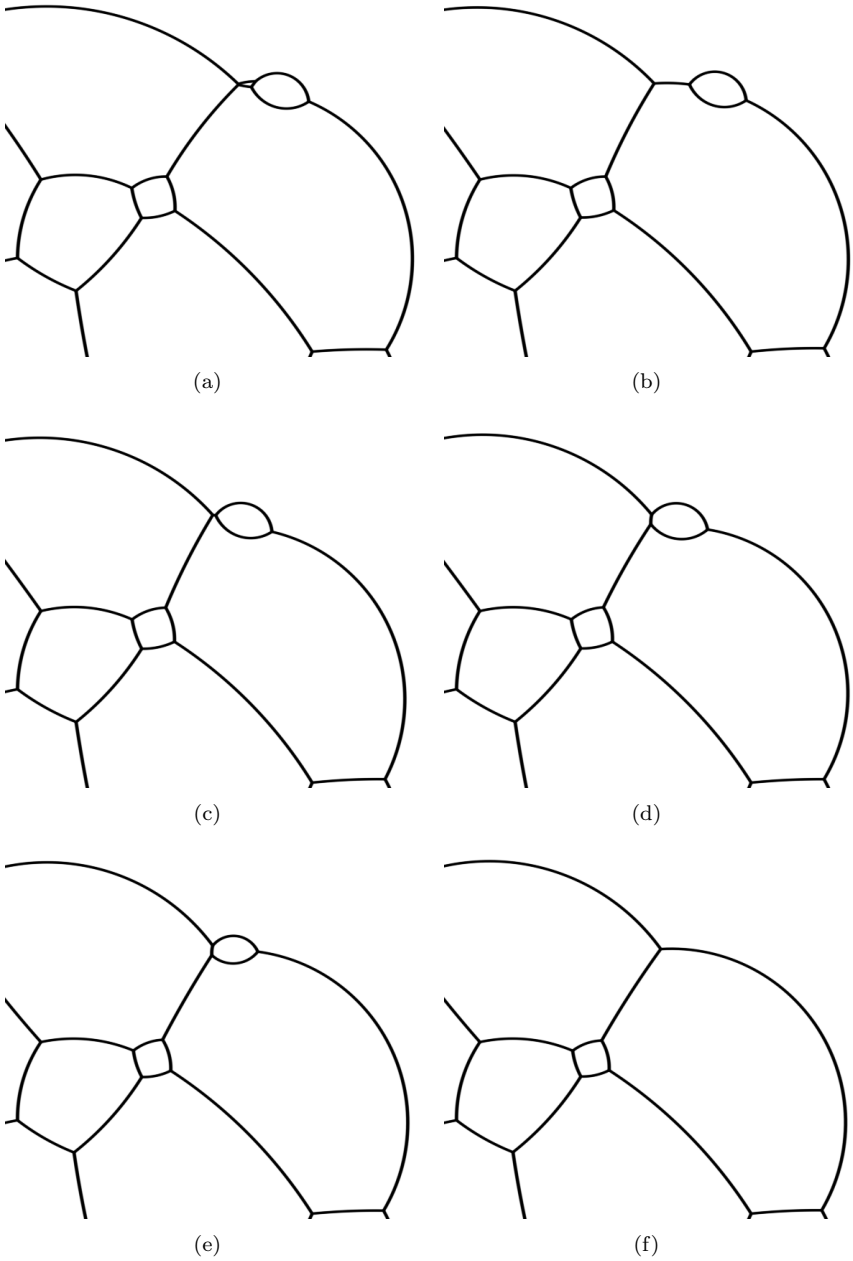


Figure 3.11: The typical life of a two-sided cell. (a) As the smaller of the two boundary cells collapses in a T2 process, (b) a two-sided cell is formed. (c) Random fluctuations in the foam will cause the cell to slide on the film of the larger cell, until it is near a corner. (d) A T1 process then occures, causing the cell to gain a film to become three-sided again. (e) Diffusion causes the three-sided cell to shrink and (f) finally, the now three-sided cell (eventually) collapses in a T2 process. This sequence covers 12 time steps of $\frac{1}{30}$ second each, equating to less than half a second of simulation. The two-sided cell lasted for two time steps before becoming three-sided again.

cells. At first glance the dual foam mesh can handle them implicitly (one-sided cells, on the other hand are not be supported,) however, there is a subtle issue that must be addressed. The problem was brought to light in [24] and has to do with the connectivity of the dual mesh graph: For two sided cells to appear there must be two discrete films separating the same two cells in the foam. In terms of the underlying simplical complex there must be two distinct σ^1 faces with the same pair of indices (in graph terms there must be two distinct edges connecting the same two leaf nodes.) This violates the discrete manifold constraints. However, if we relax the discrete manifold requirements and allow more than one film between the same cells, then the computational mesh fully supports two-sided cells.

3.6.1 Two-sided cell collapse

One further issue we must address is that, as mentioned, two-sided cells can not be allowed to collapse, as this would lead to an invalid foam. There are two aspects to this: Topological changes must not be allowed to reduce the number of films in a two-sided cell and we must ensure that two-sided cells eventually become three-sided and collapses. The first aspect can be handled in the same way as we can prevent two-sided cells, as described above, by disallowing T1 and T2 process on cells with a valency of less than three. The second aspect, ensuring that two-sided cells will collapse, is more challenging. To match the observed physical behavior of real foam as described in [26], I have chosen to move the two-sided cell along the shortest neighboring film until it is near enough a junction that a T1 operation will cause it to gain a film. It should be noted that this is not physically founded as such, but a practical solution to a problem that mimics the observed physical behaviour.

To find the shortest neighboring film, (3.5.1) can be used. Given the shortest film length l, we can calculate an angle

$$\alpha = \frac{l - \delta_{\mathrm{T1}}}{r} \ , \tag{3.6.1}$$

where r is the arc radius of the film and δ_{T1} is the distance threshold of a T1 operation. That is, if $l \leq \delta_{\mathrm{T1}}$ then the film is elegible for a T1 operation. The angle α represents how far we should rotate the two junctions in the two-sided cell to make the shortest neighboring film elegible for a T1 operation, as illustrated in Figure 3.12. Let

$$\mathbf{x}_m = \frac{1}{2}\left(\mathbf{x} + \mathbf{x}_0\right)^{\mathrm{T}}$$

be the midpoint of the straight edge between a junction $\mathbf{x}$ in the two-sided cell and the junction $\mathbf{x}_0$ we are moving toward. The vector

$$\mathbf{e} = \{\mathbf{x}^y - \mathbf{x}_0^y, -\mathbf{x}^x + \mathbf{x}_0^x\}$$

is perpendicular to a vector from $\mathbf{x}$ to $\mathbf{x}_0$ and of the same magnitude. Let

$$\mathbf{x}_c = \mathbf{x}_m + k\frac{\mathbf{e}}{\|\mathbf{e}\|} \ , \tag{3.6.2}$$

Figure 3.12: The necessary values for moving a cell along an film.

where

$$k = \mathrm{sgn}(r)\sqrt{r^2 - \left(\frac{\|\mathbf{e}\|}{2}\right)^2} \ ,$$

be the center of the circle with radius r where $\mathbf{x}$ and $\mathbf{x}_0$ lies on the border. We can now calculate the new position of the junction as

$$\mathbf{x} = \mathbf{x}_c + \mathbf{R}(\alpha)\left(\mathbf{x} - \mathbf{x}_c\right) \ , \tag{3.6.3}$$

where

$$\mathbf{R}(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

is the normal 2D rotation matrix. By applying (3.6.3) to both junctions in the two-sided cell we slide the cell along the film of the supporting cell to come to rest in distance δ_{T1} from the corner junction $\mathbf{x}_0$, but still preserve the distance between the two junctions in the two-sided cell.

At the next T1 opportunity, the two-sided cell will then undergo a T1 operation to become three-sided and then, eventually, collapse in a T2 operation, as illustrated in Figure 3.11.

3.7 The complete model

Now that we have covered all aspects of the Vertex Model of dry foam, we can gather and summarise into an algorithm. I will use the notion of a "virtual time," where simulation time is discretized in a sequence of discrete time steps of length Δt. A single time step proceeds as following:

1. For each cell, update target area based on von Neumann diffusion.
2. For each two-sided cell, move the cell to nearest corner.
3. Perform topological operations.
4. For each junction, calculate $\nabla f_j \mathbf{z} = -f_j$, solve for $\mathbf{z}$.
5. For each junction, calculate local energy E_{before}, update junction based on $\mathbf{z}$, calculate local energy E_{after}. If $E_{\mathrm{after}} > E_{\mathrm{before}}$ or if the junction has become inverted, discard changes.
6. For each cell, update pressure based on $\mathbf{z}$.

```
for each c in Cells do
  A_c^τ ← diffusion(c, Δt)

  if valency(c) = 2 then
    move(c) // See 3.6.1
  end
end

// Perform topological operations, see 3.2.2

for each j in Junctions do
  f_j ← calculate_constraints(j)
  ∇f_j ← calculate_jacobian(j)
  z ← solve(∇f_j, −f_j)

  E_before ← calculate_energy(j)
  // Update junction position, see 3.1.3
  E_after ← calculate_energy(j)
  if E_before < E_after or junction is inverted then
    // Revert to old junction position
  end
  // Update cell pressures, see 3.1.3
end
```

Figure 3.13: The Vertex Model simulation algorithm in pseudo-code.

Figure 3.13 lists the algorithm in pseudo-code. All the individual parts of the algorithm should be well understood by now, except how to advance the diffusion process. Equation (2.4.5) defines the diffusion as change in area over time. I have chosen to take a simple Euler step so that the diffusion with respect to time becomes

$$\Delta A = \frac{2\pi}{3}\gamma\kappa\,(n-6)\,\Delta t \tag{3.7.1}$$

and the cell target area can then be updated to

$$A_c^\tau \leftarrow A_c^\tau + \Delta A\ .$$

There are well understood problems with using Euler integration, where we accumulate errors over time and thus loose accuracy, however I have deemed that in this case the Euler method is sufficient: As all cells target area will tend to zero over time in an arbitrary foam, the errors introduced by the Euler method will only change the speed with which the cells change area, which is an acceptable trade off for the simplicity of the implementation.

We now have all the information we need to implement a computer program for simulating dry foam with the Vertex Model. In the next chapter I will cover how such an implementation can be created and demonstrate how the presented algorithm can be parallelized.

Chapter 4

Implementation

In this chapter I will cover some of the practical details of implementating the Vertex Model in a computer program. Most notable I will introduce a method for equilibrating junctions in parallel, a performance optimization that promises to significantly decrease the running time of the simulation. To demonstrate the feasability of the method, I present a CUDA implementation of the Vertex Model.

4.1 Calculating the Jacobian

As mentioned in Section 3.1.1, computing ∇f_j is done with finite difference in a straight-forward manner. However, to minimize the error introduced by the finite difference calculation we use an backtracking line search scheme where h is iteratively decreased until the error is below a set threshold. This scheme was used by Kelager and the method used here closely matches that from [13]. Figure 4.1 lists the adaptive algorithm for computing ∇f_j in pseudo-code. In each iteration we calculate the difference between ∇f_j calculated with the current step size and the previous ∇f_j. If the difference is less than δ_{error} we assume that the algorithm has converged, otherwise the step size is halved and we perform another iteration.

4.2 The quasi-static simulation

Kelager identified that the stability of the simulation can be improved by applying a dampened line search to the Newton method. By taking smaller steps, more iterations are necessary before the algorithm converges, but the chance of convergence is increased [13, 19]. In the Newton method used here, this translates into decreasing $\mathbf{z}$ from (3.1.7):

$$\mathbf{t}^{k+1} = \mathbf{t}^k + \tau \mathbf{z}^k \ , \tag{4.2.1}$$

where $\tau \in \mathbb{R}^+$ is a constant used to control the step size. In general we assume $0 \leq \tau \leq 1$. Setting $\tau = 0$ will freeze the simulation, preventing any changes to the foam. We can apply the same backtracking line search scheme to τ as was used in Section 4.1, in this case with the Energy Decay Constraint and/or

$\mathbf{A} \leftarrow \mathbf{Z}$
$scale \leftarrow 1$
$error \leftarrow \infty$
while $error > \delta_{\text{error}}$ **and** $scale > \delta_{\text{scale}}$ **do**
  $\mathbf{A}_{\text{old}} \leftarrow \mathbf{A}$
  $\mathbf{A} \leftarrow compute\_jacobian(h \cdot scale)$
  $\Delta\mathbf{A} \leftarrow \mathbf{A} - \mathbf{A}_{\text{old}}$
  $error \leftarrow \|\Delta\mathbf{A}\|_\infty$
  $scale \leftarrow scale \cdot 0.5$
end

Figure 4.1: The adaptive algorithm for minimizing the error in the ∇f_j calculation. $\mathbf{Z}$ is a 5×5 matrix where every element is zero. δ_{error} is a user supplied constant, while $\delta_{\text{scale}} \ll 1$, which depends on the precision of the floating point representation, is used to ensure that the algorithm will always halt.

the Orientation Invariant Constraint as the controlling factor. Figure 4.2 lists the algorithm for updating junction positions with adaptive damping. The advantage of using an adaptive damping scheme is that some parts of the foam will inherently be more sensitive than others. Especially cells on the boundary and cells surrounded by six-sided cells will need more damping, as there are less freedom of movement in these areas.

As the quasi-static simulation is an iterative algorithm, where we repeat the equilibration process until we deem we are close enough to a (local) minima, we need to define a stopping criteria. In Section 3.1 it was mentioned that we can measure how close to a minima we are by $\|\mathbf{z}'\|_\infty$ or $\|F\|_\infty$. Since we do not compute neither $\mathbf{z}'$ nor F explicitly we can not use these directly, so a slightly different approach is used: After each junction is equilibrated we measure the maximum pressure change Δp and position change $\Delta\mathbf{x}$. These maximum values are stored in $\Delta p_{\max}$ and $\Delta\mathbf{x}_{\max}$. Then, in each iteration of the quasi-static algorithm, we can calculate the difference between the maximum values from the current and the previous iteration. If either of these difference is greater than some given threshold then we perform another iteration. Figure 4.3 lists the algorithm in pseudo-code. Ommited is an iteration counter used to break the iteration if the number of iterations exceeds a user-provided maximum. This ensures that the algorithm will always halt.

4.3 Topological operations

While the T1 and T2 operations are well understood both from a physical and a computational stand point, the question of *when* and *in which order* to perform said operations is less clear. As described in Section 3.2.2, T1 and T2 operations are near-instantaneous, but in the simulation we have to translate these instantaneous events into discrete occurances that can be carried out in sequence or in parallel. There are two sides to this question: A) Maintaining a physically correct model and B) performance of the simulation.

In the original implementation by Kelager [13], which was later reworked in

```
for each j in junctions do
  j_old ← j
  valid ← false
  scale ← 1
  while not valid and scale > δ_scale do
    E_before ← compute_energy(j)
    j.position ← j.position + τΔx_j · scale
    E_after ← compute_energy(j)
    if E_after < E_before and junctionnotinverted then
      valid ← true
    else
      j ← j_old
      scale ← scale · 0.5
    end
  end

  if valid then
    // Store τΔp_j · scale
  end
end
```

Figure 4.2: The algorithm for updating junction positions width adaptive damping. Note that although we only measure the energy of the junction position update, the scaled damping constant τ is also applied to the pressure changes. Here δ_{scale} is the same constant as in Figure 4.1.

$\Delta p_{\max} \leftarrow 0$
$\Delta \mathbf{x}_{\max} \leftarrow 0$
$error_p \leftarrow \infty$
$error_{\mathbf{x}} \leftarrow \infty$
while $error_p > \delta_{\Delta p}$ **and** $error_{\mathbf{x}} > \delta_{\Delta \mathbf{x}}$ **do**
  $\Delta p_{\max}^{\text{old}} \leftarrow \Delta p_{\max}$
  $\Delta \mathbf{x}_{\max}^{\text{old}} \leftarrow \Delta \mathbf{x}_{\max}$
  $\Delta p_{\max} \leftarrow 0$
  $\Delta \mathbf{x}_{\max} \leftarrow 0$
  for each j **in** *junctions* **do**
    // Compute f_j
    // Compute ∇f_j
    $\mathbf{z} \leftarrow solve(\nabla f_j, -f_j)$
    $\Delta p_{\max} \leftarrow \max(\Delta p_{\max}, \mathbf{z})$
    $\Delta \mathbf{x}_{\max} \leftarrow \max(\Delta \mathbf{x}_{\max}, \mathbf{z})$
  end
  // Apply changes
  $error_p \leftarrow \Delta p_{\max} - \Delta p_{\max}^{\text{old}}$
  $error_{\mathbf{x}} \leftarrow \Delta \mathbf{x}_{\max} - \Delta \mathbf{x}_{\max}^{\text{old}}$
end

Figure 4.3: The quasi-static algorithm with halting parameters.

[24], the approach was to test each junction for the possibility of performing a T2 operation on one of the three incident cells. If it was, the T2 was carried out and the next junction was examined, else the junction was tested for whether a T1 operation was possible on one of the incident films. If so, the T1 operation was carried out and the next junction examined. Finally, if neither a T2 nor a T1 operation was possible, the junction was equilibrated. Figure 4.4 lists the algorithm in pseudo-code. There is a number of problems with this approach: First, performing T2 operations might lead to other T2 operations becoming possible, as illustrated in Figure 4.5. Second, performing T1 operations can lead to new T2 operations becoming possible, which in turn can enable more T1 and T2 operations, etc. Figure 4.7 illustrates a T1 operation leading to a T2 operation. These cascades of topological operations are not captured by the described algorithm, which can lead us to attempt to equilibrate a badly conditioned mesh with tiny cells that should have been removed (T2) or reformed (T1) by the topological operations. Third, since we iterate over junctions we will test each film for a possible T1 operation twice (once for each end point) and each cell is tested for a possible T2 operation once for each junction in the cell! This adds up to a large number of unnecessary checks which slows down the simulation.

An improved algorithm could be to first check all cells for possible T2 operations and perform them as they where found, then test all films for possible T1 operations. This is repeated until no more topological operations are possible. Figure 4.6 lists the algorithm in pseudo-code. This algorithm performes all possible T2 operations then all possible T1 operations and then loops around

```
for each j in junctions do
  if T2 is possible on a cell c touching j then
    perform_T2(c)
  else
    if T1 is possible on a film f touching j then
      perform_T1(f)
    else
      equilibrate(j)
    end
  end
end
```

Figure 4.4: The original algorithm for performing topological operations in sequence.

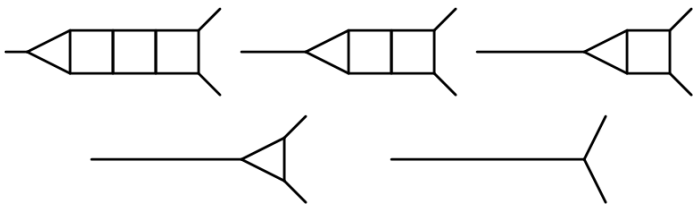


Figure 4.5: A T2 operation enables another T2 operation which in turn enables one more T2 operations, etc. The topological operations algorithm must capture all of these T2 operations.

```
t1 ← ∞
t2 ← ∞
while t1 > 0 or t2 > 0 do
  t1 ← 0
  t2 ← 0

  for each c in cells do
    if T2 is possible on c do
      perform_T2(c)
      t2 ← t2 + 1
    end
  end

  for each f in films do
    if T1 is possible on f
      perform_T1(f)
      t1 ← t1 + 1
    end
  end
end

for each j in junctions do
  equilibrate(j)
end
```

Figure 4.6: An improved but still not sufficient algorithm for performing topological operations in sequence.

iff any topological operations where performed. Only when no more topological operations can be performed are the junctions equilibrated. It is also ensures that no cell is tested for T2 and no film is tested for T1 more than once in each loop iteration.

There are, however, still problems with the updated algorithm: Each time a T1 operation have been performed, it might make a T2 operation possible. Not carrying out that T2 operation immediately might result in the T2 being "lost" due to a subsequent T1 operation. Figure 4.7 illustrates how two T1 operations can enable and then disable a T2 operation. There is no right or wrong in this situation: We are trying to put a number of simultaneous and instantaneous event into a sequence of discrete steps, so instead we must consider the problem from a practical, computational view point. In doing so, I will argue that it is better to perform as many T2 operations as possible because they acts as energy sinks, drawing energy out of the system, leading to a more stable simulation. To see why this is so, consider a foam with many tiny cells in a small area. Many T1 and T2 operations will be possible, but if we do not perform any T2 operations there is no room for the foam to grow the cells, so the foam will fluctuate as the same borders are flipped repeatedly in endless T1 cascades. Performing all possible T2 operations will give the foam better room to rearrange the junctions and thus leads to a more stable simulation. For this reason I will classify T2

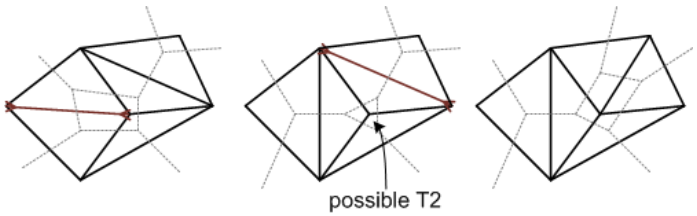


Figure 4.7: Two T1 operations in a row. The first T1 operation makes a T2 operation possible, but the second T1 operation removes the possible T2 operation.

operations as more computationally attractive than T1.

These considerations leads to a robust but more complex algorithm: By modifying the topological operations to return the new cells and films that are the result of the operation (three films and three cells in case of a T2 operation and two cells in case of a T1 operation,) we can enqueue these new films and cells to be tested again. After each T1 operation, the two resulting cells are tested for possible T2 operations which may again cause further T2 operations and so on. All these are captured by the improved algorithm listed in Figure 4.8.

When examining the final algorithm it becomes clear that it is not feasible to run topological operations in parallel. As each operation touches a neighborhood of the computational mesh, performing several operations in parallel can lead to race conditions and compromise the computational mesh. We therefore exclude the topological operations from the parallel part of the implementation and perform them exclusively in sequence on the CPU. While it is possibly that a scheme could be designed that allowed the operations in parallel, the topological operations appear to best suited for sequential consumption.

4.4 Parallelizing the simulation

The method presented in pseudo-code in Section 3.7 is, in essence, a Gauss-Seidel method. Each junction is equilibrated in sequence and the result of each equilibration is applied before the next junction is equilibrated. In this way, we use the intermediate results in the following calculations. In theory this can lead to a faster convergence, though this is of course not garanteed. However, Gauss-Seidel methods can not easily be parallelized. We would therefore rather use a Jacobi method, where the complete solution is calculated from previous values and then replaces them. As a consequence, the first step in parallelizing the Vertex Model was to determine what effect rewriting the Vertex Model algorithm to a Jacobi form would have.

4.4.1 The Vertex Model in Jacobi form

Rewriting the quasi-static simulation algorithm to Jacobi form is, in essence, a matter of identifying which parts of the algorithm can safely be run in parallel. Computing f_j and ∇f_j and solving $\nabla f_j \mathbf{z} = -f_j$ can be done in parallel. Applying the junction position changes from $\mathbf{z}$ can also be done in parallel, but applying the pressure changes can not be trivially parallelized, since $\mathbf{z}$ is per

$queue_{\mathrm{T1}} \leftarrow \emptyset$
$queue_{\mathrm{T2}} \leftarrow \emptyset$
for each c **in** $cells$ **do**
  $perform\_T2(c, queue_{\mathrm{T1}}, queue_{\mathrm{T2}})$
end
$process\_T2\_queue(queue_{\mathrm{T2}})$

$queue_{\mathrm{T1}} \leftarrow \emptyset$
for each f **in** $films$ **do**
  $perform\_T1(f, queue_{\mathrm{T1}}, queue_{\mathrm{T2}})$
end

while not $empty(queue_{\mathrm{T1}})$ **do**
  $f \leftarrow pop(queue_{\mathrm{T1}})$
  $perform\_T1(f, queue_{\mathrm{T1}}, queue_{\mathrm{T2}})$
end

for each j **in** $junctions$ **do**
  $equilibrate(j)$
end

Figure 4.8: The final algorithm for performing topological operations in sequence. This algorithm captures all topological cascades. The algorithm uses three auxillary functions which can be found in Figures 4.9, 4.10, and 4.11.

function $perform\_T1(f, queue_{\mathrm{T1}}, queue_{\mathrm{T2}})$
  $\{c_0, c_1\} \leftarrow perform\_T1(f)$
  $push(queue_{\mathrm{T2}}, c_0, c_1)$
  $process\_T2\_queue(queue_{\mathrm{T2}})$
end

Figure 4.9: Auxiliary function, see Figure 4.8.

function $perform\_T2(c, queue_{\mathrm{T1}}, queue_{\mathrm{T2}})$
  $\{f_0, f_1, f_2, c_0, c_1, c_2\} \leftarrow perform\_T2(c)$
  $push(queue_{\mathrm{T1}}, f_0, f_1, f_2)$
  $push(queue_{\mathrm{T2}}, c_0, c_1, c_2)$
end

Figure 4.10: Auxiliary function, see Figure 4.8.

```
function process_T2_queue(queue_T2)
  while not empty(queue_T2) do
    c ← pop(queue_T2)
    perform_T2(c, queue_T1, queue_T2)
  end
end
```

Figure 4.11: Auxiliary function, see Figure 4.8.

junction and several junctions will write to the same cells. Looking beyond the equilibration process, coarsening can update cell target areas in parallel and two-sided cells can be moved in parallel. There is a slight chance that two two-sided cells are next to each other, which will cause problems, but given that two-sided cells are rare and quickly disappears, I have deemed that the risk is acceptable when considering the performance gain. The topological processes can not easily be run in parallel, but must be done in sequence since a topological change affects an area around it and doing several in parallel can compromise the computational mesh. Figure 4.12 lists an updated pseudo-code showing the Vertex Model in a parallel-ready form. Note how each junctions change vector is calculated in parallel and changes are only applied afterwards in a non-parallel loop to avoid problems with the cell pressure update.

4.4.2 Verifying the Jacobi algorithm

Before committing to the parallel version of the Vertex Model algorithm, I tested what effect it would have on stability and convergence rate of the simulation. Note that the following measurements where made with an early version of the implementation. Some parts, such as the Energy Decay Constraint, was not implemented at that point. As such, the stability presented in this verification is not indicative of the complete algorithm. Both methods runs strictly in sequence on the CPU.

To compare the convergence rate of the Jacobi form compared to the Gauss-Seidel form, 1,000 foams, each with close to 2,000 cells initially, was constructed and the equilibration algorithm was run until each foam was fully equilibrated. Figure 4.4.2 shows various statistics comparing the two algorithms.

The most obvious result is that slightly more than twice the number of simulations was divergent with the Jacobi method (3.2% compared to 1.4%), which matches our expectation that the Jacobi method will be less robust than the Gauss-Seidel method. However, the number of divergent simulations is still low enough that I can conclude that the Jacobi form is viable.

It is interesting to note that both maximum and mean iterations of the Jacobi method is lower than the Gauss-Seidel. This is surprising, as we would expect the Gauss-Seidel method, which uses earlier results already in the same iteration, to converge faster.

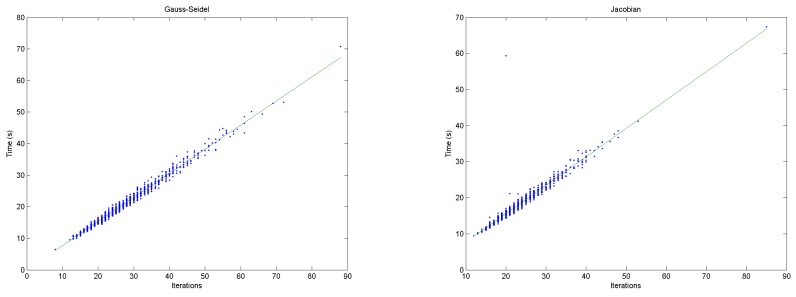
Figure 4.14 show number of iterations plotted against time for the two methods. They both show a perfectly linear correlation, which confirmes the linearity of the algorithm.

```
parallel for each c in Cells do
  A_c^τ ← diffusion(c, Δt)
end

parallel for each c in Cells do
  if valency(c) = 2 then
    move(c) // See 3.6.1
  end

// Perform topological operations, see 4.3

parallel for each j in Junctions do
  f_j ← calculate_constraints(j)
  ∇f_j ← calculate_jacobian(j)
  R[j] ← solve(∇f_j, −f_j)
end

for each j in Junctions do
  E_before ← calculate_energy(j)
  // Update junction position from R[j], see 3.1.3
  E_after ← calculate_energy(j)
  if E_before ≥ E_after then
    // Revert to old junction position
  end

  P[j.cell0] ← P[j.cell0] + R[j].pressure0
  P[j.cell1] ← P[j.cell1] + R[j].pressure1
  P[j.cell2] ← P[j.cell2] + R[j].pressure2
end

parallel for each c in Cells do
  // Update cell pressures from P[c], see 3.1.3
end
```

Figure 4.12: The Vertex Model quasi-static simulation algorithm in parallel form.

	Gauss-Seidel	Jacobi
Maximum iterations	88	85
Minimum iterations	8	12
Mean iterations	26.9077	23.1105
Maximum time	70.7393	67.3858
Minimum time	6.4417	9.3651
Mean time	20.5661	18.0340
Simulations	1000	1000
Divergent	14 (1.4%)	32 (3.2%)

Figure 4.13: Comparison of a Gauss-Seidel and a Jacobi method. Iterations is the number of times the equilibration algorithm was performed before the foam was fully equilibrated, time is the elapsed wall-clock time in seconds of performing these iterations. Simulations is the number of samples and divergent is the number of simulations that did not converge inside a given iteration threshold of 256 iterations.



(a) (b)

Figure 4.14: Plot of iterations against time for (a) the Gauss-Seidel form and (b) the Jacobi form. Note the strong linear correlation.

From this analysis I conclude that the Jacobi version is at least as stable and convergent as the Gauss-Seidel version.

4.5 A CUDA implementation

With powerful discrete Graphics Processing Units (GPU's) becoming common, both in consumer PC's and in high-performance computing (HPC), NVIDIA's Compute Unified Device Architecture (CUDA) have become a good starting point for creating a highly parallel foam simulation implementation. In the following I will present a CUDA implementation[1]. The implementation is not intended to be optimal but is meant as a proof-of-concept and as such has not been optimized and some parts are not implemented in the most performant manner.

4.5.1 A brief introduction to CUDA

In this section I will give a very brief introduction to the CUDA architecture. For an in-depth introduction see [21]. A CUDA device is a highly parallel general-purpose processor that, unlike the traditional consumer CPU, focuses on data processing and have very little data caching and flow control. CUDA devices are designed to process large amounts of data in parallel and can do so very fast, as long as the same operation can be performed on all of the data. Branching, on the other hand, is inefficient and should be avoided if at all possible. To get maximum performance out of the device, the computation should be designed to perform many arithmetic operations with minimal memory access, to hide a large memory-access latency.

A CUDA program splits data up into a number of threads that all perform the same operations. Threads are gathered together into blocks and blocks are spread out amoung the available processor cores. Each thread has a private memory, all threads in a block has access to a common *shared memory* and all blocks have access to *global memory*. Shared memory have on the order of 10 to 100 times faster access time than global memory and works as a manual cache for the global memory[2], however the amount of shared memory is severely limited.

In Vertex Model relaxation process, the arithmetically heavy operations of calculating f_j and ∇f_j are well suited to the CUDA architecture in that we perform the same calculation for all junctions. Unfortunately, as part of these calculations we need to access the three incident cells which can be of arbitrary valency. This complicates memory management and use of shared memory. The implementation used here is the most naive possible and shared memory is not used at all. However, as I will demonstrate in Section 6.3, the implementation is bottlenecked by the remaining sequential parts of the algorithm and so we can not expect a large performance gain from better memory utilization.

[1] The full source code is subject to publication and can be obtained by contacting this author.

[2] The Fermi CUDA architecure introduced an automatic cache.

```
struct σ⁰ {
  real pressure
  real target_area
  int valency
  int σ²
  bool mask
}
```

```
struct σ¹ {
  int twin
  int σ⁰
  int σ²
  bool mask
}
```

```
struct σ² {
  real x
  real y
  int σ¹₀
  int σ¹₁
  int σ¹₂
  bool mask
}
```

Figure 4.15: The structures defining the computational mesh. The structures are loosely based on a half-edge mesh, where each σ^1 is directed and has a twin going the opposite direction. The types represents floating point (**real**), integer (**int**), and boolean (**bool**) values. Integers are also used as indices.

4.5.2 Computational mesh representation

At the base of simulation is the computational mesh. It holds all the necessary data used by the simulation and its topology determines the shape of the simulated foam sample. I will adopt the terminology of simplical complexes and refer to vertices in the computational mesh as σ^0, edges as σ^1, and faces as σ^2. As the computational mesh is the dual of the foam, σ^0 corresponds to cells and holds the cells pressure and target area. Optionally we can also store the cells valency so we can avoid traversing the cell when the valency is needed. σ^1 corresponds to films. In a dry foam, films have no intrinsic physical values, so σ^1 are only used to maintain connectivity. It would be possible to store the arc radius with σ^1's, but in practical use it is more convenient to calculate the arc radius when needed. σ^2 corresponds to junctions and stores the embedded coordinates of the junction. Figure 4.15 summarises the computational mesh. The connectivity of the computational mesh is part of the structures. Each σ^0 has a representative σ^2 which can be used as starting point when traversing the films enclosing the σ^0. σ^1's, the glue that holds the computational mesh together, are directed edges[3], terminating in a junction (σ^2), that points to a cell (σ^0). Each σ^1 has a twin σ^1 which is the reversed film, pointing to the opposite σ^2. Finally, σ^2 points to the three σ^1 circling the junction. Note how this hardcodes the computational mesh to not violate the three-films-to-a-junction invariant.

Each σ also has a *mask* field. If we analyse the Vertex Model algorithm we will quickly realise that cells can only disappear; after the initial mesh has been created the number of cells (and thus films and junctions) will never increase. We can use this fact to streamline the memory allocation of the implementation. All necessary data can be allocated at the start of the simulation and, as T2 operations causes cells to disappear, the *mask* field is used to mask out σ that are no longer valid. In this way we can minimize costely memory operations.

The world cell, which must be handled seperatly in many aspects of the simulation, is included as a standard σ^0, but the index of this σ^0 is stored so that it can be filtered out when necessary.

[3]Similar to directed edges in a half-edge data structure.

4.5.3 Initializing the simulation

The first task we are faced with in the simulation is how to generate the initial foam. The dual computational mesh makes this task easier, as any triangulated mesh can be turned into a foam with just a few changes. In the presented implementation, a simple jittered grid was used as an initial point cloud: A vertex was placed in a random position in each cell of a 2D grid and a Delaunay triangulation was then used to create a mesh from this point cloud. However, in contrast to most common uses for a triangulated mesh, we will need the positions stored not in the vertices but inside the triangles. I therefore, as the computational mesh is constructed, compute the barycenter of each triangle and store that in the σ^2's. For each σ^0, the initial pressure is set to 1 and the target area to the area of the cell.

There is another step that, dependent on the mesh and the intended usage, may be necessary: We need to connect the mesh to the world cell. All cells that should lie on the boundary of the foam sample when initiating the simulation must be connected to a single σ^0 in the computational mesh and this σ^0 is then designated the world cell [24].

I have used the method described in [24] to connect all boundary cells with the world cell by adding "fins" to the computational mesh before the world cell is assigned. A single triangle (a fin) is appended to all triangles that have a boundary edge in the computational mesh. The tip of these fins are then replaced with a single vertex, the world cell. Figure 4.16 shows the steps from initial triangulated mesh to computational mesh, with fins added to connect the boundary to the world cell, to foam sample. The position of the new vertex in the fin can be computed as

$$\mathbf{x}_{\text{free}} = \frac{\mathbf{x}_0 + \mathbf{x}_1}{2} + s\frac{\widehat{\mathbf{x}_1 - \mathbf{x}_0}}{\|\mathbf{x}_1 - \mathbf{x}_0\|} ,$$

where $\mathbf{x}_{0,1}$ are the two end points of the boundary edge, as illustrated in Figure 4.17, and $s \in \mathbb{R}^+$ is a scalar controlling the size of the fin. Here $\widehat{\mathbf{x}} = \widehat{\{x, y\}} = \{-y, x\}$. Note that fins are added to the computational mesh before the barycenter is computed.

We now have our initial computational mesh in an unstable state. We will need to make an initial equilibration before we can use the foam. Figure 4.18 shows a foam sample before and after the initial equilibration.

Before we start the simulation there are several constants that we need to consider. Figure 4.19 lists the relevant values. In the experiments and examples shown in this thesis the values of these constants where chosen to insure stability and convergence, not to match any real world material or medium. As such, the values should be taken as no more than suggestions.

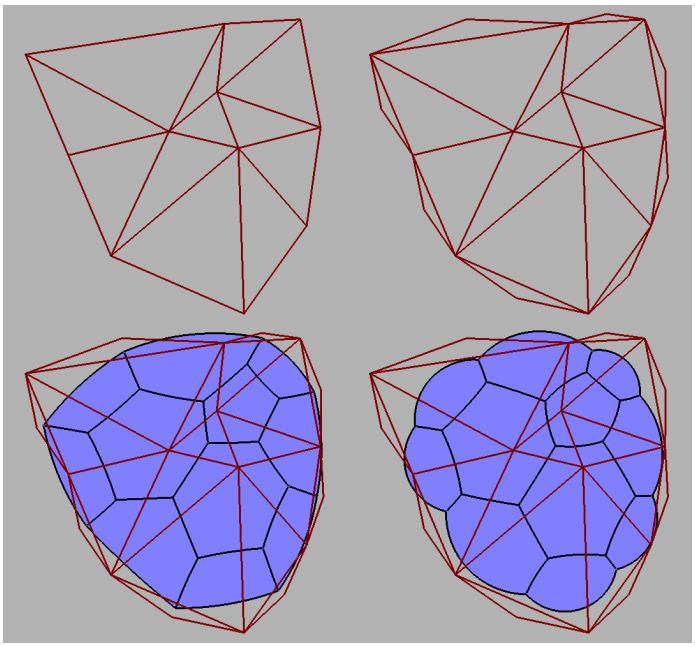


Figure 4.16: The stages in converting an initial mesh to a computational foam mesh. From left to right, top to bottom: The initial mesh, the initial mesh with fins added, the dual mesh, the foam after initial equilibration. Note that the initial mesh has not been updated to reflect the initial equilibration. This image is a reproduction of an image in [24].

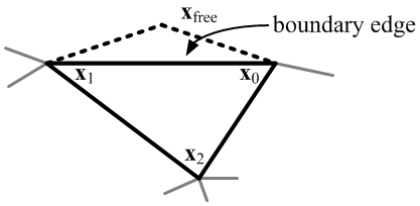
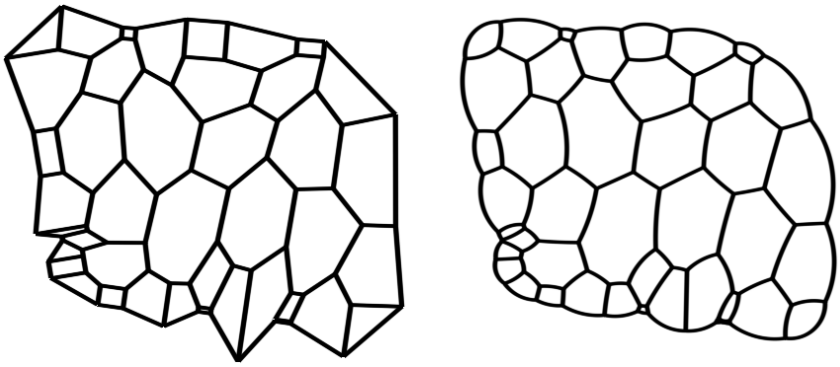


Figure 4.17: A boundary triangle in the computational mesh.



(a) (b)

Figure 4.18: (a) The initial, unequilibrated foam. (b) The same foam, after it has been equilibrated.

Constant	Description	Reference	Value
γ	Surface tension	2.2	$0.25\frac{\mathrm{N}}{\mathrm{cm}}$
κ	Film permiability for gas diffusion	2.4	$1\frac{\mathrm{cm}^2}{\mathrm{s}}$
τ	∇f_j calculation damping	4.2	0.1
h	Finite difference step size	3.1.1	0.01
δ_{error}	Error threshold in the adaptive ∇f_j calculation.	4.1	1
iterations	The maximum number of iterations of the quasi-static simulation	4.2	256
$\delta_{\Delta p}$	The quasi-static simulation minimum pressure change	4.2	$0.001\frac{\mathrm{N}}{\mathrm{cm}^2}$
$\delta_{\Delta \mathbf{x}}$	The quasi-static simulation minimum position change	4.2	0.001cm
δ_{T1}	T1 distance threshold	3.2.2	0.025cm
δ_{T2}	T2 area threshold	3.2.2	$0.05\mathrm{cm}^2$
Δt	Time step size	3.7	$\frac{1}{30}\mathrm{s}$
δ_{scale}	The minimum scale factor in backtracking line searches	4.1 & 4.2	10^{-5}

Figure 4.19: A summary of the constants used in the quasi-static simulation. The Reference column gives references to the sections in this thesis that explains the usage of these constants. The Value column are suggested values used in the experiments shown in this thesis. Note that they are chosen to give the desired result, not to match any real-world material properties.

Chapter 5

Rendering of Foam Films

When it comes to generating images of foams, most implementations focusing on the physics of foams have been limited to two-tone line drawings. In the following I will demonstrate a method for drawing foams that, while still a line drawing, provides higher flexibility than simple line drawing, by utilizing the current generation of programmable Graphics Processing Units (GPU's.)

The output of the simulator is a collection of edges, each with a start and an end point and an arc radius. Given these, our problem is to render an arc for each edge, a task perfectly suited for the Geometry Shader (GS) found on modern GPU's.

Consider an edge with end points $\mathbf{x}_0$ and $\mathbf{x}_1$ and arc radius $r \geq \frac{\|\mathbf{x}_1 - \mathbf{x}_0\|}{2}$ (refer back to Section 3.3 for an explanation of this constraint). For convenience we shall only consider edges with positive arc radius; this is reasonable, since the sign of the radius of a film can be reversed if we also swap the end points. We now wish to draw the edge as an arc with radius r. To achieve this we will subdivide the edge into a number of discrete line segments. Let $\mathbf{x}_2$ be the center point in a circle with radius r such that points $\mathbf{x}_0$ and $\mathbf{x}_1$ lie on the boundary of the circle. $\mathbf{x}_0$ and $\mathbf{x}_1$ must then be equidistant from $\mathbf{x}_2$ and $\triangle \mathbf{x}_0\mathbf{x}_1\mathbf{x}_2$ is an isosceles triangle with two sides of length r and one of length $d = \|\mathbf{x}_1 - \mathbf{x}_0\|$ as illustrated in Figure 5.1. The angle between the two line segments meeting at $\mathbf{x}_2$ is

$$\theta = 2\sin^{-1}\left(\frac{d}{2r}\right) \ . \tag{5.0.1}$$

In practical terms, equation (5.0.1) is problematic: As $r \to \infty$, $r \gg d$ so we lose precision in the IEEE-754 single precision floating point representation used by the GPU [20]. Instead we would prefer the numerator and denominator of the fraction to be of the same order of magnitude, which we can achieve by replacing (5.0.1) with

$$\theta = 2\cos^{-1}\left(\frac{h}{r}\right) \ , \tag{5.0.2}$$

where $h = \|\frac{\mathbf{x}_0 + \mathbf{x}_1}{2} - \mathbf{x}_2\|$ is the distance of $\mathbf{x}_2$ from the midpoint of the straight edge from $\mathbf{x}_0$ to $\mathbf{x}_1$. Because h is on the same order of magnitude as r we will get a better precision.

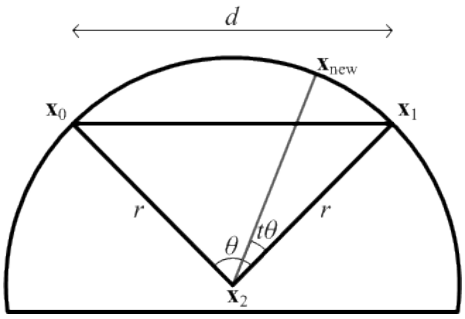


Figure 5.1: Sampling an arc for rendering. By rotating a vector from $\mathbf{x}_2$ to $\mathbf{x}_1$ by $t\theta$ degree we find a point on the arc with radius r.

We can now find any point on the arc between $\mathbf{x}_0$ and $\mathbf{x}_1$ by rotating a vector $\mathbf{x}_1 - \mathbf{x}_2$ by an angle $0 \leq t\theta \leq \theta$:

$$\mathbf{x}_{\text{new}} = \mathbf{x}_2 + \mathbf{R}\left(t\theta\right)\left(\mathbf{x}_1 - \mathbf{x}_2\right)^{\mathrm{T}} , \tag{5.0.3}$$

where $0 \leq t \leq 1$ controls how far along the arc we go and

$$\mathbf{R}(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

is the normal 2D rotation matrix. By calculating $\mathbf{x}_{\text{new}}$ for different values of t we get a set of discrete points which we can connect to get a piecewise approximation to the arc. (Note the similarity to the method used to move two-sided cells.)

Now that we can create a piecewise approximation of the arc we are ready to construct an algorithm for rendering the arcs. The goal is to construct a ribbon of triangles, smoothly approximating the arc. The triangles should form quadrilaterals, rectangles formed by joining two triangles (quads). We can create a single quad by sampling the arc in two places:

$$\mathbf{x}^a_{\text{new}} = \mathbf{x}_2 + \mathbf{R}\left(t\theta\right)\left(\mathbf{x}_1 - \mathbf{x}_2\right)^{\mathrm{T}} \tag{5.0.4}$$

$$\mathbf{x}^b_{\text{new}} = \mathbf{x}_2 + \mathbf{R}\left((t+\delta t)\theta\right)\left(\mathbf{x}_1 - \mathbf{x}_2\right)^{\mathrm{T}} \tag{5.0.5}$$

where $0 \leq t < t + \delta t \leq 1$. The smaller we choose δt to be, the less an area of arc the quad will cover and thus we need more quads to fully cover the arc but we get a better approximation of the arcs curve. To construct the four corners of the quad we find two unit vectors

$$\mathbf{v}^a = \frac{\mathbf{x}^a_{\text{new}} - \mathbf{x}_2}{\left\|\mathbf{x}^a_{\text{new}} - \mathbf{x}_2\right\|} , \tag{5.0.6}$$

$$\mathbf{v}^b = \frac{\mathbf{x}^b_{\text{new}} - \mathbf{x}_2}{\left\|\mathbf{x}^a_{\text{new}} - \mathbf{x}_2\right\|} \tag{5.0.7}$$

which we use to create points under and over the arc:

$$\mathbf{x}^0_{\text{quad}} = \mathbf{x}^a_{\text{new}} - \frac{1}{2}\mathbf{v}^a c_{\text{width}} \ , \tag{5.0.8}$$

$$\mathbf{x}^1_{\text{quad}} = \mathbf{x}^a_{\text{new}} + \frac{1}{2}\mathbf{v}^a c_{\text{width}} \ , \tag{5.0.9}$$

$$\mathbf{x}^2_{\text{quad}} = \mathbf{x}^b_{\text{new}} - \frac{1}{2}\mathbf{v}^b c_{\text{width}} \ , \tag{5.0.10}$$

$$\mathbf{x}^3_{\text{quad}} = \mathbf{x}^b_{\text{new}} + \frac{1}{2}\mathbf{v}^b c_{\text{width}} \ , \tag{5.0.11}$$

where $c_{\text{width}} \in \mathbb{R}^+$ is a user supplied constant used to control the width of the quad and thus the width of the arc.

When a ribbon of quads have been formed we can create different rendering effects as they are rendered. The simplest, and the method used in the figures in this thesis, is to calculate texture coordinates that stretch over the ribbon. A texture can then be applied to create the desired effect. Another option would be to generate colors in the Pixel Shader based on some criteria. For example, Bærentzen *et al.* demonstrated in [4] how a distance map can be used to create smooth, supersampled lines by calculating a distance field over the rendered N-gons. Their method would be very well suited for this application, as the required information is already present in the Geometry Shader.

5.1 A GPU arc render

In the previous section we found a way of calculating the corners of quads to approximate an arc. So how do we go about generating and rendering these quads? The answer depends, of course, on the chosen rendering API, but in my case the goal was a method that works with modern GPU's. This makes the Geometry Shader (GS) a perfect choice. The GS comes between the Vertex Shader (VS) and Pixel/Fragment Shader (PS) and takes as input from the VS a single primitive (point, line, or triangle), optionally with adjacency information. The output is a set of new primitives which are then passed on to the PS. The strength for our purpose is that we can send a single primitive per film and have the GS generate the arc quads.

Some thoughts must be made on how to send data from the simulation to the GS. A set of line primitives is the natural first choice, but we also need the radius of the arc. We could store the radius in the vertices, but then we must duplicate the radius information on the two vertices in a line primitive and we can not reuse vertices between adjacent films. Another option would be to store the radii in a buffer on the GPU and do a look up in the GS based on a primitive ID. However, this would require that we transfer two blocks of data to the GPU instead of one, which would be slower. Instead I have chosen a different approach and constructs a triangle primitive for each edge, where the third point is $\mathbf{x}_2$ from the previous section. This way we only need to transfer once to the GPU, there are no redundant information, and the calculations in the GS becomes slightely simpler as we don't need to calculate $\mathbf{x}_2$ in the GS. The radius can then be calculated simply as the euclidian distance of $\mathbf{x}_2$ from one of the other two vertices. I shall explain later how to obtain the $\mathbf{x}_2$ point. This approach is ideally suited to the concept of *index/vertex buffers* which is the prefered

method for presenting primitives to the GPU. In this approach, a vertex buffer (Vertex Buffer Object in OpenGL, Vertex Buffer in Direct3D) is allocated for vertex data and another for index data (Index Buffer in Direct3D), where the index buffer holds indices used to construct a primitive from the vertex buffer data. This allows the GPU to process primitives fast and efficiently.

5.1.1 Image space rendering

Before presenting an implementation we must consider which coordinate system we are working in. So far in this thesis we have implicitly assumed that all coordinates and calculations are in the same global coordinate system, where origo is the center of the plane on which the foam is simulated. We shall adopt a common convention and call this the *world space* coordinate system or simply world space (in OpenGL the term *model space* is commonly used.) This is a natural and obvious choice for simulation, but it has some subtle consequences for our arc render: If we calculate $\mathbf{v}^{a,b}$ and define c_{width} in world space, then the width of the arcs, when rendered, will depend on the foams distance to the camera. If the camera is close to the foam the arcs will be broad, but as the camera moves away they will dwindle away until they can no longer be seen. While this may be the desired behaviour in some applications, here we wish for the arcs to remain visible and have the same width no matter the camera position, so that we can inspect any detail of the foam and get the same visual result.

Before I present a solution to this problem, let us briefly review the process of transforming from 3D world space coordinates to 2D screen space, ready for display on a computer screen. In the following I shall in most aspects adopt the terminology used by Microsoft Direct3D [16], but the terms and concepts are, in general, agnostic to the specific graphics application programming interface (API) used. So far we have been working in $\{x, y\} \in \mathbb{R}^2$ 2D coordinates through-out, but to utilize the GPU we must work with $\{x, y, z\} \in \mathbb{R}^3$ 3D coordinates. In 3D we can represent affine transformations, such as translation, scaling, rotation, and shearing, as 4×4 matrices. To apply such a transformation to an $\mathbb{R}^3$ coordinate we must extend the vector to $\mathbb{R}^4$, which we do by working in *homogeneous coodinates* $\{x, y, z, w\} \in \mathbb{R}^4$ [9]. When a primitive is rendered by the GPU it goes through five steps [9, 16]:

1. The coordinates are extended with a fourth ordinate: $\mathbf{x}^{\text{H}} = \{\mathbf{x}^x, \mathbf{x}^y, \mathbf{x}^z, 1\}$ to form homogeneous coordinates.

2. The coordinates are transformed from *model space* (or *object space*) into world space via the matrix $\mathbf{T}^{\text{world}}$. This step transforms the model being rendered from its local coordinate system to the global world coordinate system. In our foam rendering the foam is already in world space and so $\mathbf{T}^{\text{world}} = \mathbf{I}$, the 4×4 identity matrix.

3. The coordinates are transformed from world space to *view space* (or *camera space*) via $\mathbf{T}^{\text{view}}$. This transforms the coordinates into the space defined by the virtual camera through which we observe the scene.

4. The coordinates are transformed from view space to the *canonical view volume* via $\mathbf{T}^{\text{projection}}$. This transformation projects the coordinates unto

the view plane, which is the image that is displayed on the computer screen (hence projection transformation).

5. Coordinates are projected back into 3D as $\{\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\} \in \mathbb{R}^3$. The coordinates are now in *image space* and the z-ordinate is the distance of the point from the projection plane.

The last step in this process is crucial to the problem of rendering lines with equal weight under all projections. In image space all distances are final, so by generating homogeneous coordinates with $w = 1$, the x and y ordinates remain unchanged and we can control the final size of rendered elements.

However, we can not simply do all calculations in image space: When the projection is applied, the aspect ratio of the rendered image is also applied. As a consequence, the distance between the three vertices of the triangle depends on their angle in relation to each other, with the result that we do not get the correct radius when calculating the euclidian distance between the vertices. The answer is to do all calculations up to $\mathbf{x}_{\text{new}}$ in world space, then transform $\mathbf{x}_{\text{new}}$ into image space and carry on from there. We will still get a slight distortion of the final image because of aspect ratio, namely the width of the rendered lines will be slightly larger than the height. If desired we could avoid this by dividing the width by the aspect ratio, but I have found the slight distortion acceptable.

5.1.2 The Geometry Shader

We now have all the necessary pieces to create a GS program to render arcs. Figure 5.2 lists the Geometry Shader algorithm in pseudo-code, while Appendix B.2 lists the full code. Figure 5.5 shows a single edge rendered with the presented method and Figure 5.6 shows a section of a foam with the films rendered with the method. The program is a straight-forward implementation of the theory presented in this and the previous sections. First θ is calculated using Equation (5.0.2), then the number of segments (quads) is calculated. Notice that s_{length} (`segment_size` in the code,) the length of a quad, is calculated in image space to ensure that quads always have the same length. Finally the quads are generated in the *gs_arc_edge* (`GS_ArcEdge`) helper function listed in figure 5.4. Using the Geometry Shaders ability to output triangle strips, we only generate two vertices per quad plus two vertices to start the strip. For each vertex pair (over and under the arc,) $\mathbf{x}_{\text{new}}$ is calculated using equation (5.0.4) and the point is transformed into image space before the final positions are calculated using Equations (5.0.8)-(5.0.11). By setting $w = 1$ in the final positions, we ensure that the points are in image space. Note that we calculate points "backwards" from $\mathbf{x}_1$ to $\mathbf{x}_0$. Therefore we must also remember to reverse to texture coordinates.

In the presented GS it is necessary and important to recognize what happens when $r \to \infty$. The ratio $\frac{h}{r} \to 1$, which presents us with a similar precision problem as when calculating θ. The result is that θ is not calculated sufficiently precise for very large r values, when the edge is close to straight in other words, and therefore the edge starts to over- or undershoot the end points which leads to very noticable rendering artifacts. The simple solution is to detect that $\frac{h}{r}$ is close to one and render a single straight quad instead. In the simulation, the special value of "`+INF`" (Positive Infinity) is used to signify a perfectly straight edge. When we divide by $r =$ `+INF` the result is the special value "`NaN`" (Not a Number). We can detect this value and again render a single straight quad.

function $gs(IN[3])$
  // Calculate position in image space
  $\mathbf{x}_0 \leftarrow \frac{IN[0].position_{xyz}}{IN[0].position_w}$
  $\mathbf{x}_1 \leftarrow \frac{IN[1].position_{xyz}}{IN[1].position_w}$
  $\mathbf{x}_2 \leftarrow \frac{IN[2].position_{xyz}}{IN[2].position_w}$

  $\mathbf{v}_{\text{normal}} \leftarrow IN[1].world - IN[2].world$
  $radius \leftarrow \|\mathbf{v}_{\text{normal}}\|$
  $\mathbf{x}_m \leftarrow 0.5 \cdot (IN[0].world + IN[1].world)$
  $h \leftarrow \|\mathbf{x}_m - IN[2].world\|$
  $ratio \leftarrow \frac{h}{radius}$
  if $is\_NaN(h)$ **or** $ratio > \delta_{\text{error}}$ **then**
    $gs\_straight\_edge\,(\mathbf{x}_0, \mathbf{x}_1)$
  else
    $gs\_arc\_edge\,(IN, \mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, ratio, \mathbf{v}_{\text{normal}})$
  end
end

Figure 5.2: The arc render Geometry Shader in pseudo-code. A listing of the actual code can be found in Appendix B.2. "IN" is function parameter from the Vertex Shader which contains the position of the three vertices in a triangle primitive with World Space position and post-projection space position. The function uses two auxillary functions which can be found in Figures 5.3 and 5.4. δ_{error} is a constant used to control how large the radius of the edge must be before we consider the edge straight. A value of $\delta_{\text{error}} = 0.99998$ (single precision floating point) have been used in the example renderings in this thesis.

function $gs\_straight\_edge(\mathbf{x}_0, \mathbf{x}_1)$
  $\mathbf{v} \leftarrow \mathbf{x}_1 - \mathbf{x}_0$
  $\mathbf{p}_0 \leftarrow \{\mathbf{x}_0 - 0.5\mathbf{v} \cdot s_{\text{thickness}}, 1\}$
  $\mathbf{p}_1 \leftarrow \{\mathbf{x}_0 + 0.5\mathbf{v} \cdot s_{\text{thickness}}, 1\}$
  $\mathbf{p}_2 \leftarrow \{\mathbf{x}_1 - 0.5\mathbf{v} \cdot s_{\text{thickness}}, 1\}$
  $\mathbf{p}_3 \leftarrow \{\mathbf{x}_1 + 0.5\mathbf{v} \cdot s_{\text{thickness}}, 1\}$
  $make\_quad(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$
end

Figure 5.3: Auxillary function used by the arc rendering Geometry Shader to straight edges. $s_{\text{thickness}}$ dictates the width of the arc ribbon.

function $gs\_arc\_edge(IN, \mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, ratio, \mathbf{v}_{\text{normal}})$
  $\theta \leftarrow 2\cos^{-1}(ratio)$
  $\mathbf{p}_0 \leftarrow \mathbf{0}$
  $\mathbf{p}_1 \leftarrow \mathbf{0}$
  $segments \leftarrow \max\left(2, \min\left(63, floor\left(\frac{\|\mathbf{x}_1 - \mathbf{x}_0\|}{s_{\text{length}}}\right)\right)\right)$
  for $i \leftarrow 0$ *to* $segments + 1$ **do**
    $t \leftarrow \frac{i}{segments}$
    $\mathbf{v}_{\text{rotated}} \leftarrow \{\cos(t\theta)\mathbf{v}^x_{\text{normal}} - \sin(t\theta)\mathbf{v}^y_{\text{normal}},$
      $\sin(t\theta)\mathbf{v}^x_{\text{normal}} + \cos(t\theta)\mathbf{v}^y_{\text{normal}}\}$
    $\mathbf{x}_{\text{new}} \leftarrow IN[2].world + \mathbf{v}_{\text{rotated}}$
    $\mathbf{x}_{\text{IS}} \leftarrow \{\mathbf{x}_{\text{new}}, 1\} \cdot \mathbf{T}^{\text{IS}}$
    $\mathbf{x}_{\text{IS}} \leftarrow \frac{\mathbf{x}_{\text{IS}}}{\mathbf{x}^w_{\text{IS}}}$
    $\mathbf{v} \leftarrow \frac{\mathbf{x}_{\text{IS}} - \mathbf{x}_2}{\|\mathbf{x}_{\text{IS}} - \mathbf{x}_2\|}$
    $\mathbf{p}_2 \leftarrow \{\mathbf{x}_{\text{IS}} - 0.5\mathbf{v} \cdot s_{\text{thickness}}, 1\}$
    $\mathbf{p}_3 \leftarrow \{\mathbf{x}_{\text{IS}} + 0.5\mathbf{v} \cdot s_{\text{thickness}}, 1\}$
    if $i > 0$ **then**
      $make\_quad(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$
    end
    $\mathbf{p}_0 \leftarrow \mathbf{p}_2$
    $\mathbf{p}_1 \leftarrow \mathbf{p}_3$
  end
end

Figure 5.4: Auxillary function used by the arc rendering Geometry Shader to render arcs. $\mathbf{T}^{\text{IS}}$ is a matrix that transforms World Space positions into image space. The constant 63 used when computing *segments* is dictated by maximum number of output vertices allowed by the GPU. $s_{\text{thickness}}$ dictates the width of the arc ribbon and s_{length} the length of individual quads in the ribbon.

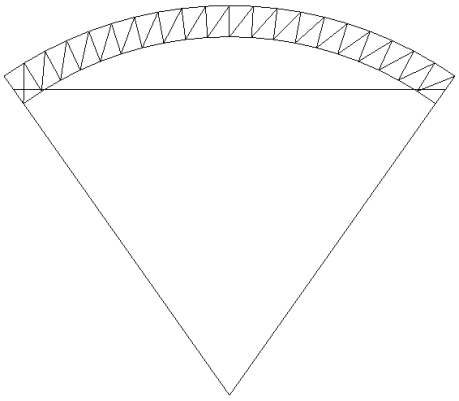


Figure 5.5: An example of a single edge rendered as an arc, with the triangle used to represent the edge.

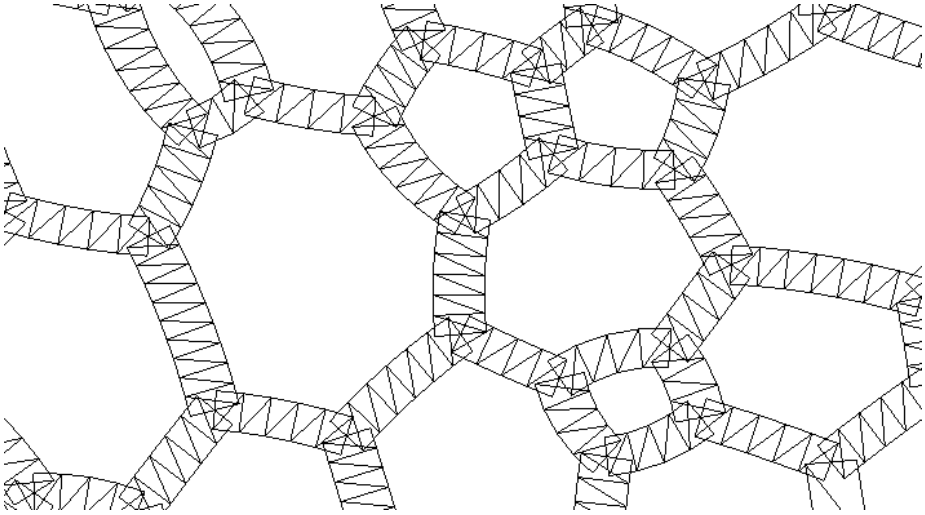


Figure 5.6: An example of films in a foam rendered as arc ribbons.

5.2 Constructing edge triangles

The final missing piece from the arc rendering algorithm is how to construct the triangles sent to the GS from edges represented as two end points, $\mathbf{x}_0$ and $\mathbf{x}_1$, and a radius r. This is, however, quite simple: Let $\mathbf{x}_m = \frac{\mathbf{x}_0+\mathbf{x}_1}{2}$ be the midpoint of the straight line edge, and let $\mathbf{e} = \mathbf{x}_1 - \mathbf{x}_0$ be a vector parallel to the straight line edge in direction from $\mathbf{x}_0$ to $\mathbf{x}_1$. Let further more $h = \frac{\|\mathbf{e}\|}{2}$ and $k = \sqrt{r^2 - h^2}$ (we disregard negative k under our earlier assumption that r is positive.) Using the fact that we are in a 2D plane, let $\mathbf{l} = \widehat{\mathbf{e}} = \{\widehat{\mathbf{e}^x, \mathbf{e}^y}\} = \{\mathbf{e}^y, -\mathbf{e}^x\}$ be a vector perpendicular to $\mathbf{e}$. We can now calculate

$$\mathbf{x}_2 = \mathbf{x}_m + k\frac{\mathbf{l}}{\|\mathbf{l}\|} \ ,$$

which gives us the third point in the triangle $\triangle\mathbf{x}_0\mathbf{x}_1\mathbf{x}_2$.

5.3 Examples

Figures 5.7 and 5.8 show a foam with arcs rendered with a "DIKU" texture used as films. Figure 5.9 shows a more artistic line style, while figure 5.10 shows how a wet foam can be approximated by decorating a dry foam [25]. Note how the junctions in figure 5.10 does not join smoothly. This is a consequence of rendering each arc in separation. Referring back to Figure 5.6 we see why this is so: Where the ribbon of quads overlap a discontinuity is introduced. It is this discontinuity that is visible on the wide junctions of Figure 5.10. The size and visual impact of the discontinuity depends on the texture used and width of the arc ribbon. Figure 5.11 shows an example of a foam rendered with OpenGL line primitives, the method used in [13] by Kelager. Notice the highly aliased films. In constrast the new method described here produces, depending on the texture used, smooth anti-aliased films.

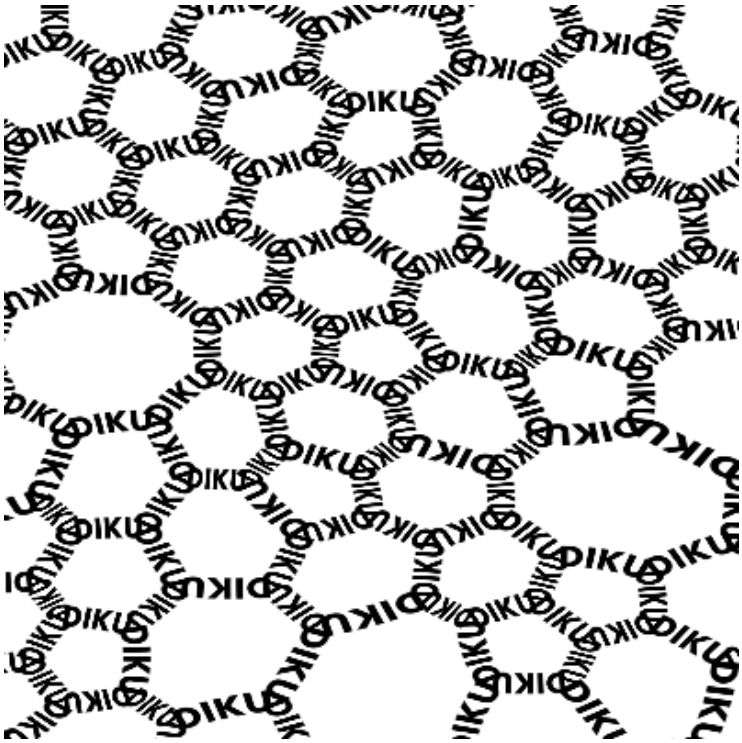


Figure 5.7: Example of a foam with a "DIKU" texture used as films. Segment length = 0.01, segment width = 0.05.

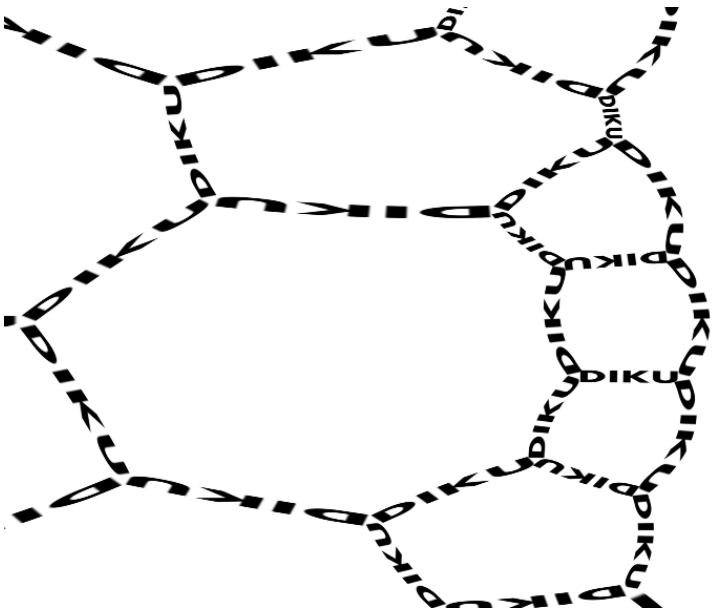


Figure 5.8: Example of a foam with a "DIKU" texture used as films. Segment length = 0.01, segment width = 0.05.

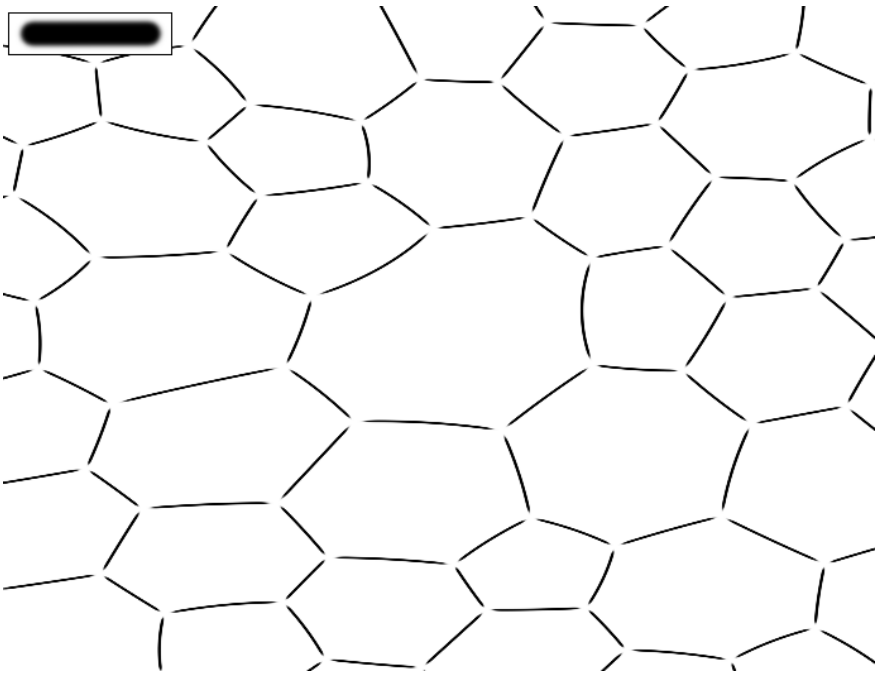


Figure 5.9: Example of a foam arc rendering style. The insert shows the texture used.

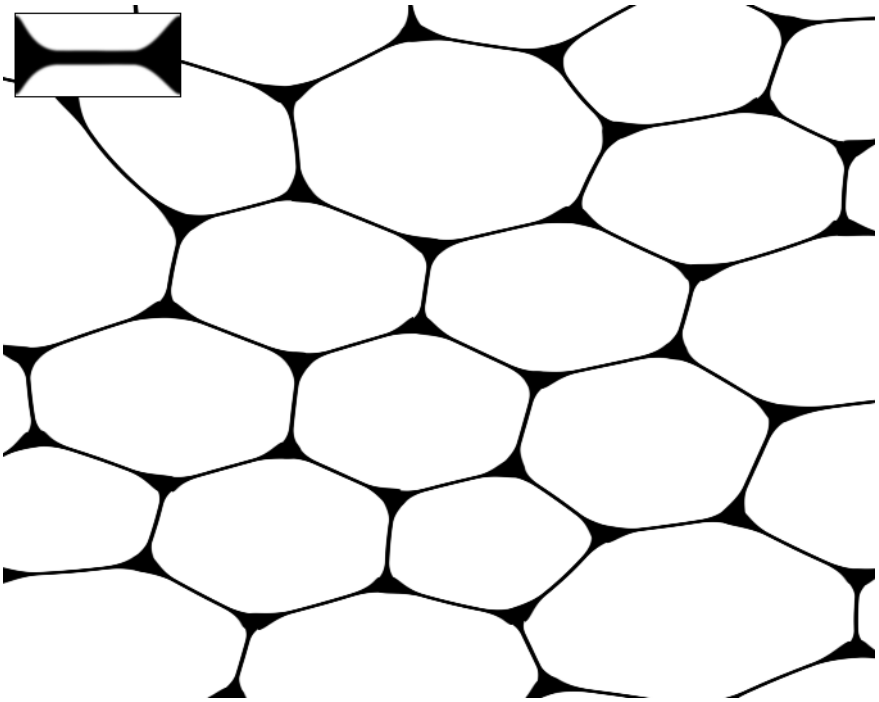


Figure 5.10: Example of a foam arc rendering style. The insert shows the texture used. Note the artifacts at junctions.

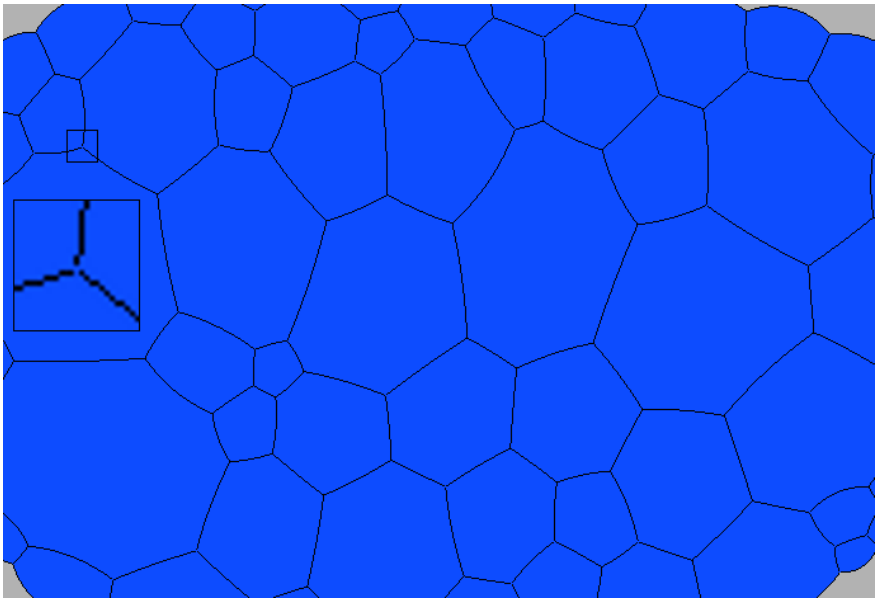


Figure 5.11: A foam rendered with the method used by Kelager. Notice the thin, highly aliased films.

Chapter 6

Results

In the following I will present the results of simulations of foams of different sizes, including essential statistics. The goal is to demonstrate that the current implementation is at least as correct as what has been previously published. All measurements have been made with the CUDA implementation presented in Chapter 4. Video sequences showing foams can be found online at: `http://infinitemonkey.dk/foam.html`.

6.1 Foam evolution

In the following I will present several experiments with the purpose of demonstrating and verifying the behavior of the Vertex Model while a foam is subjected to coarsening.

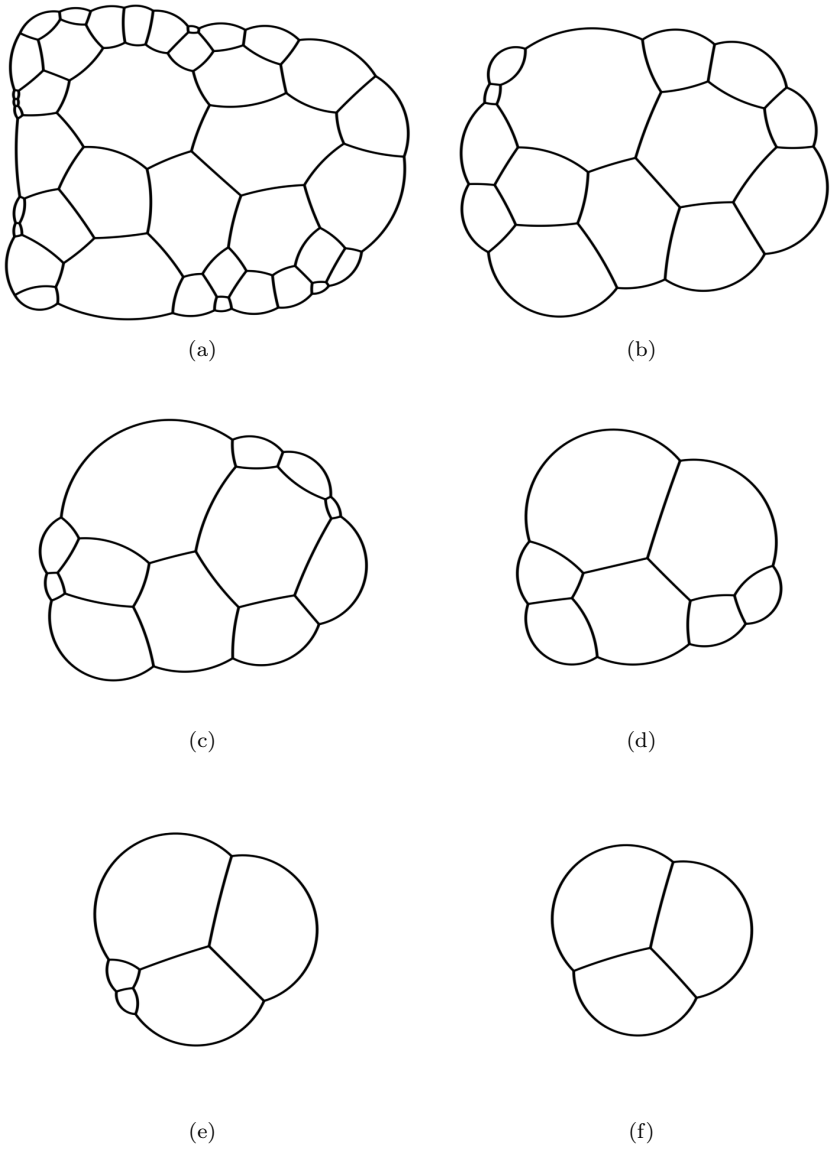
6.1.1 Lower extreme

In the lower extreme, three cells (not including the world cell) is the least number of cells the simulation will support. A configuration of two two-sided cells can exist, but the simulation will be unstable. Figure 6.1 shows a sequence of 40 cells coarsening and collapsing into a stable three-cell configuration. The final arrangement of cells matches the expected shape, where the three cells of near-equal pressure meet with a single central junction of 120° [23].

6.1.2 Development with the Energy Decay Constraint

To examine the evolution of a foam sample over a large time scale, I constructed a foam with initially 20.906 cells using the jittered-grid method described earlier. The foam was allowed to coarsen over a periode of 60.000 frames of $\Delta t = \frac{1}{30}$s, equating to 33 minutes of simulated time, with the Energy Decay Constraint, but not the Orientation Invariant Constraint. Figure 6.2 lists the constants used. After 60.000 frames the foam was reduced to 413 cells. Figure 6.3 shows a sequence of six frames from the simulation. Note the anomalies that develop on the free boundary of the foam as it evolves. I shall return to and describe these anomalies later.

Figure 6.4 shows the development in total number of cells. In this and following figures, we differentiate between the foam measured with and without cells



(a) (b)

(c) (d)

(e) (f)

Figure 6.1: A foam sample with 40 cells are allowed to evolve over 1.000 frames until it is stable with three cells left. With just three cells the Energy Decay Constraint holds the foam rigid and unchanging.

γ	$0.25\frac{\mathrm{N}}{\mathrm{cm}}$	κ	$1\frac{\mathrm{cm}^2}{\mathrm{s}}$
τ	0.1	h	0.01
δ_{error}	1	iterations	256
$\delta_{\Delta p}$	$0.001\frac{\mathrm{N}}{\mathrm{cm}^2}$	$\delta_{\Delta \mathbf{x}}$	0.001cm
δ_{T1}	0.025cm	δ_{T2}	0.05cm^2
Δt	$\frac{1}{30}\text{s}$	δ_{scale}	10^{-5}

Figure 6.2: Constants used in the experiments. Refer to Figure 4.19 for descriptions of the constants.

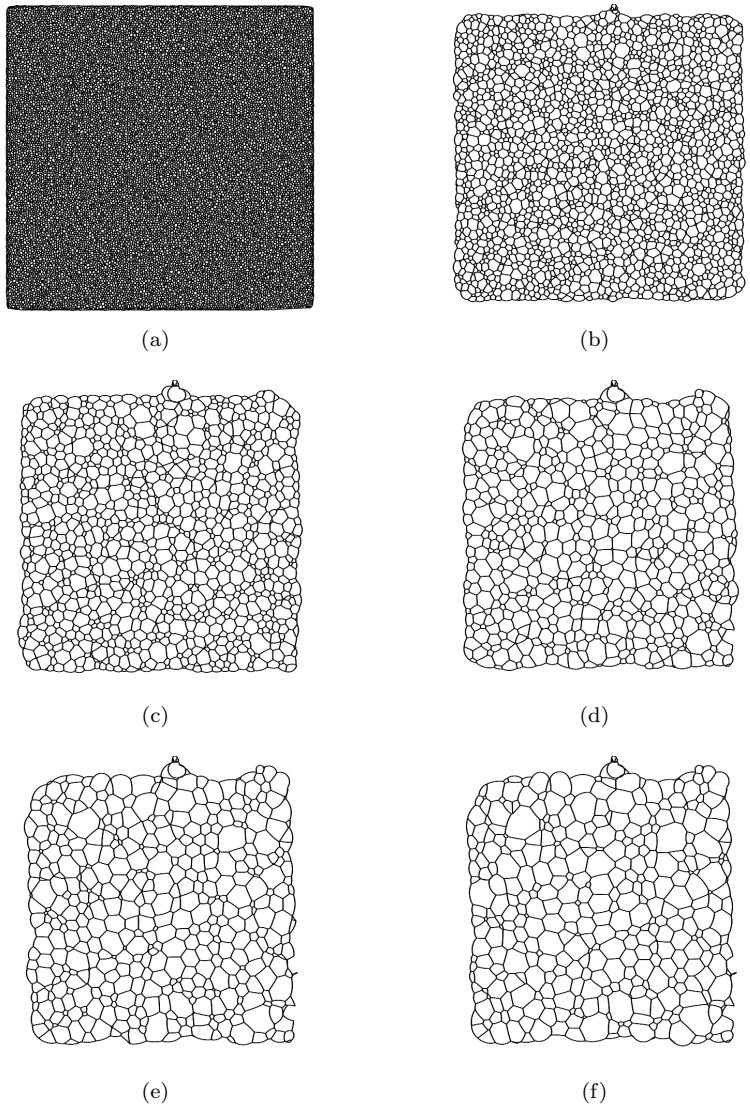


Figure 6.3: 20.906 cells simulated evolving over 60.000 frames. Frame 1, 12.001, 24.001, 36.001, 48.001, and 60.001 are shown.

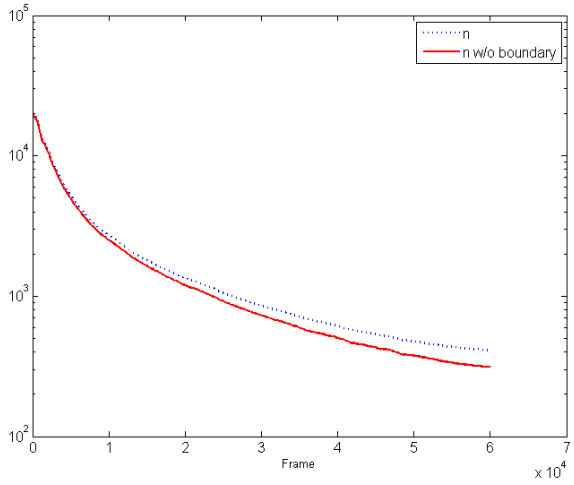


Figure 6.4: Cell count. Note logarithmic scale.

that form the free boundary. Statistics without the boundary is the bulk of the foam and is as close to real-world measurements on foams as we can come [22]. Trivially we expect the total number of cells to decrease monotonically, since cells can not be created, only collapse, which is also what we observe. Right after the foam has been created, it is highly disordered, with many small cells close to the T2 limit. Therefore we observe that the number of cells decrease rapidly inside the first 5.000 frames as the foam rearranges itself to a more stable shape. After this initial periode the number of T2 operations decrease and we get a super-linear development of the total number of cells. Note the growing difference between cell count with and without boundary. We expect the foam to become increasingly dominated by the boundary as the number of cells decrease. The observed difference is bulk cells coming into contact with the boundary and disappearing from the bulk measurement as boundary cells collapse.

Based on statistics gathered from observations of real foam samples, we expect the distribution of cell valency (number of films in a cell) to be roughly Gaussian with peak at $n = 6$. This distribution is the basis of foam statistics, as defined in Section 2.6. Figure 6.5 shows the development in cell distribution over the course of the simulation. Focusing our attention first on the bulk foam without boundary, we see approximately the expected distribution, though there is a pronounced skew towards $n = 5$, especially in (c) and (d). Examining the second moment μ_2 of $p(n)$ in Figure 6.6 (b) is more telling. After the initial 5.000 frames $p(n)$ stabilizes with $\mu_2 \approx 1.3$ and increasing. Between frames 20.000 and 30.000 we see a sudden jump after which, though very noisy, $\mu_2 \approx 1.6$. This value is slightly above what we would expect from real-world experiments. Stavans and Glazier [22] found a value of $\mu_2 \approx 1.4$. It is difficult to say where this difference stems from, but it is possible that there is some influence from the free boundary, as the boundary becomes more dominant in the foam as the number of cells decrease. The experiments performed by Stavans and Glazier used what amounts to a fixed boundary condition and, as foam is a space-

filling structure [28] that will stretch to fill available space, their foam was less free in it's movements. Therefore it is conceivable that, even though they also considered only the foam bulk, the different boundary conditions can cause a change in the behavior of the foam and thus in the measured cell distribution.

The observed behavior of μ_2 is in aggrement with the notion of a *scaling regime* as hypothesised in [22], where after an initial chaotic rearrangment of the foam, the foam reaches a stable state and is only affected by cell growth due to diffusion. If this is indeed the case then it would seem that we reach the scaling regime after approximately 23.000 frames or 13 minutes of simulated time. Correlating back to Figure 6.4 we observe that there is indeed a slight change in the rate of change of total number of cells close after frame 20.000 where the decrease becomes slightly more linear.

Turning our attention to the foam with boundary we see a slightly different picture. We expect the average distribution of boundary cells to be closer to five than six, since the boundary "removes" one junction from the boundary cells. In other words, if we add another layer of cells to the foam, such that the boundary cells become bulk cells, each cell will, on average, gain one junction and thus one film. As a result we expect $p(n)$ to be skewed towards five when including the boundary, but most pronounced for smaller numbers of cells when the boundary becomes dominating. Examining Figure 6.6 (b) we see that this is indeed the case. Over the evolution of the foam, the difference between $p(n)$ with and without boundary grows, but $p(n)$ with boundary is consistently larger than $p(n)$ without boundary for $n < 5$ and until (d) also for $n < 6$. As the boundary becomes dominating we see that the discrepancy between $p(n)$ with and without boundary increases. Looking at the second moment in Figure 6.6 we see that at first the two measurements are close but as the number of cells decreases and the boundary becomes dominating, the discrepancy becomes pronounced. In Figure 6.6 (b) we see that $\mu_2 \approx 2$ towards the end of the sequence as the number of four- and five-sided cells increase. It is important to note, though, that the boundary-including measurements are unreliable as the anomalies mentioned earlier will inevitably affect the statistics.

It is interesting to note that there are quite a large number of two-sided cells in the foam. From frame ∼36.000 and onwards there are in fact more two-sided cells than there are cells with $n \geq 10$. This suggests that two-sided cells are an important part of the simulation and that the decision to devote time to them was not misplaced.

From 2.6 we recall that the Aboav-Weaire law

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

relates an n-sided cell to the average number of sides in neighboring cells. Stavans and Glazier [22] showed an experimentally verified relation with $a = 1$ and $\mu_2 = 1.4$. Figure 6.7 shows the development in $nm(n)$ during the simulation, plotted against the idealised $nm(n) = n\left(5 - \frac{7.4}{n}\right)$ suggested by Stavans and Glazier. Through-out the simulation we see a very close match between the measured values and the ideal. In the first part of the sequence the measured points are slightly below the ideal line, while later, when we enter the scaling regime, the points are slightly above the line. This matches our earlier result where we found $\mu_2 \approx 1.6$ where Stavans and Glazier found $\mu_2 \approx 1.4$.

Let us now consider some of the computational aspects of the simulation.

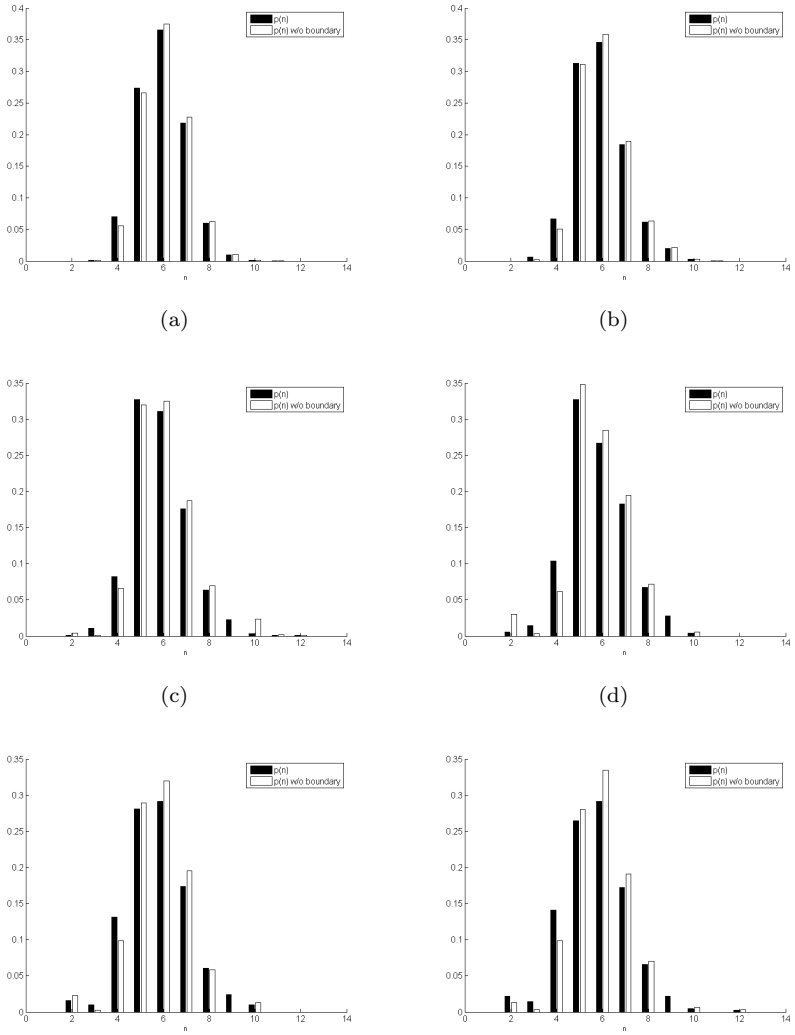
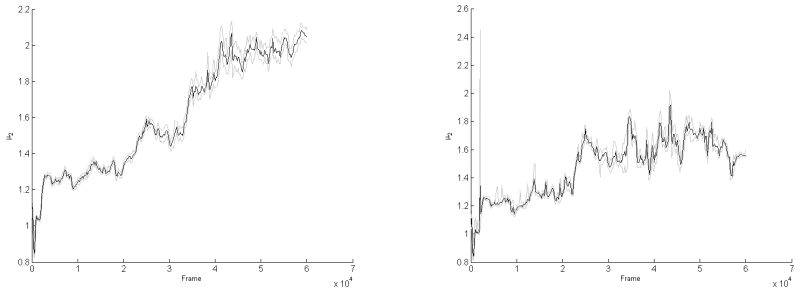


Figure 6.5: The distribution $p(n)$ of cells with relation to their valency at frame 1, 12.001, 24.001, 36.001, 48.001, and 60.001.



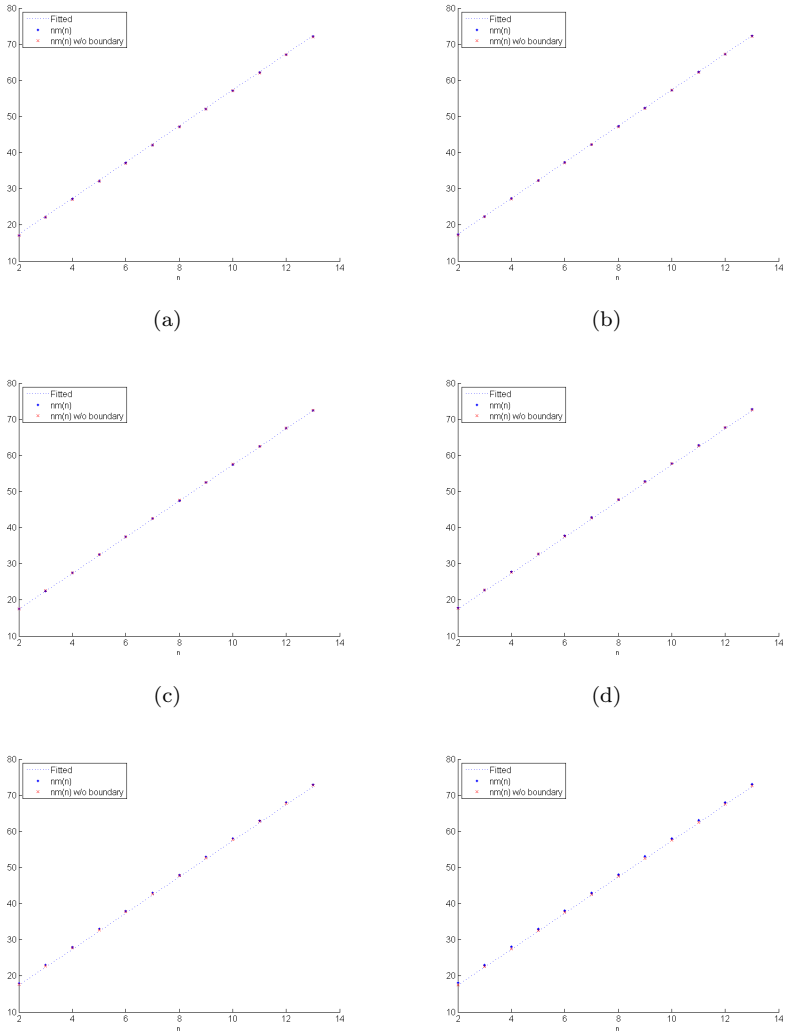
(a) (b)

Figure 6.6: μ_2 measured (a) with boundary and (b) without boundary per frame. Each point shows min, max, and average value for the preceding 200 frames.

Figure 6.8 shows the number of quasi-static iterations per frame averaged over 200 frames. We see a sharp decline in number of iterations until we reach the scaling regime, after which the rate levels out and becomes almost linear. This matches our expectation that a large number of iterations are necessary in the first, chaotic regime but as we enter the more stable scaling regime less iterations are necessary. The number of iterations necessary fluxtuates considerable, as can be seen from the relatively high standard deviation. This is mostely due to frames with many T1 and T2 operations needing more iterations. Figures 6.9 and 6.10 shows total number of T1 and T2 operations respectively per 200 frames. T2 operations are especially interesting as we see a large number of T2 operations in the early simulation, but a rapid decrease until T2 operations becomes rare occurances in the scaling regime. The number of T1 operations are less clear, but we can observe that the number of T1 operations seems to fall until we enter the scaling regime after which we see a sharp increase in T1 operations again. However, as we can see from the standard deviation, the number of T1 operations fluctuates wildly between frames.

To examine how much work we are doing per cell, Figure 6.11 show how many iterations per cell (IpC) was done. Here we see an increase in IpC until frame 35.000 after which the IpC seems to be roughly constant. The IpC is interesting as it shows that, though we are doing vastly more iterations in the early frames we are doing more work per cell in the later frames. This seems to suggest that, although the scaling regime looks stable, the quasi-static simulation is finding it harder to equilibrate the foam. This is not unreasonable, as even a small change to a near-stable foam can cause a neighborhood to become less stable. That neighborhood will then need to be relaxed again, as illustrated in Figure 6.12. This is unfortunately a consequence of only equilibrating locally. Another possible explanation is that we, in general, need more iterations to equilibrate free boundary cells. As the number of bulk cell decreases and the boundary becomes dominating, the necessity for performing more iterations on the boundary becomes more pronounced.

Figure 6.13 shows the wall-clock time of the simulation. The reader should not place to much importance on the actual values, as the implementation is not optimized. Figure 6.14 shows the number of iterations per second (IpS). Here we see a sharp increase in IpS until we enter the scaling regime, after which the

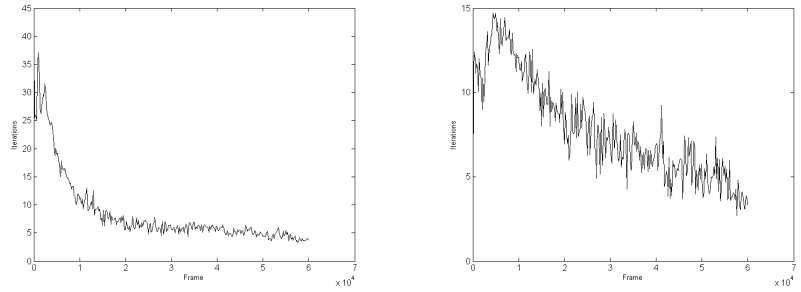


(a) (b)

(c) (d)

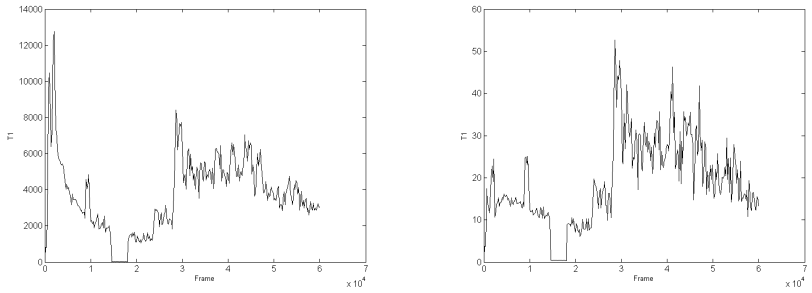
(e) (f)

Figure 6.7: The Aboav-Weaire relation $m(n) = 6 - a + \frac{6a+\mu_2}{n}$ at frame 1, 12.001, 24.001, 36.001, 48.001, and 60.001. The dotted line is an ideal $nm(n)$ with $\mu_2 = 1.4$ and $a = 1$.



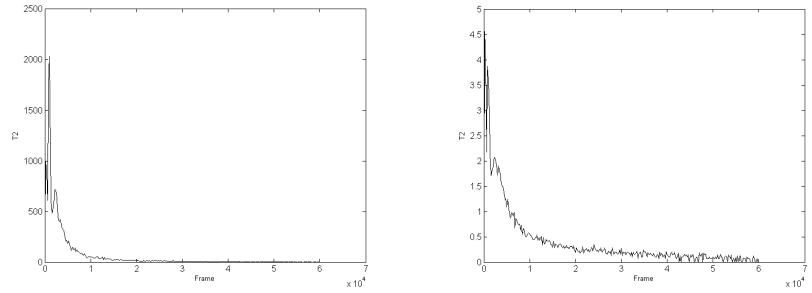
(a) (b)

Figure 6.8: Number of iterations per frame. (b) Standard deviation



(a) (b)

Figure 6.9: Total number of T1 operations per frame. (b) Standard deviation



(a) (b)

Figure 6.10: Total number of T2 operations per frame. (b) standard deviation

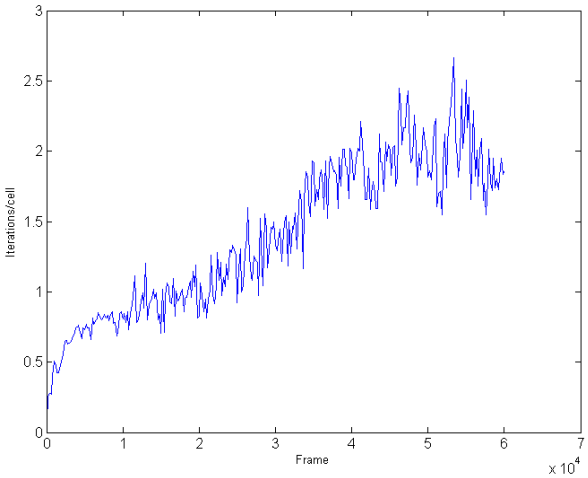


Figure 6.11: The number of iterations per cell per frame.

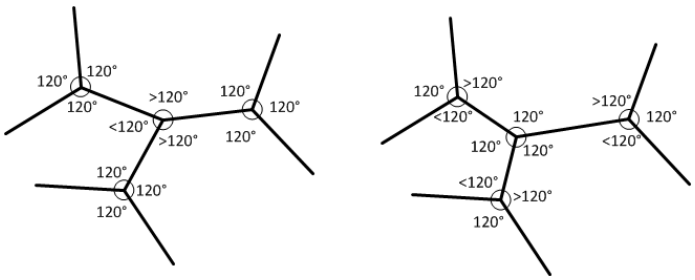
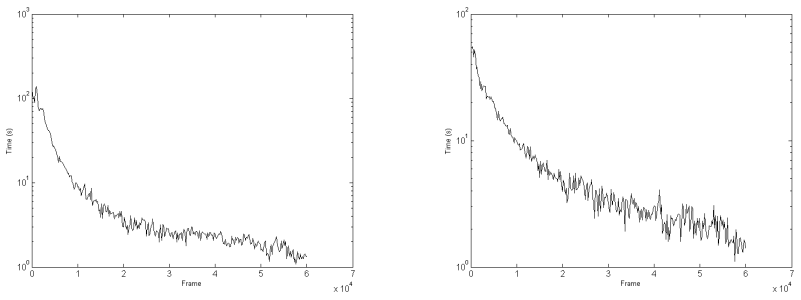


Figure 6.12: A local relaxation of a junction causes a neighborhood to become slightly less equilibrated.



(a) (b)

Figure 6.13: The wall-clock time in seconds per frame. Note the logarithmic scale. (b) standard deviation.

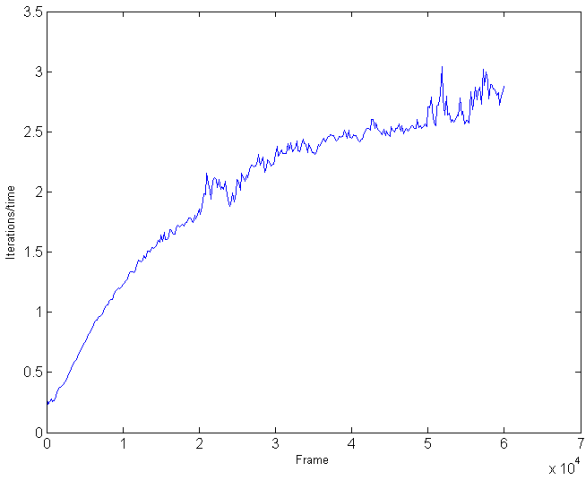


Figure 6.14: The number of iterations per second per frame.

increase becomes roughly linear. The IpS mimicks the number of iterations in this. This shows that, even though we saw an increasing number of iterations per cell, the decreasing total number of cells means that the time per frame does not increase.

6.1.3 Development with the Orientation Invariant Constraint

In a further experiment, another foam with initially 20.140 cells was allowed to coarsen over a periode of 50.000 frames of $\Delta t = \frac{1}{30}$s after which there was 515 cells left. The same constants as in the previous experiment was used, but the foam was influenced only by the Orientation Invariant Constraint. Figure 6.15 shows frames 1, 9.961, 19.921, 30.047, 40.007, and 49.967. The foam quickly develops anomalies on the boundary and after frame 30.000 also in the bulk of the foam. I will return to these anomalies later. Figure 6.16 shows the

development in cell count. It seems the foam enters the scaling regime after ∼25.000 frames.

Even with anomalies we see a reasonably Gaussian distribution of cells over the sequence, as shown in figure 6.17. This is also reflected in the second moment, Figure 6.18, both with and without boundary. Though the second moment without boundary have some extreme outliers the general tendency is that μ_2 lies between 1 and 2. The Aboav-Weaire relation, plotted against the ideal $nm(n) = n\left(\frac{5-7.4}{n}\right)$ in Figure 6.19, again shows a good relation between the theory and the simulation.

This experiment clearly shows that the Orientation Invariant Constraint in itself is not enough to keep the simulation from developing invalid junction configurations. However, the statistics suggests, that as long as the anomalies are influences a relatively limited area, as is the case here, the essential statistics are not significantly affected.

6.1.4 Further experiments

In addition to the experiments just described, two other experiments was conducted, one with a foam of 273 cells and one with a foam of 2.286 cells. Statistics similar to those presented for the 20.906 cell foam can be found in Appendices A.1, A.2, and A.3 respectively. Note that the same foam scale and constants where used in all three experiments. Only the initial computational mesh was changed. Specifically, they where constructed from Delaunay triangulations of 200 and 2.000 vertices.

Examining the 273-cell foam (Appendix A.1) we see a foam that is dominated by the boundary from the start. There is a close to linear progression in cell count, but the distribution of cells quickly move away from the expected Gaussian distribution which is also clear from the second moment of $p(n)$. It is difficult to see whether there is a scaling regime in this small foam. From this it must be concluded that from the view-point of creating a physically correct simulation, the Vertex Model with free surface boundary is not a good choice for small foam samples. The greater the influence of the boundary in relation to the foam bulk, the further from the physical measurements we come. Performance-wise, not surprisingly we need significantly fewer iterations per frame, which also reflects in the wall-clock time per frame, which even drops below the resolution of the timer used after approximately 1.000 frames. IpS is, even in this unoptimized implementation, on the order of 10^6 which makes real-time simulation of small foams feasible.

The 2.286 cell foam shows better physical behavior. The simulation was run both with the Energy Decay Constraint (EDC) (Appendix A.2) and with the Orientation Invariant Constraint (OIC) (Appendix A.3). Looking first at the sequence with EDC, we see the same behavior as the 20.000 cell foam when examining the total cell count per frame, with a break at approximately frame 10.000 that suggests that we enter the scaling regime there. The foam have a good disribution of cells, though the second moment μ_2 is highly unstable when disregarding the boundary. The boundary shows a distribution centered closer to five than six, which matches our expectation. These statistics are backed up by the Aboav-Weaire relation which shows that the foam bulk is very close to the ideal though-out the sequence. This clearly shows that the increased number of cells over the 273 cell foam diminishes the negative effects from the

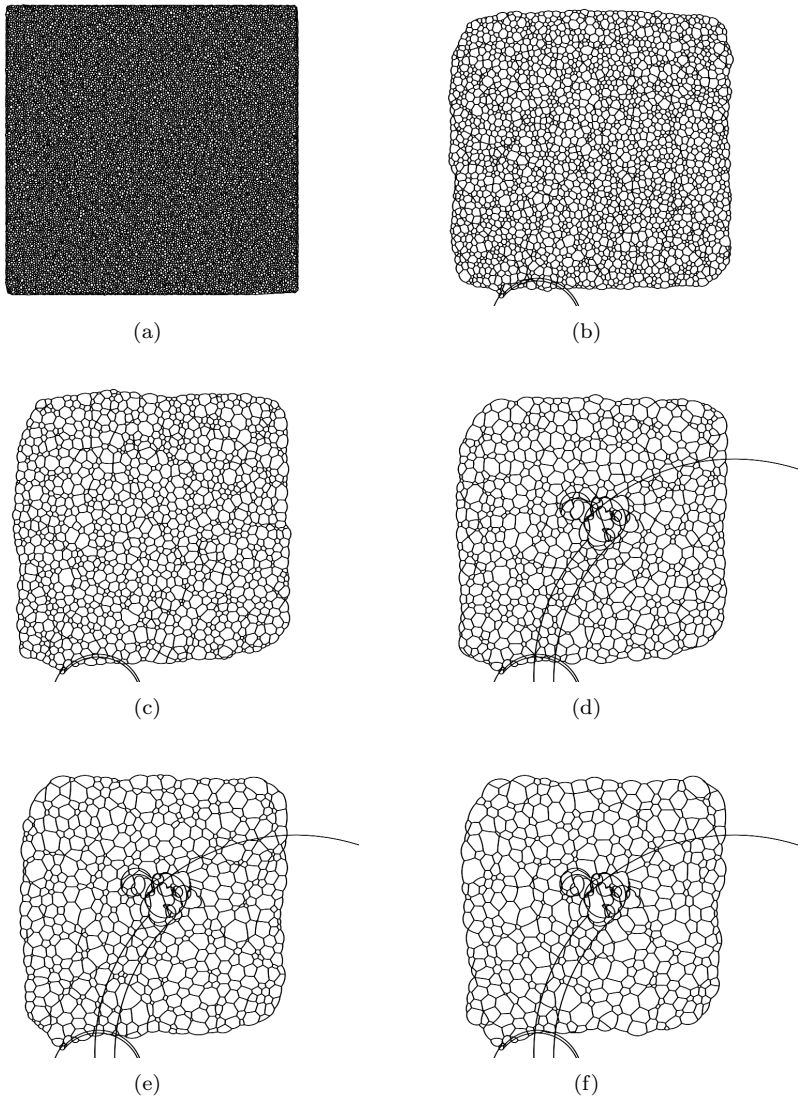


Figure 6.15: 20.906 cells simulated evolving over 60.000 frames. Frames 1, 9.961, 19.921, 30.047, 40.007, and 49.967 are shown.

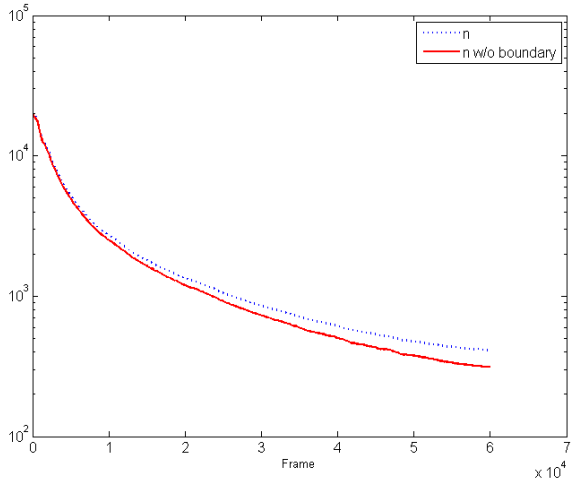


Figure 6.16: Development in cell count in a foam developed with the Orientation Invariant Constraint. Note logarithmic scale.

boundary. Over the sequence, the number of iterations per frame quickly drops, while the IpC climbes steadily, further supporting that the free surface boundary cells needs a larger number of iterations to be equilibrated. Note the short-lived anomaly in the lower-right corner of frame 6.001. The simulation was, in this sequence, able to recover and the anomaly completely disappeared.

With the OIC, we see similar statistics, but without any anomalies forming. The second moment of $p(n)$, both with and without bondary, is slightly more stable which suggests that the Energy Decay Constraint have a hard impact on the simulation.

6.2 Foam equilibrium

To investigate the equilibrium state of a foam, I created a foam sample of 2.272 cells and ran 1.000 frames without diffusion to ensure that the foam had reached a wholly stable state. After the first frame there was 2.239 cells left and this remained constant throughout the sequence. Figure 6.20 (a) shows the foam sample, while (b) shows the cell distribution $p(n)$. When disregarding the boundary we get a perfect Gaussian distribution, while $p(n)$ with boundary has the expected skew towards the lower values. The second moment $\mu_2 \approx 1.2$ with and $\mu_2 \approx 1.1$ without boundary, confirms this. The Aboav-Weaire relation $nm(n)$, shown in (c), is very close to, but slightly below, the ideal value, which is not surprising, as the ideal $\mu_2 = 1.4$ is for a foam in the scaling regime, which this static sample never enters.

6.3 Implementation performance

I have intentionally not made any formal profiling to measure the performance of the implementation. It is a proof-of-concept implementation and the focus of

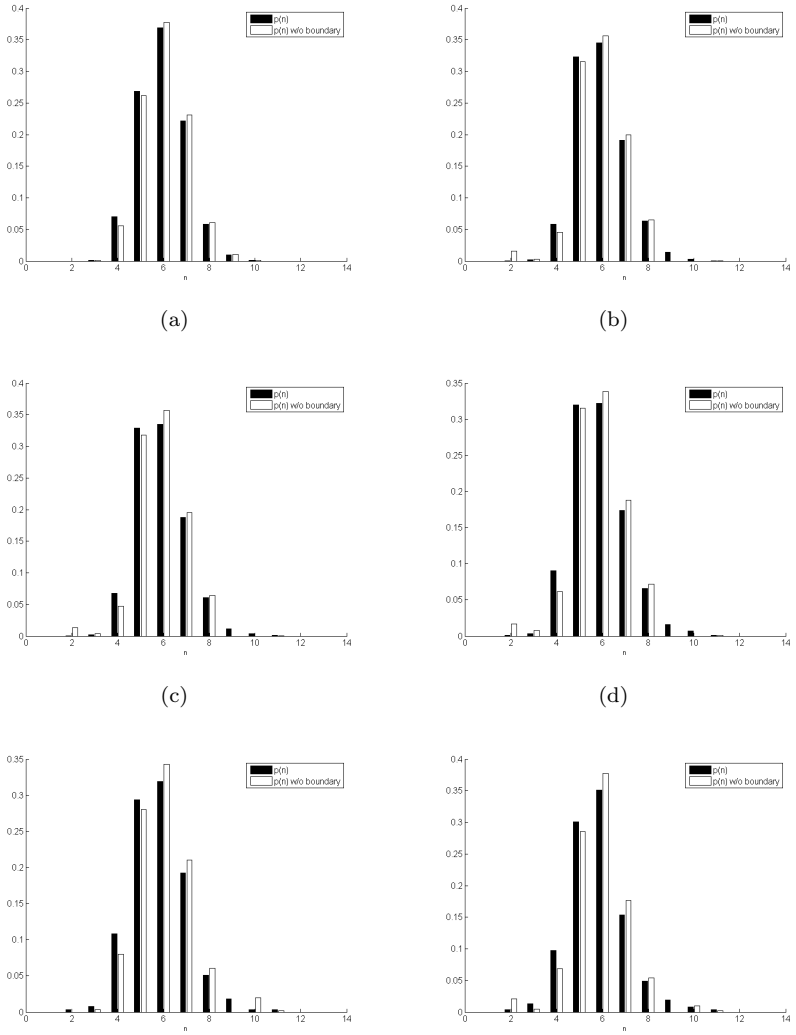
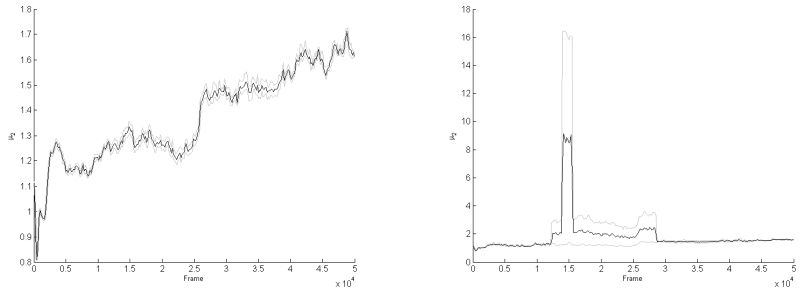


Figure 6.17: The distribution $p(n)$ of cells with relation to their valency at frames 1, 9.961, 19.921, 30.047, 40.007, and 49.967.



(a) (b)

Figure 6.18: μ_2 measured (a) with boundary and (b) without boundary per frame. Each point shows min, max, and average value for the preceding 200 frames.

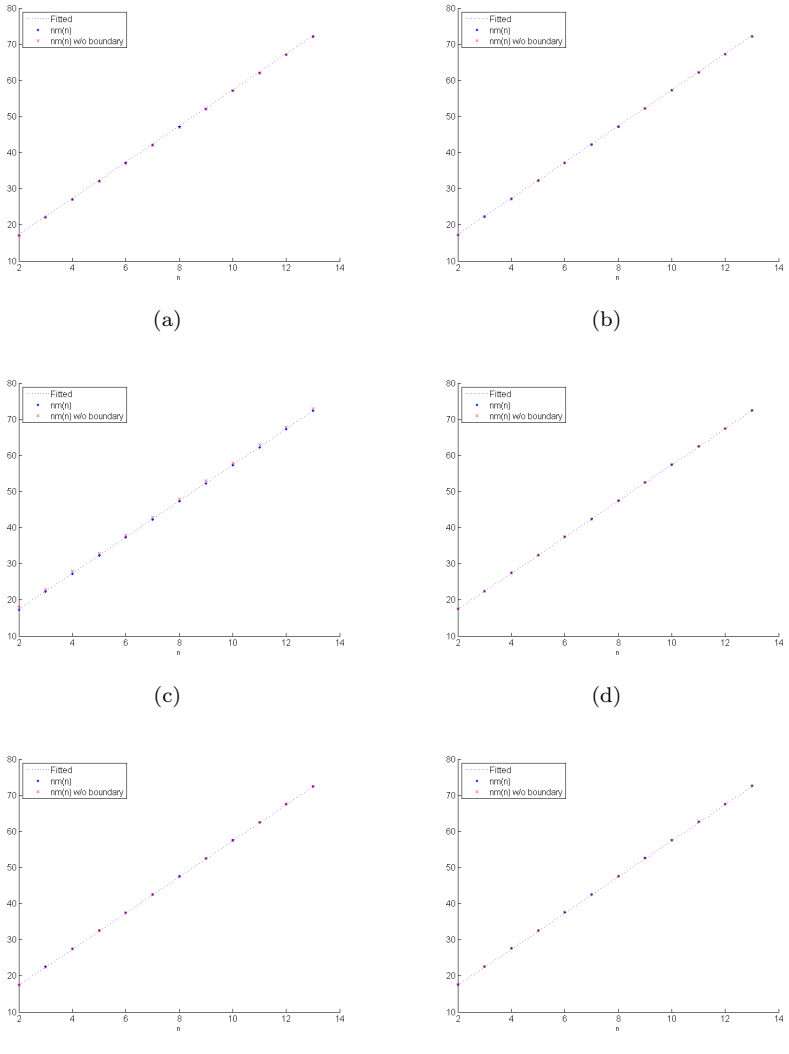
this thesis has been on the model and the correctness of the simulation. However, in an informal test, 1.000 frames of the quasi-static simulation with coarsening was run on two identical foams of $\sim$ 2.000 cells, one with the Jacobi method implemented to run purely sequential on the CPU and one with the CUDA implementation. The CPU implementation took 6 hours and 23 minutes on an Intel Core2 Quad 2.4 GHz CPU, while the CUDA version took 1 hour and 12 minutes on a NVidia Geforce GTS 250 attached to the same CPU. Even with the naive CUDA implementation we see a five-fold speed up over the sequential implementation.

In another experiment, a foam was allowed to coarsen over 300 frames. A profiling showed that 92% of the run time on the CUDA device was spend in the sequential junction update, 7.7% was spend calculating and solving the $\nabla f_j \mathbf{z} = -f_j$ systems and less than 0.3% was spend copying data between the CPU and the CUDA device. Note that this only takes into account the GPU run time. The topological operations, which are run sequentially on the CPU, are not included. In the future, some time should be dedicated to creating an optimized implementation. We can however with reasonably certainty conclude that the current implementation is bound by the non-parallel parts and in particular by the topological processes.

6.4 Discussion

The aspect of the simulation that is most stricking is undoubtedly that the simulation is not unconditionally stable and that anomalies are formed. The anomalies are caused by the simulator being unable to correctly equilibrate a junction (neighborhood) so instead of moving towards a an equilibrated state the simulation moves towards a less equilibrated state. A similar behavior was observed in previous implementations [13, 24]. Even though the simulation is still not unconditionally stable, the implementation presented in this thesis is markedly more stable than the previous implementations of the Vertex Model. This claim is based in no small part on the fact that Kelagers implementation could become unstable to such a degree that it became divergent. The presented implementation has in numerous tests not once become divergent.

Of the two presented constraints, the Orientation Invariant Constraint (OIC)

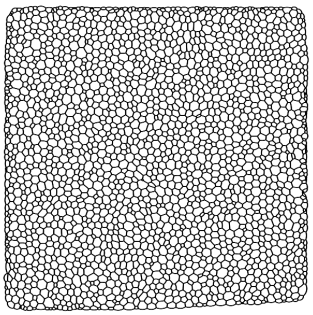


(a) (b)

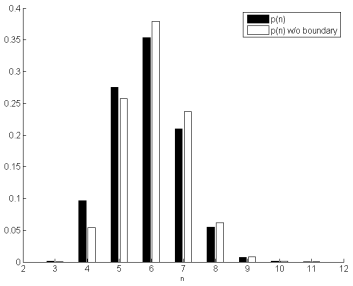
(c) (d)

(e) (f)

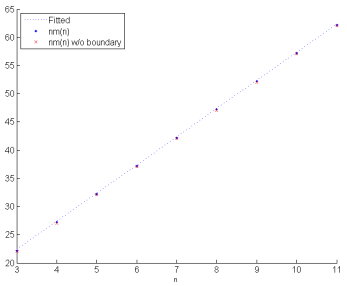
Figure 6.19: The Aboav-Weaire relation $m(n) = 6 - a + \frac{6a+\mu_2}{n}$ at frame 1, 9.961, 19.921, 30.047, 40.007, and 49.967. The dotted line is an ideal $nm(n)$ with $\mu_2 = 1.4$ and $a = 1$.



(a)



(b)



(c)

Figure 6.20: Statistics for a 2.239 cell foam in equilibrium. (a) The foam sample, (b) cell valency distribution $p(n)$, (b) Aboav-Weaire relation $nm(n)$.

is to weak to prevent all undesirable foam configurations, while the Energy Decay Constraint (EDC) is to strong and limits the equilibration process. To understand why the EDC fails, let us consider the definition of the constraint: Junctions are only allowed to move in directions that keeps the energy constant, or lowers the energy. We know from Section 2.5 that the point of minimum energy is well-defined and inside the triangle spanned by the three incident junctions. In the extreme, where the arc radius of the three films $r \to \infty$, the energy is minimized when the turning angles between the three films are all 120°. In other words, the EDC is a restatement of Plateau's law, but we have promoted it from a soft constraint of equal strength to the area constraint, to a hard constraint of greater strength. This destroys the natural interplay between the two constraints and the result is the anomalies we are observing: The area constraint are trying to move the junction in one direction but the EDC disallows that movement and the junction remains rigidly fixed. This also explains why we only see obvious anomalies on the boundary: In the bulk of the foam, junctions are close to equilibrium and generally inside the triangle spanned by the three incident junctions. Therefore the movement constraint interfere less with the junctions slight movements.

The OIC is more successful in fulfulling it's role. It effectively stops inverted junctions from forming, while still allowing movement of problem junctions. However the OIC is not strong enough to prevent all invalid foam configurations: It stops inverted junctions from forming, but it does not stop junctions from making large jumps. Without the EDC, the foam is highly sensitive to the initial mesh, as is evident from the 20.000 cell experiment where anomalies where formed instantly if the EDC was not used.

While it is somewhat well-understood why anomalies can form when the EDC is used, it is much less clear why anomalies forms when the EDC is not used. The OIC in itself is unlikely to cause these anomalies, as its impact is small: There was no indication of near-inverted junctions near the anomalies formed in the bulk of the foam. One possible cause is the calculation of ∇f_j in the equilibration process: If ∇f_j is near-singular, the $\nabla f_j \mathbf{z} = -f_j$ system becomes ill-conditioned and it is possible that this is what we are observing.

Disregarding the boundary, the simulation displays a good correlation to the statistics measured from real foam samples by Stavans and Glazier. From this we can conclude that the underlaying theoretical model is sound. It is harder to be conclusive about the free surface boundary as we do not have measurements on a real foam to compare against and as such our arguments must necessarily be more subjective. But taking this and the anomalies into account we observe the expected behavior from the free surface: The internal pressure in the foam is greater than the surrounding environment, causing the free surface to be convex. We also see a slightly lower number of films in boundary cells which is also expected. However, we can observe that most anomalies forms on the free boundary and from this we must conclude that the boundary is problematic. The foam is highly sensitive on the boundary, suggesting that boundary junctions are, in general, badly conditioned. This further suggests that the Vertex Model, as it has been stated here, is not sufficiently robust to handle the free boundary. From a simulation point of view it may be necessary to reintroduce periodic boundaries if a completely stable simulation is required. This will also partly remove the problems observed with the EDC, as it is better suited to the conditions found in the bulk of the foam.

The dual computational mesh is a definite success. Not only does it simplify most aspects of the simulation, it also enables the addition of two-sided cells to the model. The decoupling of topology and geometry in the computational mesh gives us greater flexibility and allows for a more efficient data structure when compared to the one used by Kelager.

The topological operations are, on the whole, a weak point in the Vertex Model: As opposed to other parts of the model, the topological operations are only loosely defined with no rigid underlying physical theory. While we have a well-defined model for how the topological operations will change the computational mesh, there is very little understanding of the implications inherent to when and where they should be performed, not least when *not* to perform topological changes. I have attempted to put focus on this issue, but we are still far from a solid model. The operations are also difficult to parallelize, making them a bottleneck in the execution.

Chapter 7

Conclusion

In this thesis I have presented a comprehensive explanation of the Vertex Model and have demonstrated that it produces foams with statistics comparable to measurements of real-world foams. While doing so I have introduced several improvements and additions to the model, not least the dual computational mesh.

As has been demonstrated, the Vertex Model is still not completely stable. It is at this point unclear whether the problems are due to some minor detail of the model, or if the model is fundamentally flawed. What is clear is that currently some additional constraints are needed to keep simulations from becoming (locally) unstable. I have presented two such constraints, but have also shown that one is too strong (the Energy Decay Constraint) while the other is too weak (the Orientation Invariant Constraint). What is needed is a constraint that falls between these two in strength. Below I will briefly outline a new method for relaxing foams which may prove helpful.

The parallel CUDA implementation presented here, while functional, is far from optimized. This presents a practical problem of simulating large foam samples. Many parts of the implementation is still strictly sequential and making more of these parts parallel will present interesting challenges in the future.

7.1 Future work

There are several aspects of the Vertex Model that are still open to improvements. In the following I will discuss some of the larger issues.

7.1.1 Improved Newton root search problem

The two constraints discussed leads me to suggest a fundamental change to the function being relaxed in the Newton root search. Let me first summarize what we have learned about the relaxation process:

- The film turning angle calculation is insufficient and unstable.
- If a junction becomes inverted, the relaxation process can not in itself bring the junction back into a un-inverted state.
- We do not wish for the surface energy of a junction to increase.

- The surface energy of a junction is minimized when the film turning angles are 120°.

This summary provides both the problem and the solution: If we can remove the film turning angle from the relaxation process we no longer need the two constraints. To see how this is possible, consider the root search problem in the relaxation process:

$$\begin{bmatrix} \frac{\partial A_i}{\partial p_i} & \frac{\partial A_i}{\partial p_j} & \frac{\partial A_i}{\partial p_k} & \frac{\partial A_i}{\partial x} & \frac{\partial A_i}{\partial y} \\ \frac{\partial A_j}{\partial p_i} & \frac{\partial A_j}{\partial p_j} & \frac{\partial A_j}{\partial p_k} & \frac{\partial A_j}{\partial x} & \frac{\partial A_j}{\partial y} \\ \frac{\partial A_k}{\partial p_i} & \frac{\partial A_k}{\partial p_j} & \frac{\partial A_k}{\partial p_k} & \frac{\partial A_k}{\partial x} & \frac{\partial A_k}{\partial y} \\ \frac{\partial \theta_i}{\partial p_i} & \frac{\partial \theta_i}{\partial p_j} & \frac{\partial \theta_i}{\partial p_k} & \frac{\partial \theta_i}{\partial x} & \frac{\partial \theta_i}{\partial y} \\ \frac{\partial \theta_j}{\partial p_i} & \frac{\partial \theta_j}{\partial p_j} & \frac{\partial \theta_j}{\partial p_k} & \frac{\partial \theta_j}{\partial x} & \frac{\partial \theta_j}{\partial y} \\ \frac{\partial \theta_k}{\partial p_i} & \frac{\partial \theta_k}{\partial p_j} & \frac{\partial \theta_k}{\partial p_k} & \frac{\partial \theta_k}{\partial x} & \frac{\partial \theta_k}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta p_i \\ \Delta p_j \\ \Delta p_k \\ \Delta x \\ \Delta y \end{bmatrix} = - \begin{bmatrix} A_i^\tau - A_i \\ A_j^\tau - A_j \\ A_k^\tau - A_k \\ \frac{2\pi}{3} - \theta_i \\ \frac{2\pi}{3} - \theta_j \\ \frac{2\pi}{3} - \theta_k \end{bmatrix}.$$

As we can see, the last three rows of the Jacobian and the constraint vector depends on the film turning angle, but as we have learned, the film turning angle contraint is satisfied in the same location as the film surface energy is minimized. We can therefore theoretically change the root search problem to

$$\begin{bmatrix} \frac{\partial A_i}{\partial p_i} & \frac{\partial A_i}{\partial p_j} & \frac{\partial A_i}{\partial p_k} & \frac{\partial A_i}{\partial x} & \frac{\partial A_i}{\partial y} \\ \frac{\partial A_j}{\partial p_i} & \frac{\partial A_j}{\partial p_j} & \frac{\partial A_j}{\partial p_k} & \frac{\partial A_j}{\partial x} & \frac{\partial A_j}{\partial y} \\ \frac{\partial A_k}{\partial p_i} & \frac{\partial A_k}{\partial p_j} & \frac{\partial A_k}{\partial p_k} & \frac{\partial A_k}{\partial x} & \frac{\partial A_k}{\partial y} \\ \frac{\partial E_i}{\partial p_i} & \frac{\partial E_i}{\partial p_j} & \frac{\partial E_i}{\partial p_k} & \frac{\partial E_i}{\partial x} & \frac{\partial E_i}{\partial y} \\ \frac{\partial E_j}{\partial p_i} & \frac{\partial E_j}{\partial p_j} & \frac{\partial E_j}{\partial p_k} & \frac{\partial E_j}{\partial x} & \frac{\partial E_j}{\partial y} \\ \frac{\partial E_k}{\partial p_i} & \frac{\partial E_k}{\partial p_j} & \frac{\partial E_k}{\partial p_k} & \frac{\partial E_k}{\partial x} & \frac{\partial E_k}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta p_i \\ \Delta p_j \\ \Delta p_k \\ \Delta x \\ \Delta y \end{bmatrix} = - \begin{bmatrix} A_i^\tau - A_i \\ A_j^\tau - A_j \\ A_k^\tau - A_k \\ E_i^\tau - E_i \\ E_j^\tau - E_j \\ E_k^\tau - E_k \end{bmatrix},$$

where $E_{i,j,k}$ is the energy of film i, j, and k respectively and $E^\tau_{i,j,k}$ are target energies for the films.

The advantage of the updated system is that, instead of having the energy as a hard constraint with the area as a soft constraint, both the energy and area are soft constraints of equal importance. The updated system should also robustly handle inverted junctions: Since the film surface energy, unlike with film turning angle, is correctly calculated even for inverted junctions, the system should correctly "unravel" inverted junctions. Lastely, because the system attempts to minimize the film energy, and thus film length, large jumps in junction position away from the energy minima are discouraged.

There are two problems with this new approach: The first problem is that it is not clear what $E^\tau_{i,j,k}$ should be. It could either be $E^\tau_{i,j,k} = 0$, which would give us the global energy minimum, or it could be the minimum energy of the junction calculated with straight edges, which would give us the local energy minimum. We would need to conduct further tests to determine which of these options give the best result. The second problem is that the system is over-constrained. In the original system we can remove one row because we have a direct relationship between the three angles. It is clear that similarly there must be some relationship between the three film energies, but it is less clear what this relationship is.

In the future I intend to investigate this updated root search problem and determine whether it is a viable replacement and whether the proposed advantages and stability is visible in actual simulations or not.

7.1.2 Improved topological operations

The least physically founded part of the simulation is the topological operations. While the current method mechanically fulfills the required purpose of rearranging the topology of the foam, the method should be investigated further. A physical theory of the dynamics involved in the foam during the rearrangement which is translated into topological changes could only improve the simulation.

7.1.3 Further improvements

The ultimate goal of any foam simulation is, of course, to move into three dimensions. This presents several problems as films are transformed from lines into surfaces, most of them centered around how to pose the Newton root search problem and how to calculate the angles, areas, and volumes necessary. The improved root search problem suggested in Section 7.1.1 can prove to be helpful in this transition, as the surface energy is comparatively simpler to calculate than the angle between surfaces.

Another important milestone in the further development of our foam simulator is introducing true Plateau borders to create wet foam. This will add three new radii to junctions as the thickness of the Plateau border must be taken into account.

Going in another direction, adding external dynamics to the foam could increase the visual impact of the simulation. Especially multiple interacting foam samples, foam samples that can combine and split up, would be an large improvement both in the applicability and presentation of the simulator. As it is, the simulator, while technically proficient, lacks a certain “wow” factor for the general public. Towards this goal, adding visual aspects of soap films, such as light interference patterns would improve the marketability of the method.

7.2 Acknowledgement

I would like to thank my supervisor Kenny Erleben of the University of Copenhagen, Department of Computer Science, for his invaluable help in the writing of this thesis.

Bibliography

[1] Aste, T. Equilibrium and evolution of froths under topological constraints. *Philosophical Magazine Part B 71*, 5 (1995), 967–979.

[2] Barrales Mora, L. A. 2d vertex modeling for the simulation of grain growth and related phenomena. *Math. Comput. Simul. 80*, 7 (2010), 1411–1427.

[3] Brakke, K. A. The surface evolver. *Experimental Mathematics 1*, 2 (1992), 141–165.

[4] Bærentzen, J. A., Munk-Lund, S., Gjøl, M., and Larsen, B. D. Two methods for antialiased wireframe drawing with hidden line removal. *Proceedings of the Spring Conference in Computer Graphics* (2008).

[5] Cleary, P. W., Pyo, S. H., Prakash, M., and Koo, B. K. Bubbling and frothing liquids. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), ACM, p. 97.

[6] Desbrun, M., Kanso, E., and Tong, Y. Discrete differential forms for computational modeling. In *SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 courses* (New York, NY, USA, 2008), ACM, pp. 1–17.

[7] Elcott, S., and Schröder, P. Building your own dec at home. In *SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 courses* (New York, NY, USA, 2008), ACM, pp. 1–5.

[8] Erleben, K., Sporring, J., Henriksen, K., and Dohlman, K. *Physics-based Animation (Graphics Series)*. Charles River Media, Inc., Rockland, MA, USA, 2005.

[9] Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. *Computer graphics: principles and practice*, 2nd ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[10] Fuchizaki, K., Kusaba, T., and Kawasaki, K. Computer modeling of three-dimensional cellular pattern growth. *Philosophical Magazine B 71*, 3 (1995), 333–357.

[11] Icart, I., and Arqués, D. An approach to geometrical and optical simulation of soap froth. *Computers & Graphics 23* (1999), 405–418.

[12] Kawasaki, K., Nagai, T., and Nakashima, K. Vertex models for two-dimensional grain growth. *Philosophical Magazine Part B 60*, 3 (1989), 399–421.

[13] Kelager, M. Vertex-based simulation of dry foam. Master's thesis, Department of Computer Science, University of Copenhagen, September 2009.

[14] Kück, H., Vogelgsang, C., and Greiner, G. Simulation and rendering of liquid foams. In *Proceedings of Graphics Interface* (2002), pp. 81–88.

[15] Marsh, S. P., Masumura, R. A., and Pande, C. S. A curvature-driven vertex model for two-dimensional grain growth. *Philosophical Magazine Letters 72*, 6 (1995), 429–434.

[16] Microsoft Corporation. *Windows DirectX Graphics Documentation*, June 2010.

[17] Messer, R. *Linear Algebra: Gateway to Mathematics*. HarperCollins College Publishers, 1994.

[18] Neubert, L., and Schreckenberg, M. Numerical simulation of two-dimensional soap froth. *Physica A 240* (1997), 491–502.

[19] Nocedal, J., and Wright, S. J. *Numerical Optimization*, 2nd ed. Springer, 2006.

[20] NVIDIA. *NVIDIA CUDA^{TM}Programming Guide*, 2.3 ed., January 2009.

[21] NVIDIA. *NVIDIA CUDA^{TM}C Programming Guide*, 3.1 ed., May 2010.

[22] Stavans, J., and Glazier, J. A. Soap froth revisited: Dynamic scaling in the two-dimensional froth. *Physical Review Letters 62*, 11 (1989), 1318–1321.

[23] Ďurikovič, R. Animation of soap bubble dynamics, cluster formation and collision. *Eurographics 20*, 3 (2001).

[24] Vedel-Larsen, B. K. Foam: Constructing the dual mesh. Tech. rep., Department of Computer Science at University of Copenhagen, January 2010.

[25] Weaire, D., and Hutzler, S. *The Physics of Foams*. Oxford University Press, 1999.

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

[27] Weaire, D., and Kermode, J. P. Computer simulation of a two-dimensional soap froth ii. *Philosophical Magazine Part B 50*, 3 (1984), 379–395.

[28] Weaire, D., and Rivier, N. Soap, cells and statistics - random patterns in two dimensions. *Contemporary Physics 25*, 1 (1984), 59–99.

[29] Wejchert, J., Weaire, D., and Kermode, J. P. Monte carlo simulation of the evolution of a two-dimensional soap froth. *Philosophical Magazine B 53*, 1 (1986), 15–23.

[30] Weygand, D., Bréchet, Y., and Lépinoux, J. A vertex dynamics simulation of grain growth in two dimensions. *Philosophical Magazine B 78*, 4 (1998), 329–352.

[31] Weygand, D., Bréchet, Y., Lépinoux, J., and Gust, W. Three-dimensional grain growth: A vertex dynamics simulation. *Philosophical Magazine B 79*, 5 (1999), 703–716.

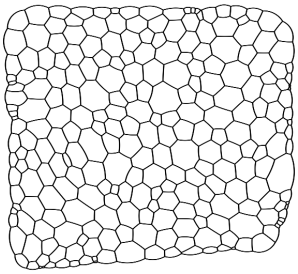
[32] Zheng, W., Yong, J.-H., and Paul, J.-C. Simulation of bubbles. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2006), Eurographics Association, pp. 325–333.

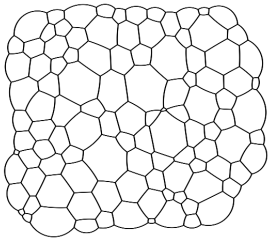
Appendix A

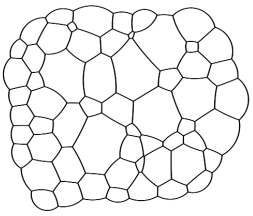
Results

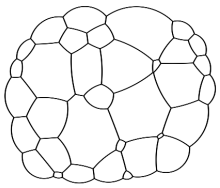
A.1 273 cell experiment - With Energy Decay Constraint

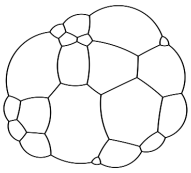
Frames 1, 1.365, 2.707, 4.071, 5.413, and 6.777 from the simulation.

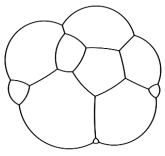




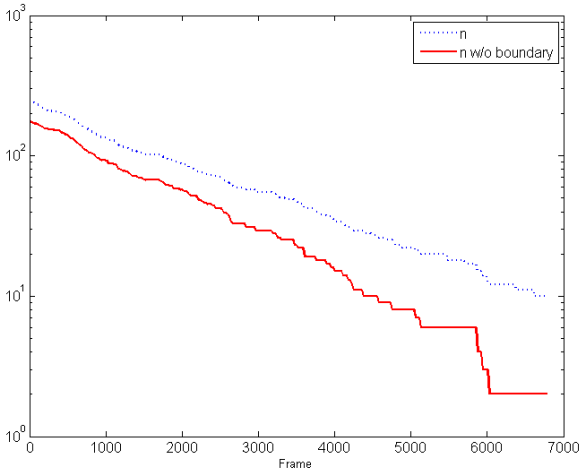




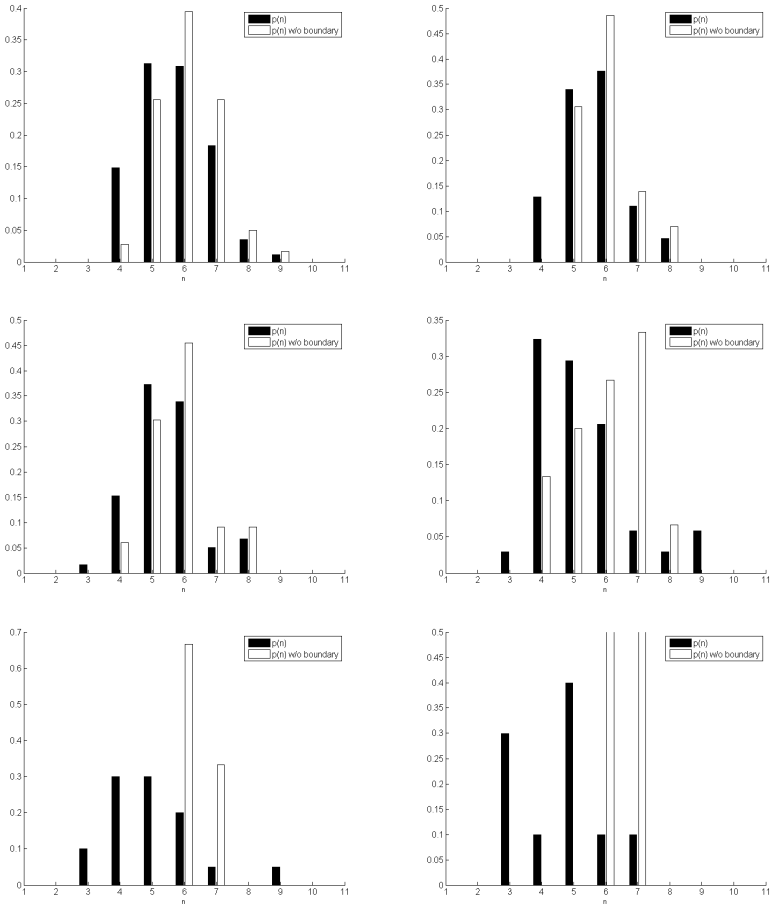




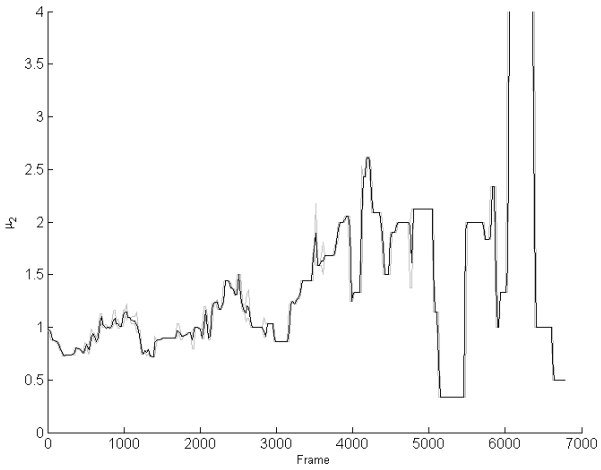
Total number of cells per frame.



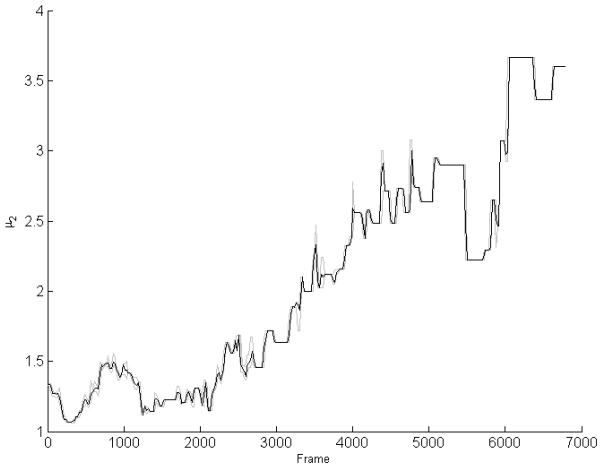
Cell valency distribution $p(n)$ for frames 1, 1.365, 2.707, 4.071, 5.413, and 6.777.



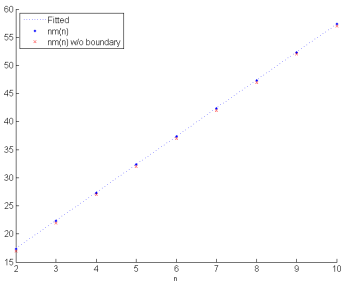
Second moment of $p(n)$ per frame without boundary.

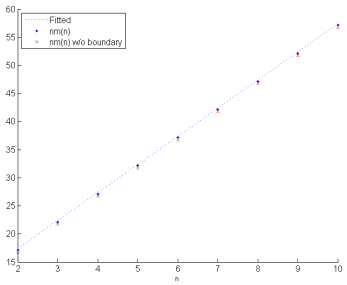


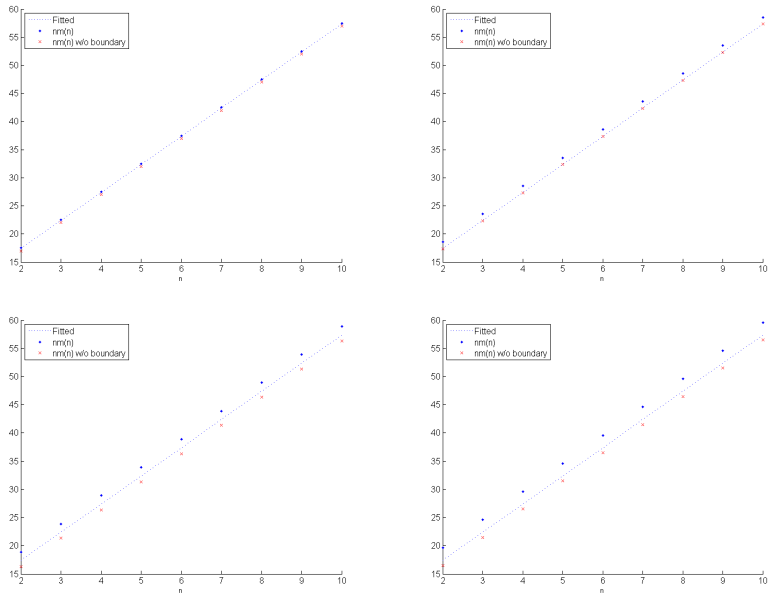
Second moment of $p(n)$ per frame with boundary.



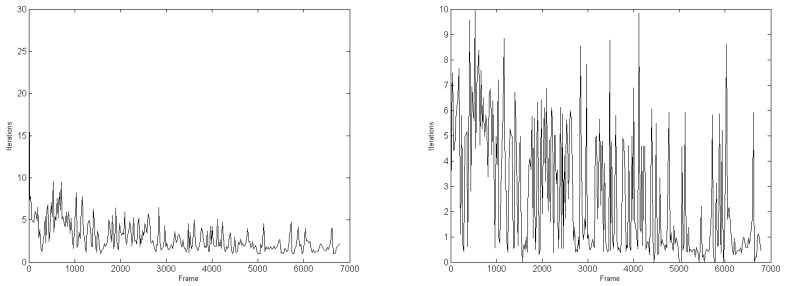
Aboav-Weaire relation $m(n)$ for frames 1, 1.365, 2.707, 4.071, 5.413, and 6.777.



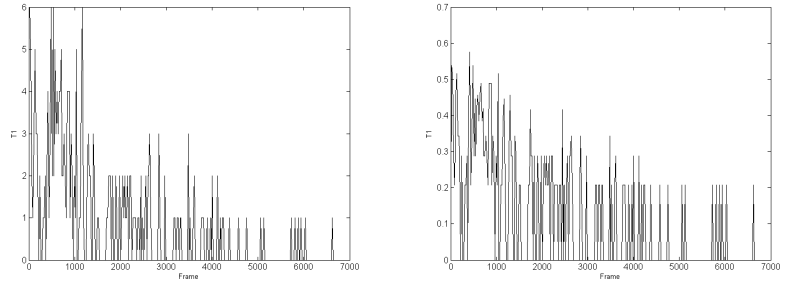




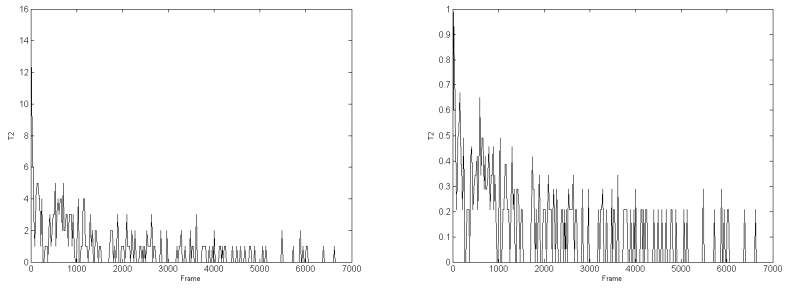
Number of iterations per frame and standard deviation.



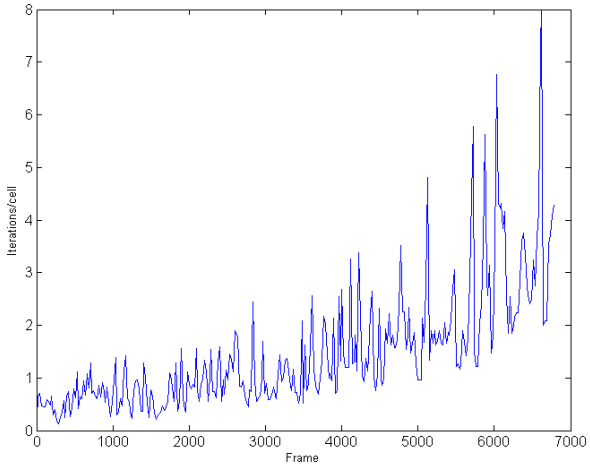
Total number of T1 operations and standard deviation.



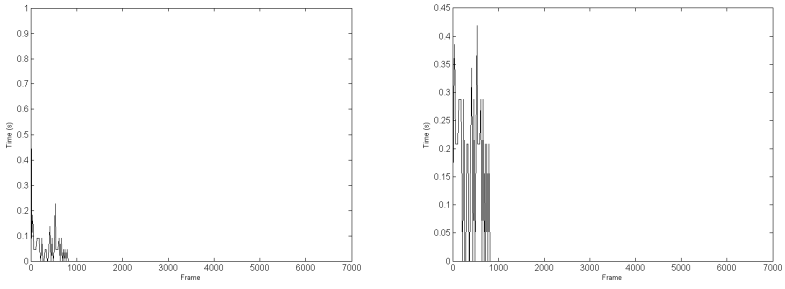
Total number of T2 operations and standard deviation and standard deviation.



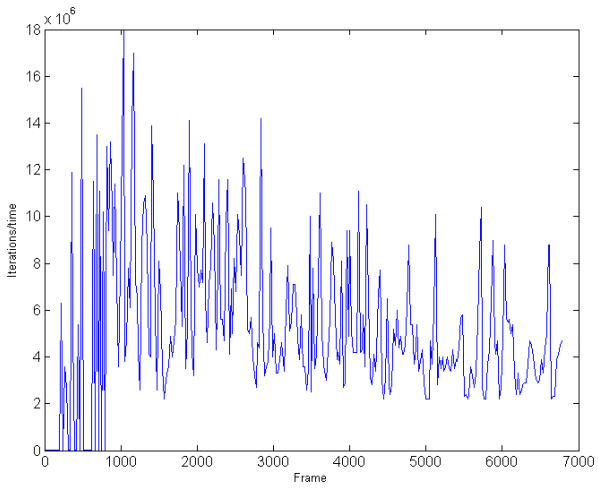
Number of iterations per cell per frame.



Wall-clock time in seconds per frame and standard deviation.

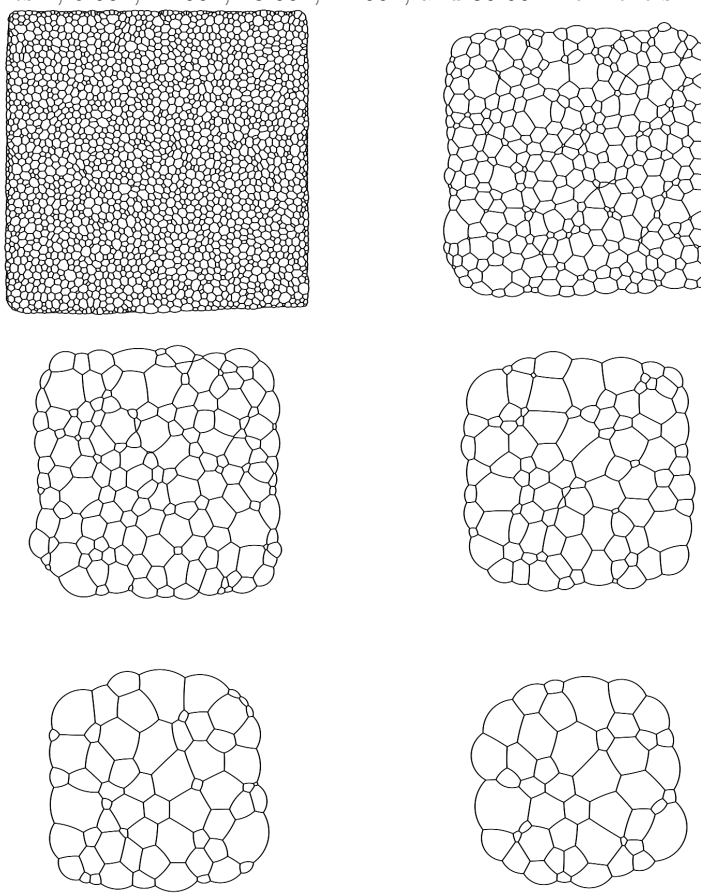


Wall-clock time in seconds per iteration per frame.

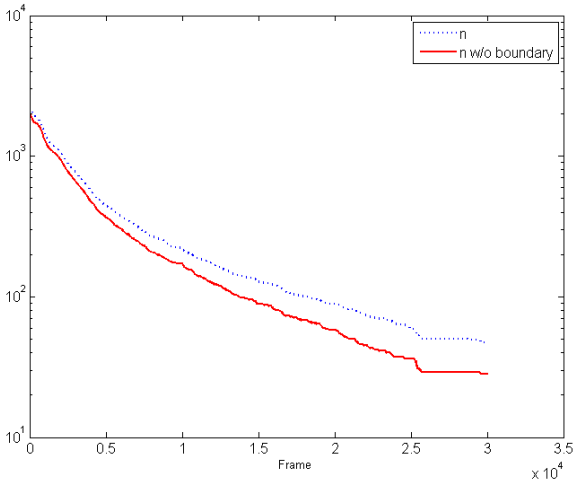

x 10^6
18
16
14
12
10
8
6
4
2
0
Iterations/time
0
1000
2000
3000
4000
5000
6000
7000
Frame

A.2 2.286 cell experiment - With Energy Decay Constraint

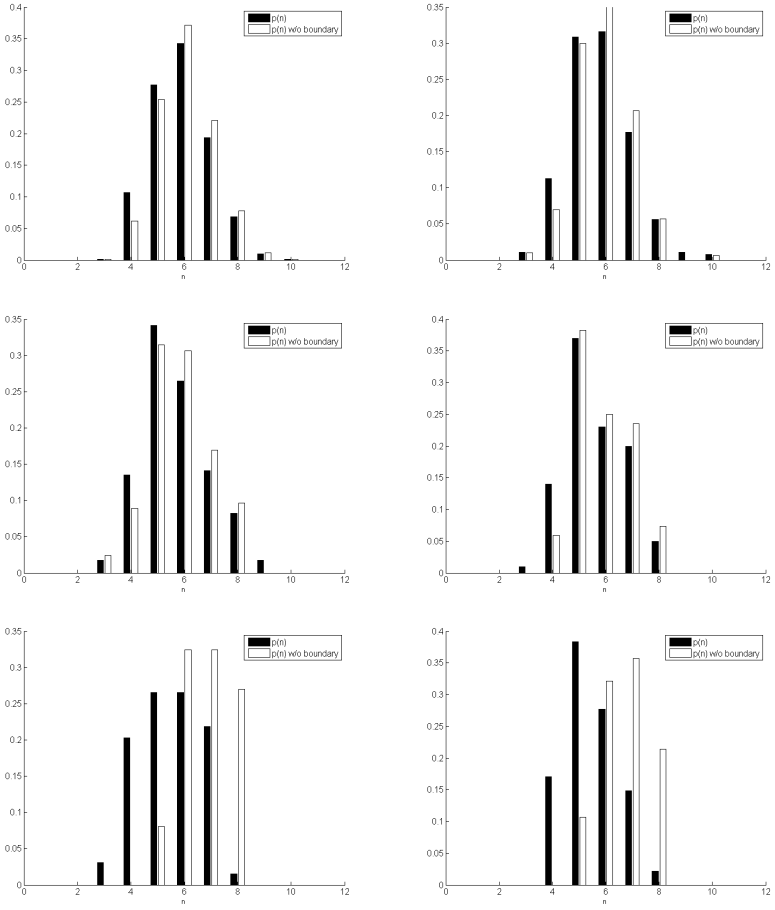
Frames 1, 6.001, 12.001, 18.001, 24.001, and 30.001 from the simulation.



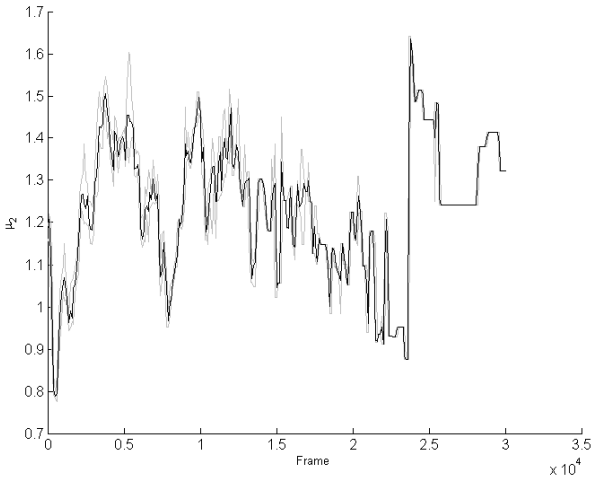
Total number of cells per frame.



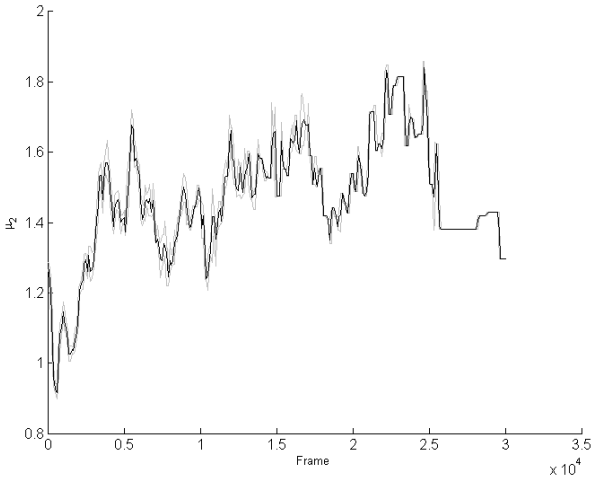
Cell valency distribution $p(n)$ for frames 1, 6.001, 12.001, 18.001, 24.001, and 30.001.



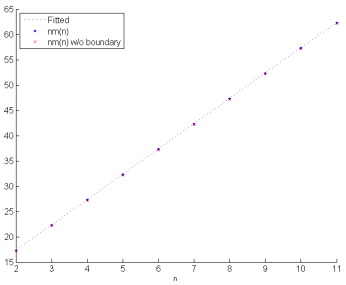
Second moment of $p(n)$ per frame without boundary.

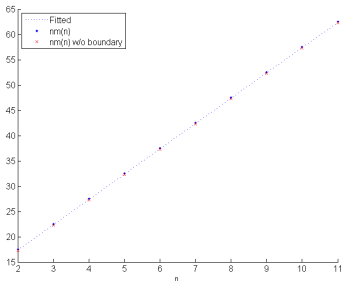


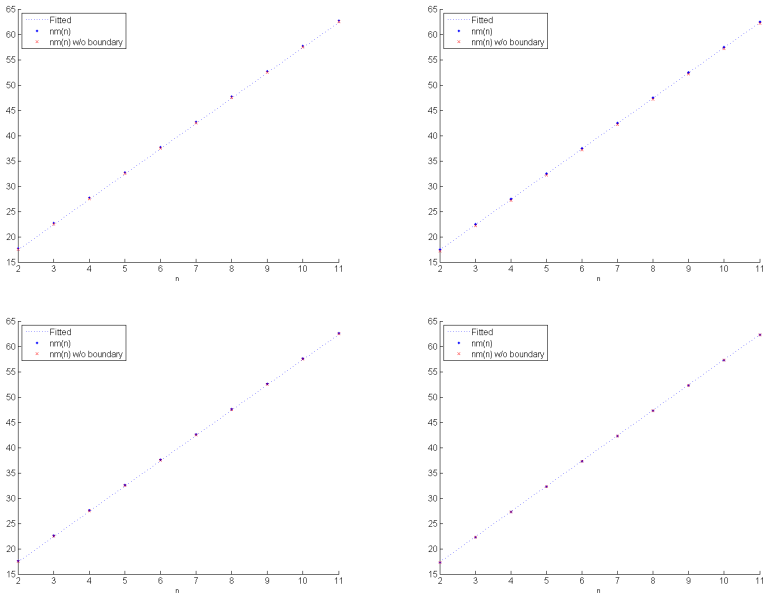
Second moment of $p(n)$ per frame with boundary.



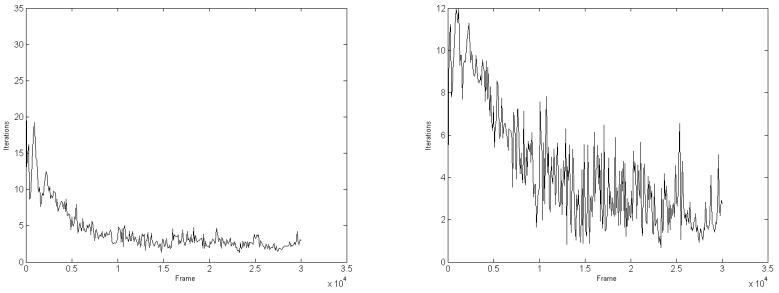
Aboav-Weaire relation $m(n)$ for frames 1, 6.001, 12.001, 18.001, 24.001, and 30.001.



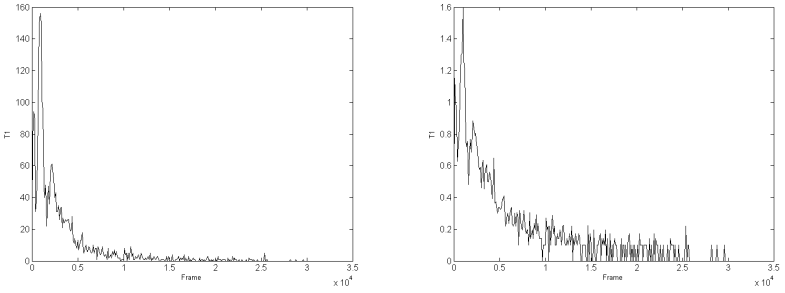




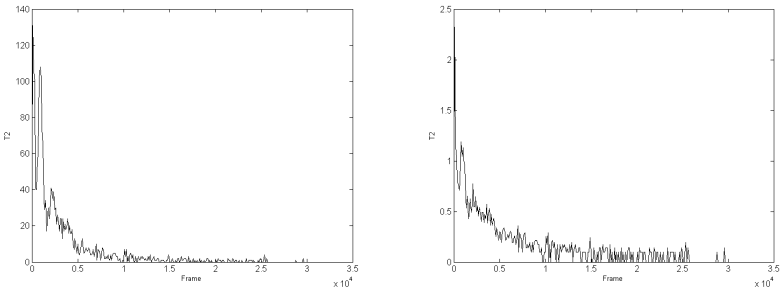
Number of iterations per frame and standard deviation.



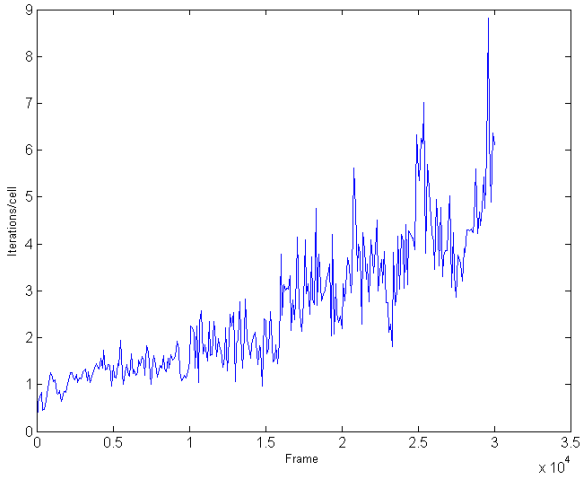
Total number of T1 operations and standard deviation.



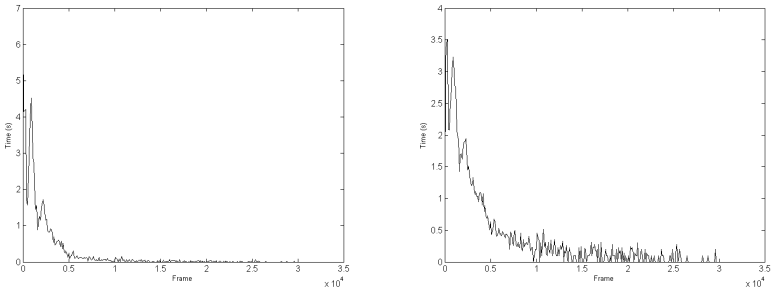
Total number of T2 operations and standard deviation and standard deviation.



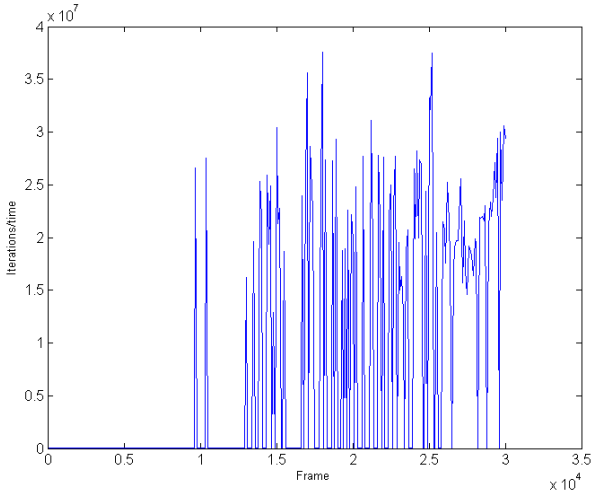
Number of iterations per cell per frame.



Wall-clock time in seconds per frame and standard deviation.

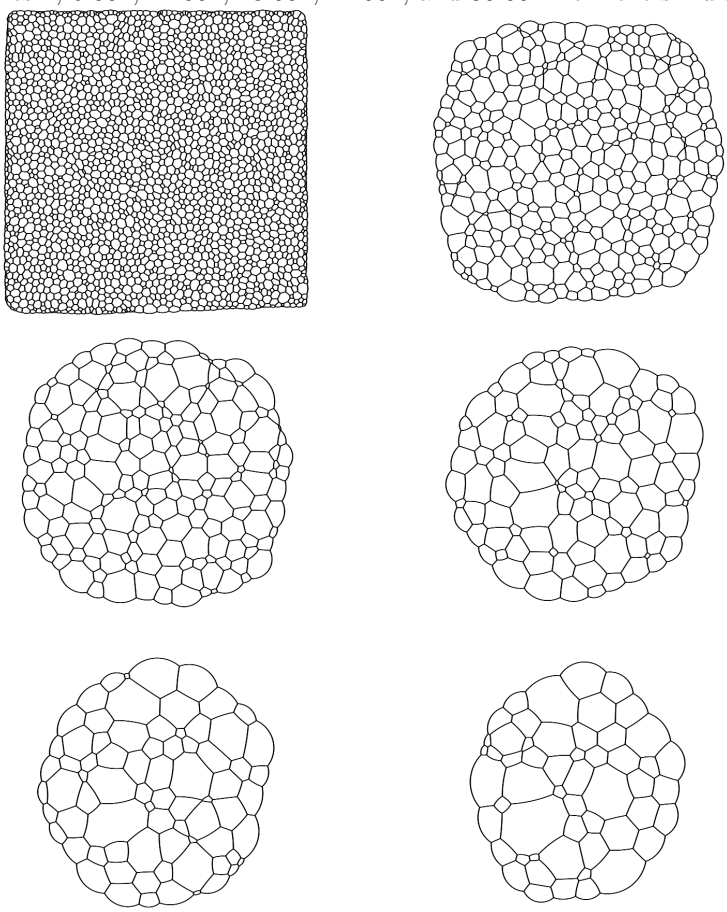


Wall-clock time in seconds per iteration per frame.

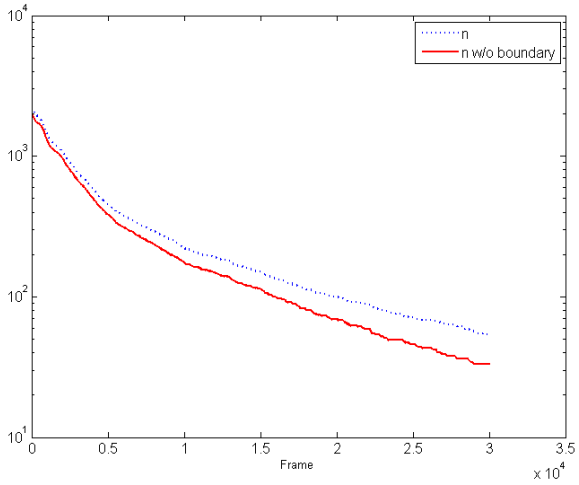

x 10^7
4
3.5
3
2.5
2
1.5
1
0.5
0
Iterations/time
0
0.5
1
1.5
2
2.5
3
3.5
Frame
x 10^4

A.3 2.286 cell experiment - With Orientation Invariant Constraint

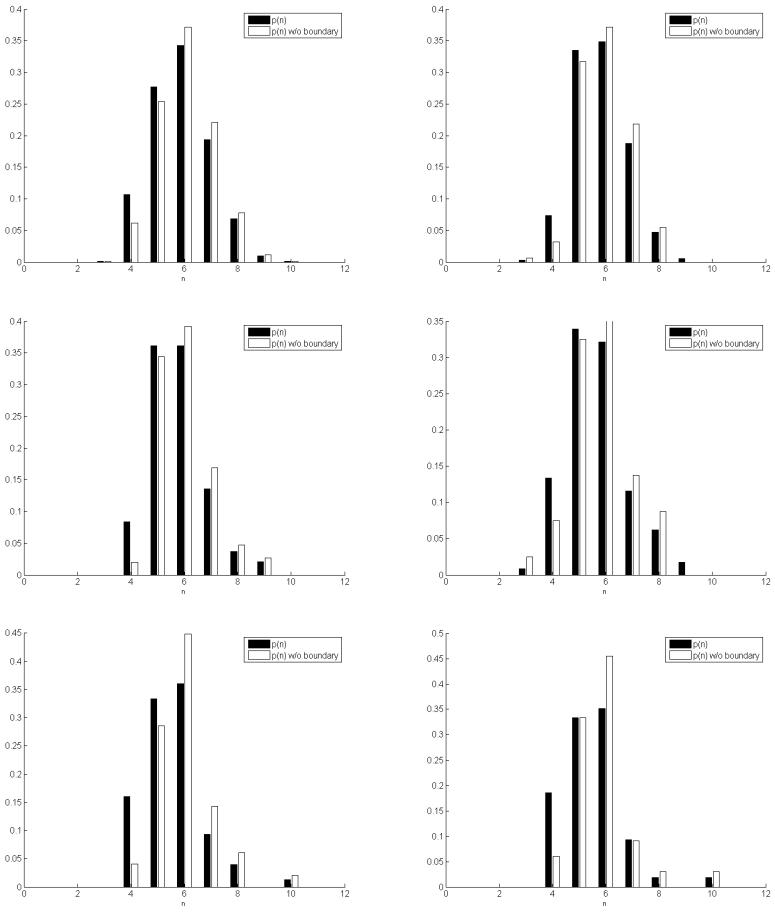
Frames 1, 6.001, 12.001, 18.001, 24.001, and 30.001 from the simulation.



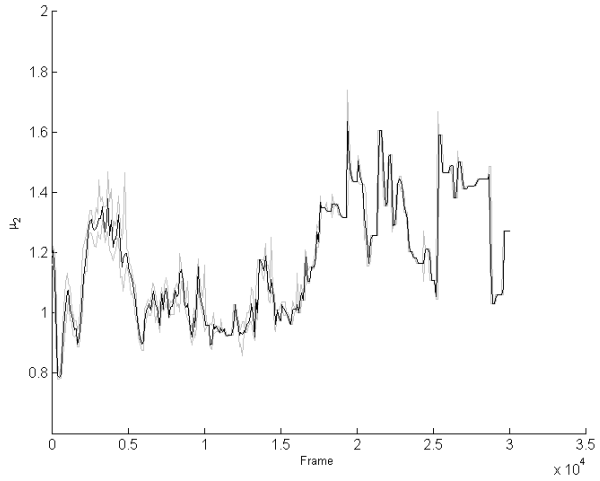
Total number of cells per frame.



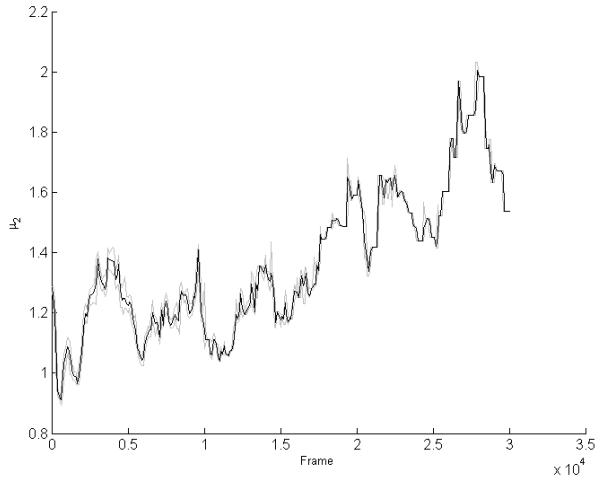
Cell valency distribution $p(n)$ for frames 1, 6.001, 12.001, 18.001, 24.001, and 30.001.



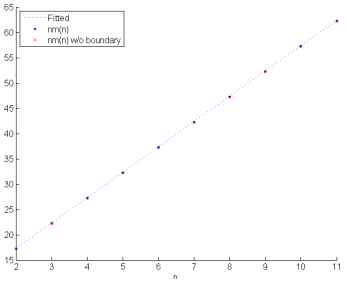
Second moment of $p(n)$ per frame without boundary.

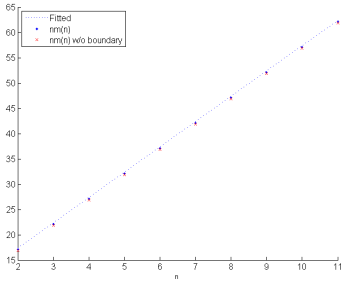


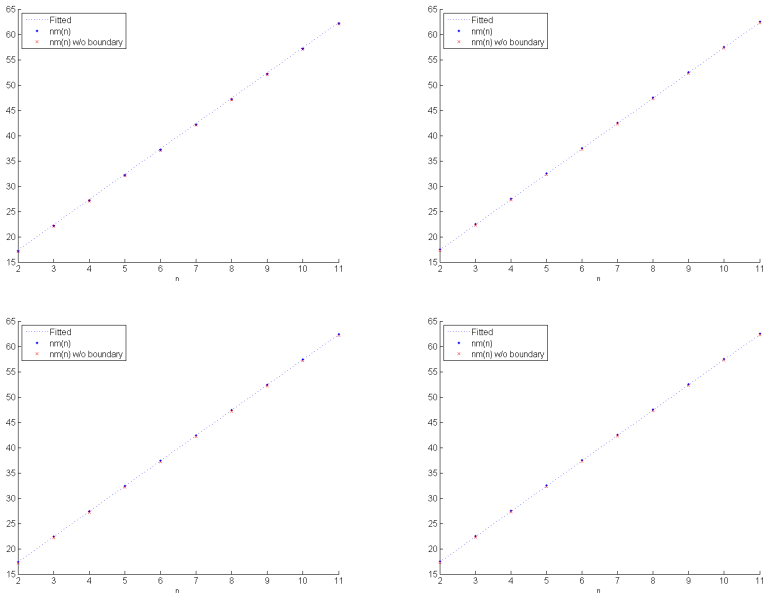
Second moment of $p(n)$ per frame with boundary.



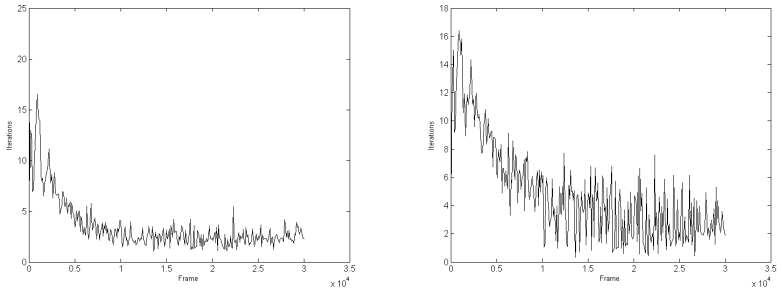
Aboav-Weaire relation $m(n)$ for frames 1, 6.001, 12.001, 18.001, 24.001, and 30.001.



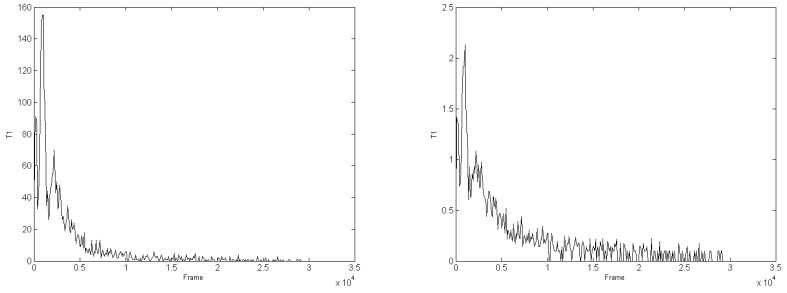




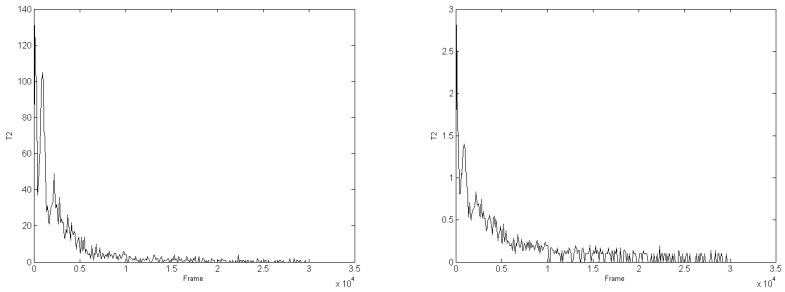
Number of iterations per frame and standard deviation.



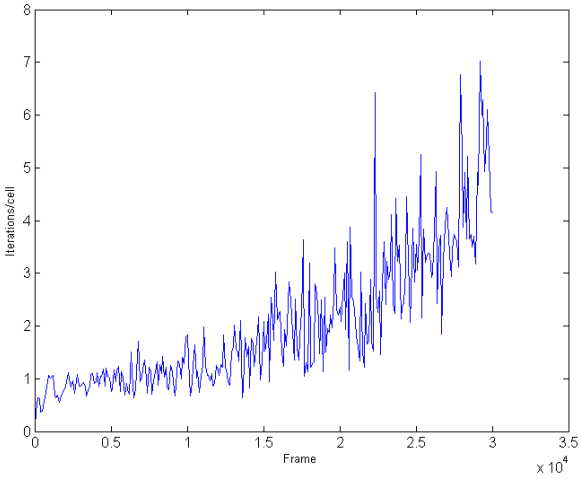
Total number of T1 operations and standard deviation.



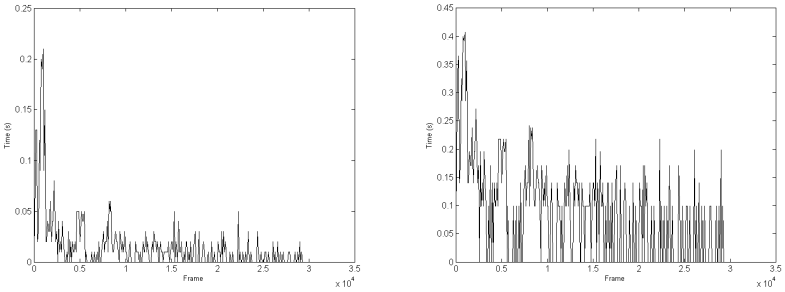
Total number of T2 operations and standard deviation and standard deviation.



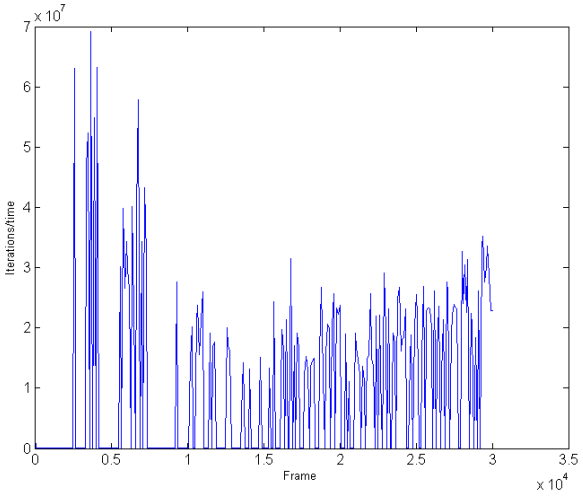
Number of iterations per cell per frame.



Wall-clock time in seconds per frame and standard deviation.



Wall-clock time in seconds per iteration per frame.


x 10^7
7
6
5
4
3
2
1
0
Iterations/time
0
0.5
1
1.5
2
2.5
3
3.5
Frame
x 10^4

Appendix B

Implementation Details

B.1 Gauss-Jordan Reduction

An example of an implementation of a Gauss-Jordan reduction. The input matrix is assumed to be row-major. The implementation allows us to ignore a row/column pair to accomodate for the world bubble.

```
x = b;

// Do a Gauss-Jordan reduction to solve the linear system
// 15 divisions, 47 multiplications

// Lower triangle
// Row 0
if( ignore_row != 0 ) {
    x._0 = x._0 / a._00;
    x._1 = x._1 - x._0 * a._10;
    x._2 = x._2 - x._0 * a._20;
    x._3 = x._3 - x._0 * a._30;
    x._4 = x._4 - x._0 * a._40;

    a._04 = a._04          / a._00; a._03 = a._03          / a._00; a._02
        = a._02          / a._00; a._01 = a._01          / a._00;
    a._14 = a._14 - a._04 * a._10; a._13 = a._13 - a._03 * a._10; a._12
        = a._12 - a._02 * a._10; a._11 = a._11 - a._01 * a._10;
    a._24 = a._24 - a._04 * a._20; a._23 = a._23 - a._03 * a._20; a._22
        = a._22 - a._02 * a._20; a._21 = a._21 - a._01 * a._20;
    a._34 = a._34 - a._04 * a._30; a._33 = a._33 - a._03 * a._30; a._32
        = a._32 - a._02 * a._30; a._31 = a._31 - a._01 * a._30;
    a._44 = a._44 - a._04 * a._40; a._43 = a._43 - a._03 * a._40; a._42
        = a._42 - a._02 * a._40; a._41 = a._41 - a._01 * a._40;
}

// Row 1
if( ignore_row != 1 ) {
    x._1 = x._1         / a._11;
    x._2 = x._2 - x._1 * a._21;
    x._3 = x._3 - x._1 * a._31;
    x._4 = x._4 - x._1 * a._41;

    a._14 = a._14          / a._11; a._13 = a._13          / a._11; a._12
        = a._12          / a._11;
    a._24 = a._24 - a._14 * a._21; a._23 = a._23 - a._13 * a._21; a._22
        = a._22 - a._12 * a._21;
    a._34 = a._34 - a._14 * a._31; a._33 = a._33 - a._13 * a._31; a._32
        = a._32 - a._12 * a._31;
    a._44 = a._44 - a._14 * a._41; a._43 = a._43 - a._13 * a._41; a._42
        = a._42 - a._12 * a._41;
}

// Row 2
```

```
if( ignore_row != 2 ) {
    x._2 = x._2         / a._22;
    x._3 = x._3 - x._2 * a._32;
    x._4 = x._4 - x._2 * a._42;

    a._24 = a._24          / a._22; a._23 = a._23          / a._22;
    a._34 = a._34 - a._24 * a._32; a._33 = a._33 - a._23 * a._32;
    a._44 = a._44 - a._24 * a._42; a._43 = a._43 - a._23 * a._42;
}

// Row 3
if( ignore_row != 3 ) {
    x._3 = x._3         / a._33;
    x._4 = x._4 - x._3 * a._43;

    a._34 = a._34          / a._33;
    a._44 = a._44 - a._34 * a._43;
}

// Row 4
if( ignore_row != 4 ) {
    x._4 = x._4 / a._44;
}

// Upper triangle
// Row 4
if( ignore_row != 4 ) {
    x._3 = x._3 - x._4 * a._34;
    x._2 = x._2 - x._4 * a._24;
    x._1 = x._1 - x._4 * a._14;
    x._0 = x._0 - x._4 * a._04;
} else {
    x._4 = 0.0f;
}

// Row 3
if( ignore_row != 3 ) {
    x._2 = x._2 - x._3 * a._23;
    x._1 = x._1 - x._3 * a._13;
    x._0 = x._0 - x._3 * a._03;
} else {
    x._3 = 0.0f;
}

// Row 2
if( ignore_row != 2 ) {
    x._1 = x._1 - x._2 * a._12;
    x._0 = x._0 - x._2 * a._02;
} else {
    x._2 = 0.0f;
}

// Row 1
if( ignore_row != 1 ) {
    x._0 = x._0 - x._1 * a._01;
} else {
    x._1 = 0.0f;
}

// Row 0
if( ignore_row == 0 ) {
    x._0 = 0.0f;
}
```

B.2 Arc render Geometry Shader

Following is the full Geometry Shader used to construct the arc ribbon for rendering.

```
cbuffer constants {
    row_major matrix WorldViewProjection;
    row_major matrix ViewProjection;
    row_major matrix World;
    float line_thickness;
    float segment_size;
}

struct GS_Input {
    float4 position : POSITION;
    float3 world    : TEXCOORD0;
};

struct GS_Output {
    float4 position : SV_POSITION;
    float2 uv       : TEXCOORD0;
};

#include "arc_straight_edge_gs.fx"
#include "arc_edge_gs.fx"

[maxvertexcount( 128 )]
void GS_ImageSpace( triangle GS_Input IN[3], inout TriangleStream<
    GS_Output > OUT ) {
    float3 x0 = IN[0].position.xyz / IN[0].position.w;
    float3 x1 = IN[1].position.xyz / IN[1].position.w;
    float3 x2 = IN[2].position.xyz / IN[2].position.w;

    float2 normal = IN[1].world.xy - IN[2].world.xy;        // In world
        space
    float  radius = length( normal );                       // In world
        space
    float2 xm     = 0.5f * ( IN[0].world.xy + IN[1].world.xy );
    float  h      = length( xm - IN[2].world.xy );
    float  ratio  = h / radius;

    if( isnan( h ) || ratio > 0.99998f ) {
        GS_StraightEdge( x0, x1, OUT );
    } else {
        GS_ArcEdge( IN, x0, x1, x2, ratio, normal, OUT );
    }

    OUT.RestartStrip();
}

void GS_ArcEdge
    ( GS_Input IN[3]
    , float3 x0
    , float3 x1
    , float3 x2
    , float ratio
    , float2 normal
    , inout TriangleStream< GS_Output > OUT
    )
{
    float theta = 2.0f * acos( ratio );

    // Max segment count is one less than half the maximum vertex count
    int segments  = max( 2, min( 63, (int)floor( length( x1 - x0 ) /
        segment_size ) ) );

    GS_Output o0;
    GS_Output o1;

    // Generate two new vertices on the triangle strip
    for( int i = 0; i < segments + 1; ++i ) {
        float t = (float)i / (float)segments;
```

```
        float s, c;
        sincos( t * theta, s, c );

        float2 rotated = float2( c * normal.x - s * normal.y, s * normal
            .x + c * normal.y );
        float2 xnew    = IN[2].world.xy + rotated;
        float4 pos_is  = mul( float4( xnew, IN[2].world.z, 1 ),
            ViewProjection );
        pos_is /= pos_is.w;

        float3 normal_is = normalize( pos_is.xyz - x2 );

        o0.position = float4( pos_is.xy - normal_is.xy / 2.0f *
            line_thickness, pos_is.z, 1 );
        o0.uv = float2( 1.0f - t, 1 );

        o1.position = float4( pos_is.xy + normal_is.xy / 2.0f *
            line_thickness, pos_is.z, 1 );
        o1.uv = float2( 1.0f - t, 0 );

        OUT.Append( o0 );
        OUT.Append( o1 );
    }
}

void GS_StraightEdge
    ( float3 x0
    , float3 x1
    , inout TriangleStream< GS_Output > OUT
    )
{
    GS_Output o0;
    GS_Output o1;
    GS_Output o2;
    GS_Output o3;

    float2 normal = normalize( float2( x1.y - x0.y, x0.x - x1.x ) );

    o0.position = float4( x0.xy - normal / 2.0f * line_thickness, x0.z,
        1 );
    o0.uv = float2( 1, 1 );

    o1.position = float4( x0.xy + normal / 2.0f * line_thickness, x0.z,
        1 );
    o1.uv = float2( 1, 0 );

    o2.position = float4( x1.xy - normal / 2.0f * line_thickness, x1.z,
        1 );
    o2.uv = float2( 0, 1 );

    o3.position = float4( x1.xy + normal / 2.0f * line_thickness, x1.z,
        1 );
    o3.uv = float2( 0, 0 );

    OUT.Append( o0 );
    OUT.Append( o1 );
    OUT.Append( o2 );
    OUT.Append( o3 );
}
```