Technische Universität Dresden • Faculty of Mathematics

Modelling of Cells and their Dynamics

Bachelorthesis

to obtain the first degree

***Bachelor of Science* (*B.Sc.*)**

by

Tim VOGEL

(born on June 9, 2002 in Finsterwalde)

Day of submission: November 27, 2023

Supervisor: Jun.-Prof. Dr. Markus Schmidtchen
(Institute of Scientific Computing)

Danksagung

Vielen Dank Mum, Dad und Willi. Ihr wart schon immer der beste Rückhalt in meinem Leben. Ohne eure familiäre und emotionale Unterstützung hätte ich den bisher größten Meilenstein meines Mathematiker-Daseins niemals erreicht.

Contents

List of Figures

1 Introduction

The diffusion of finite size particles in confined geometries is a topic of interest in physics, chemistry, and materials science, particularly when studying the behavior of molecules, nanoparticles, or other particles that are not infinitesimally small.
The modeling of cells and the addition of cell dynamics to analyse the diffusion behavior will be the focus of the following thesis.
First of all, some already published cell and particle models are presented in the introduction to relate the subsequent work. Then, a short overview over the different chapters is given. Figure 1 shows some visualisations of the coming models.
Although there are models in higher dimensions, we will from now on focus on those that operate on a bounded domain $\Omega \subset \mathbb{R}^2$ that is a subset of the two dimensional space. All models study the behavior of a number of $N \in \mathbb{N}$ particles.
The point particle model comes first. In the point particle model, one considers particles that are infinitely small. This also means that no interaction between the particles can happen, because the probability of two particles colliding is zero. The particles are initially randomly distributed in Ω. Each particle's movement is described by Brownian motion. In the real world, Brownian motion is the random and continuous zigzagging movement of particles, usually suspended in a fluid medium, resulting from their collisions with surrounding molecules. Brownian motion can be mathematically modeled using stochastic differential equations (SDEs) which describe the particle's movement as a function of time while incorporating random noise terms to represent the irregular, fluctuating forces acting on the particle. Let $\vec{x}_i(t) \in \Omega$, $(1 \leqslant i \leqslant N)$, be the location of the particle i at time $t > 0$. Then, the particle movement can be explained with the diffusion equation

$$d\vec{x}_i(t) = \sqrt{2D}\, dB_t^{(i)}, \qquad 1 \leqslant i \leqslant N,$$

where $D > 0$ is a constant that represents the diffusion coefficient which determines the scale of the random fluctuations. The term $dB_t^{(i)}$ simulates the random Brownian motion with the use of randomly generated, normally distributed random variables.
Next, we want to take a look at $\rho(t, \vec{x})$ representing the probability density function which indicates the probability of finding a particle at a specific position $\vec{x}$ at time t. In the described setting, ρ behaves according to the partial differential equation (PDE)

$$\frac{\partial \rho(t, \vec{x})}{\partial t} = D\Delta_{\vec{x}}\rho(t, \vec{x}), \tag{1}$$

where $\Delta_{\vec{x}}$ symbolises the Laplacian operator with respect to the spatial variables. It uses the same so called Diffusion constant $D > 0$ as the SDE for the particle movement.
Adding a real size to the particles and different interactions to the particle movement will yield the next models. When particles have a finite size, they cannot overlap or occupy the same space simultaneously. This leads to exclusion effects where the presence of one particle restricts the movement of other particles in its vicinity. Since particles cannot overlap, the domain $\Omega_\epsilon^{(i)}$, that holds the information where the centre of particle i can be located, must exclude the areas where

$\|\vec{x}_i - \vec{x}_j\|_2 \leqslant \epsilon$ for all $1 \leqslant j \leqslant N$, $j \neq i$. The domain that holds all possible locations of the particles is then given by $\Omega_\epsilon^N = \Omega_\epsilon^{(1)} \times \ldots \times \Omega_\epsilon^{(N)}$. As a result, the diffusion of finite size particles is inherently different from that of point like particles.
In [Bruna and Chapman, 2012] such a hard sphere particle model is considered. Here, particles are spherical with a diameter $0 < \epsilon \ll 1$. All particles are distinguishable. Hard sphere means that any interaction between different particles might cause a change of the moving direction of the particles, but the spherical shape can never change. Hardcore collision are expressed as reflective boundary conditions on the collision surfaces $r = \|\vec{x}_i - \vec{x}_j\|_2 = \epsilon$, with $1 \leqslant i < j \leqslant N$.
The external forces, such as electromagnetic, friction, convection and potential forces, that act on a particle of the system, are given by the force function

$$f : \mathbb{R}^2 \to \mathbb{R}^2.$$

It is only dependent on the location of the considered particle. The function

$$\vec{F} : \Omega_\epsilon^N \to \mathbb{R}^{2N}, \qquad \vec{F}(\vec{X}) = (f(\vec{x_1}), \ldots, f(\vec{x_N}))^T,$$

where $\vec{X} = (\vec{x}_1, \ldots, \vec{x}_N)^T$, holds the information of the external forces for every particle.
The dynamic of the particles changes to

$$d\vec{x}_i(t) = \sqrt{2D}\, dB_t^{(i)} + f(\vec{x}_i(t))\, dt, \qquad 1 \leqslant i \leqslant N.$$

In this model, the particles are initially randomly distributed in Ω_ϵ^N such that no overlap between the particles occurs. As derived in [Bruna and Chapman, 2012], the joint probability density function P (joint PDF) of the N particles satisfies a high dimensional Fokker-Planck equation

$$\frac{\partial P}{\partial t} = \nabla_{\vec{X}} \cdot (D \nabla_{\vec{X}} P - P\vec{F}),$$

where $\nabla_{\vec{X}}$ and $\nabla_{\vec{X}}\cdot$, respectively, stand for the gradient and divergence operators with respect to the N particle position vector $\vec{X}$.
Using the method of matched asymptotic expansions, the authors also derived the probability density function ρ of finding a single particle at time t and position $\vec{x}$ which fulfils

$$\frac{\partial \rho(t, \vec{x})}{\partial t} = \Delta_{\vec{x}} \rho + \alpha_d (N-1) \epsilon^d \Delta_{\vec{x}}(\rho^2) - \nabla_{\vec{x}} \cdot (f(\vec{x})\rho). \tag{2}$$

If one neglects f and lets $\epsilon \to 0$, one obtains the probability density function of the point particle model (1), except for a rescaling factor. Thus, this model is a direct extension of the point particle model.
This model can be extended, for example, by using soft spherical particles. In the soft sphere model, particles are treated as spheres that can deform or interact through a potential energy function that depends on the distance between them.
The paper [Bruna et al., 2017], written by Bruna, Chapman and Robinson, analyses the diffusion properties of such a model. The new interaction potential u is applied pairwise to all particles, i.e.

$$d\vec{x}_i(t) = \sqrt{2D}\, dB_t^{(i)} + f(\vec{x}_i(t))\, dt - \sum_{j \neq i} \nabla_{\vec{x}_i} u(\|\vec{x}_i(t) - \vec{x}_j(t)\|_2)\, dt, \qquad 1 \leqslant i \leqslant N,$$

where $\nabla_{\vec{x}_i}$ is the gradient with respect to $\vec{x}_i$. The interaction potential u can be repulsive or attractive and it can vary in complexity. The presence of an interaction potential affects the dynamics of the particles. For the modeling of short range interacting soft sphere particles, the authors computed the one particle probability density $\rho(t, \vec{x})$ of finding a given particle at position $\vec{x}$ at time t developing according to

$$\frac{\partial \rho}{\partial t} = \nabla_{\vec{x}} \cdot (D \nabla_{\vec{x}} \rho - f(\vec{x})\rho + \alpha_u \epsilon_u^2 (N-1) \rho \nabla_{\vec{x}} \rho), \tag{3}$$

where α_u depends on the interaction potential u and $0 < \epsilon_u \ll 1$ is the interaction range of u. While the models presented only deal with spherical particles, in this thesis a large variety of shapes should be able to be modeled. The shape of a cell is another property that can influence the change in particle density over time in the domain. More complex shapes can be represented as in the model below, which can be assigned to the category of phase field models.

Such a model is presented in [Happel and Voigt, 2023]. Here, each cell is modeled via a two dimensional phase field variable which is a smooth, continuous function $\phi_i \in [-1, 1]$, $(1 \leqslant i \leqslant N)$. Vectors $\vec{x}$ with positive values in ϕ_i are interior of the cell i and all vectors with a negative value are considered to be exterior. Values of $\phi_i = 0$ denote the cell wall. The dynamic for each ϕ_i reads

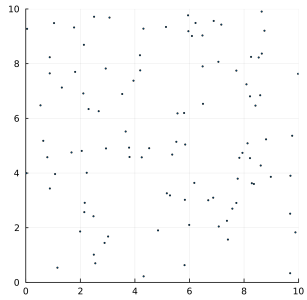
$$\frac{\partial \phi_i}{\partial t} + v_0(\mathbf{v}_i \cdot \nabla_{\vec{x}} \phi_i) = \Delta_{\vec{x}} \frac{\delta F}{\delta \phi_i}, \qquad 1 \leqslant i \leqslant N, \tag{4}$$

where $\mathbf{v}_i$ is a vector field used to incorporate activity, with a self propulsion strength v_0, F is a free energy and $\dfrac{\delta F}{\delta \phi_i}$ denotes the first variation. F results from a sum of different energies which are e.g. used to ensure that ϕ_i stays between -1 and 1, that the cell keeps a certain shape, or that the different cell interactions are correctly included.

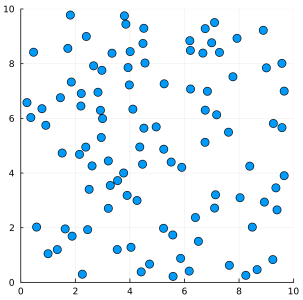
An easier approach to model cell shapes are vertex models, like for example in [Fletcher et al., 2014]. Vertex models are part of a broader class of computational models used in computational biology and biophysics to study the biomechanics and behavior of cells and tissues. They are valuable tools for understanding how cells respond to mechanical cues and how their behavior contributes to biological processes. In a vertex model of a cell, the cell's outline or boundary is approximated as a polygon. The vertices of this polygon represent discrete points along the cell's boundary. Movements or transformations of the cell are given by forces that are applied on each vertex individually. In [Fletcher et al., 2014] the cell dynamic reads

$$\eta \frac{d\vec{x}_i}{dt} = F_i, \qquad 1 \leqslant i \leqslant N, \tag{5}$$

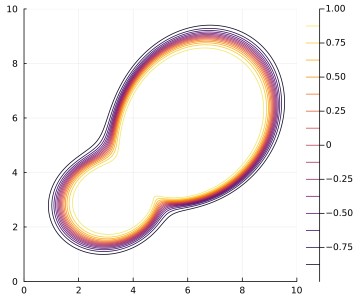
where η is a scaling factor and F_i is the total force acting on $\vec{x}_i$. Like in the phase field model, F_i is a sum of different forces that define the cell behavior, such as the cell flexibility or the interaction with other cells.



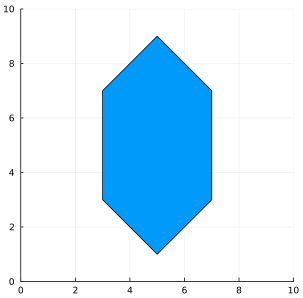
(a) Here, one can see a possible snapshot of the point particle model that is explained at the beginning. The particles are randomly distributed in Ω. The execution of the Brownian motion will cause the particle density to change according to the Diffusion equation.



(b) In contrast to Figure (a), the particles do have a real radius in the sphere models. This causes particle interaction terms as addition in the particle movement, since particles can now collide. The type of interaction terms determines whether it is a hardsphere or softsphere model.



(c) A contour plot of a phase field variable ϕ illustrates how cells can be modeled through phase field models. The cell's inside is the area where $\phi > 0$. The cell wall sits on the red line where $\phi = 0$. The outer lines display the smooth transition to the outside.



(d) Another possibility to model cell forms are Vertex models. An example of this is shown here. This cell has six vertices. In order to model cell deformations, one can define forces that act on each vertex separately and thus cause them to move in an according direction.

Figure 1: To illustrate the models from the introduction, we can see a corresponding plot for each model. In (a) and (b) the focus is more on the density distribution of the particles in Ω. The amount of particles is $N = 100$. The remaining sub figures (c) and (d) are concerned with the representation of cell shapes. In all sub figures, the axes denote the spatial x and y coordinates.

The aim of this work is to introduce mathematical models of cells that can cover a wide range of shapes like the models in [Happel and Voigt, 2023] and also in [Fletcher et al., 2014]. We also want to derive forces that are applicable to the new models and that have an influence on the diffusion behavior of the cell systems as in [Bruna and Chapman, 2012] and [Bruna et al., 2017].
Section 2 defines such models, named centre radius form (CRF) and discrete form (DF). It also gives some functionalities that will be needed in order to implement desired cell actions and interactions.

On one hand, the CRF represents a continuous form of a cell. A centre point and a radius function allow a precise representation of the cell in which any number of wall points can be calculated. Every bounded and star shaped domain that has a smooth boundary can be expressed via a CRF. The smooth indicator function, used to plot CRF cells and compute their areas or cell overlaps, has similarities to a phase field variable described in [Happel and Voigt, 2023]. In Section 3 a dynamic is introduced that allows the transition from one CRF to another.
On the other hand, we have got DF cells. This is a vertex model, similar to the model considered in [Fletcher et al., 2014]. Various forces and interactions between the vertices and cells are modeled to simulate cell behavior in Section 4.
At the end of this thesis, advantages and disadvantages of the introduced models, as well as possible improvements and extensions are discussed in the conclusion.
A large part of the development of this thesis is the implementation and simulation of the cell forms and their dynamics. The Julia programming language is used for this. Julia offers advantages such as high performance numerical computing, easy integration with existing code written in languages like C and Fortran and a user friendly syntax for rapid development of scientific and data analysis applications.
Hereinafter, any points $\vec{x} \in \mathbb{R}^d$ ($d \in \mathbb{N}_{>1}$) are written with a superscript arrow. The line segment between two points $\vec{x}, \vec{y} \in \mathbb{R}^d$ is expressed through the notation $\overline{\vec{x}\,\vec{y}}$.

2 Cell definitions and area computations

Before we are able to run programs and simulate cell behavior, we must first set a definition of what exactly a cell is, i.e. how we want to model them. Thus, this section will give such a simplified mathematical model of a cell.
In this thesis, we will always consider the the two dimensional space of real numbers. Consequently, a cell will always be embedded in $\mathbb{R}^2$. Since we want to take a look at the development of the cell shape, we want a model that provides as much freedom of shape as possible. Ergo, simple discs or other static geometric forms are not versatile enough. Instead, we take a look at the following approach.

2.1 Continuous and discrete cell models

In the first approach, we define a cell by its centre point and a so called radius function which takes an angle $\phi \in [0, 2\pi)$ and returns the cell radius at that point.

Definition 2.1. Centre radius form (CRF)
For a given point $\vec{c} \in \mathbb{R}^2$ and function $r : [0, 2\pi) \to (0, \infty)$ so that $r \in C^1([0, 2\pi))$, $r(0) = r(2\pi)$ and $r'(0) = r'(2\pi)$, the tuple $(\vec{c}, r)$ describes a cell in its centre radius form (CRF) with centre point $\vec{c}$ and radius $r(\phi)$ at angle $\phi \in [0, 2\pi)$ that is taken with the x axis.

Since we want our cell wall to be smooth, it is a reasonable requirement to choose our radius function r to be smooth as well. The boundary conditions ensure a soft transition at angle $\phi = 0$ or, respectively, $\phi = 2\pi$. With this form we are able to describe and plot cells that have a star shaped form and a smooth wall. One can easily imagine that there are also cell forms which are not star shaped and which cannot be displayed with the CRF. But hereinafter, we will just focus on the wide

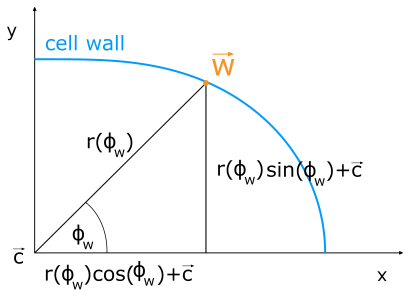
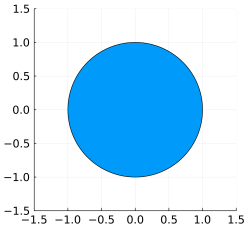
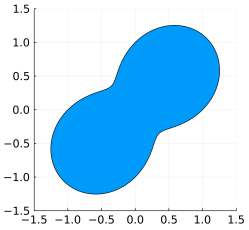


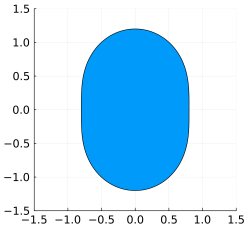
Figure 2: A sketch of the calculation of the x and y coordinates of a vertex of a cell in its CRF.



(a) unit circle cell:
$r_a(\phi) = 1$



(b) peanut shaped cell:
$r_b(\phi) = \frac{1}{2}\sin(2\phi) + 1$



(c) oval cell:
$r_c(\phi) = 1 - \frac{1}{5}\cos(2\phi)$

Figure 3: First examples of cell plots with centre $\vec{c} = (0,0)^T$.

range of shapes that can be represented by a CRF cell.
For every point on the cell wall $\vec{w}$, we can draw a right triangle and then use the theorems on trigonometric functions in right triangles to compute

$$\vec{w} = \begin{pmatrix} x \\ y \end{pmatrix} = r(\phi_w) \begin{pmatrix} \cos(\phi_w) \\ \sin(\phi_w) \end{pmatrix} + \vec{c},$$

as we can see in Figure 2.
In order to obtain our first plots, we just have to save an amount of vertices using this equation and then perform a shape plot with this list of points. Figure 3 shows us some examples.
Having established a continuous cell model, we are now also looking for a further simplification. The reason for this is that most cell dynamics in this thesis are applied on a vertex model in Section 4. It is straight forward to just use a sequence of vertices. First of all, we will introduce some definitions of two dimensional objects and their characteristics.

Definition 2.2. Star domain
Any subset $S \subset \mathbb{R}^2$ is star shaped if and only if there exists a point $\vec{c} \in S$ such that the connecting line from $\vec{c}$ to any other point $\vec{x} \in S$ is completely located in S. The point $\vec{c}$ is then called star centre.

Definition 2.3. Convex domain
A domain $D \subset \mathbb{R}^2$ is said to be convex if for any two points $\vec{x}_1, \vec{x}_2 \in D$ their connection $\overline{\vec{x}_1\,\vec{x}_2}$ lies entirely in D.

Definition 2.4. Polygon
A polygon is a closed geometric figure in the two dimensional space $\mathbb{R}^2$, formed by connecting a finite number of straight line segments. It can be represented be a sequence $(\vec{x}_1, \ldots, \vec{x}_N)$ of its vertices.
The following characteristic can be attributed to a polygon.

(i) A **simple** polygon is a polygon where no two line segments cross each other.

(ii) A polygon has a **positive orientation** if the vertices are ordered counterclockwise.

(iii) A polygon has a **negative orientation** if the vertices are ordered clockwise.

An equivalent criterion for a polygon to be convex is that each interior angle is less than or equal to 180°. Every convex polygon is also star shaped. If a polygon P is convex, one can even choose any point $\vec{x} \in P$ as the star centre.

Definition 2.5. Discrete form (DF)
An ordered sequence of points $C = (\vec{x}_1, \ldots, \vec{x}_N)$ is considered to be a cell in its discrete form (DF) if the polygon that results when connecting every point with its neighbours and $\vec{x}_1$ with $\vec{x}_N$ is simple and positively orientated.
Cells in the DF model are sometimes just called discrete cells.

Relationship between CRF and DF cells
Having introduced the two Definitions 2.1 and 2.5, we will shortly give a comparison of these different cell models.
On one hand, the CRF model is able to describe not only an endless amount of vertices, like the DF model, but it can explain the whole cell wall as a continuous path. This is why we call the model continuous. Using the CRF model, one can compute arbitrary many vertices like portrayed in Figure 2, enabling to plot the cells in a correspondingly high resolution. CRF cells always have some good properties. In a CRF $C = (\vec{c}, r)$ every angle around the centre has exactly one radius value what makes it impossible for the cell wall to cross itself. It is also not very hard to recognise that those cells are always star shaped with the star centre $\vec{c}$.
On the other hand, we have got the DF model. It is simpler than the first model, because it does not deliver the information of the whole cell wall, but only some wall points of it. Thus, we call this model discrete. The wall points are called vertices in this thesis, correlating to the paper [Fletcher et al., 2014] from the introduction. With a DF model, one can represent any simple polygon. This means that, as a rule, no further properties of these cells can be detected.
There are possibilities to compare cells from the different models.

Definition 2.6. Matching cells
A DF cell ζ and a CRF cell C are called matching if all vertices in ζ are located on the cell wall described by C.

DF cells that match a CRF cell inherit the property of being star shaped.
There are possibilities to switch from one model to the other. An example of switching from CRF to DF is the following. For $N \in \mathbb{N}$ angles

$$\phi_i = i\frac{2\pi}{N}, \qquad 0 \leqslant i \leqslant N-1,$$

compute the according vertices $\vec{x}_1, \ldots, \vec{x}_N$ like in Figure 2 and save them in this order.

It is also possible to create a matching CRF out of a given DF if it is star shaped. To do so, one has to set $\vec{c}$ as a star centre of the DF cell. Afterwards, the radius function r can be obtained through a smooth interpolation between the fixed radii $r_i = \|\vec{c} - \vec{x}_i\|_2$ at the corresponding angles of rising ϕ_i between the line $\overline{\vec{c}\,\vec{x}_i}$ and the x axis. In order to obtain a radius function fulfilling Definition 2.1, one must also consider the smoothness requirements at the interval ends of $[0, 2\pi]$.
Note, that the changeover from one model to another is never unique. For example, if one wants to derive a CF cell out of a CRF, there are different amounts N of vertices possible. Or, if one wants to change the model from DF to CRF, there are different interpolations for r possible.
It is important to understand that it is not possible to draw a one to one correspondence between two cells from the different models, because one can always convert a CRF cell to a DF cell ambiguously and vice versa.

2.2 Computation of area and overlap

Next, we want to figure out how to calculate the area of such cells. Therefor we choose the now coming approach. Let us at first focus on cells in their CRF. In this subsection C and ζ always represent CRF cells. If we had given the indicator function $\mathbb{1}_C(\vec{x})$, that has the value 1 if the point $\vec{x} \in \mathbb{R}^2$ lies within the cell $C \subset \mathbb{R}^2$ and 0 otherwise, then we could obtain the cell area with the integral

$$A_C = \int_{\mathbb{R}^2} \mathbb{1}_C(\vec{x}) d\vec{x}.$$

That is why we now construct a similar function with the information we get from the cell's CRF. First we take a look at the function

$$f(x) = \tfrac{1}{2}(1 + \tanh(s[1 - |x|])),$$

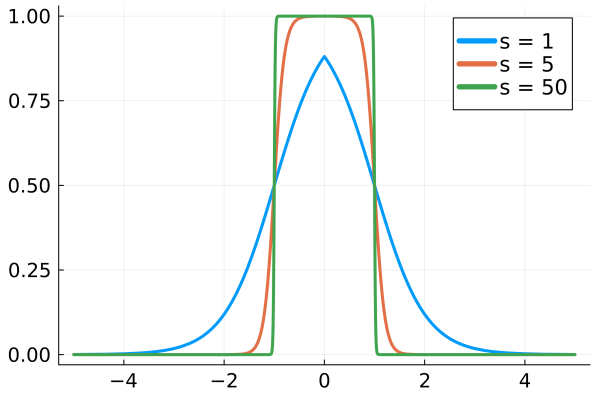


Figure 4: Different plots of $f(x)$ for $s = 1$ (blue), $s = 5$ (orange) and $s = 50$ (green). The x axis denotes the value of the argument x, while the y axis denotes the value of f at that point.

with a positive number $s \in \mathbb{R}_{>0}$. As we can see in Figure 4, the larger we choose s, the more this function looks like $\mathbb{1}_{[-1,1]}$.

For large s, f is nearly 1 if the term in the hyperbolic tangent is positive and nearly 0 otherwise. Hence, we set our $s \gg 1$ and choose a different term inside of the hyperbolic tangent which is positive if and only if $\vec{x} \in C$. The following computation will give us such a term. Let $\phi_{\vec{x}}$ be the angle of rise of the line segment that connects the centre $\vec{c}$ with $\vec{x}$ in respect to the x axis. The angle of rising can also be computed using the trigonometric functions.
Then, we can observe that a point $\vec{x} \in \mathbb{R}^2$ is in C if and only if

$$r(\phi_{\vec{x}}) \geqslant \|\vec{c} - \vec{x}\|_2,$$

what is equivalent to

$$r(\phi_{\vec{x}}) - \|\vec{c} - \vec{x}\|_2 \geqslant 0.$$

Thus, we establish our indicator comparable function, that we will call ψ, like this.

Definition 2.7. Smooth indicator function
For a cell $C = (\vec{c}, r)$ given in its CRF, we declare the smooth indicator function to be

$$\psi_C : \mathbb{R}^2 \to (0,1),\ \vec{x} \mapsto \tfrac{1}{2}(1 + \tanh(s[r(\phi_{\vec{x}}) - \|\vec{c} - \vec{x}\|_2])),$$

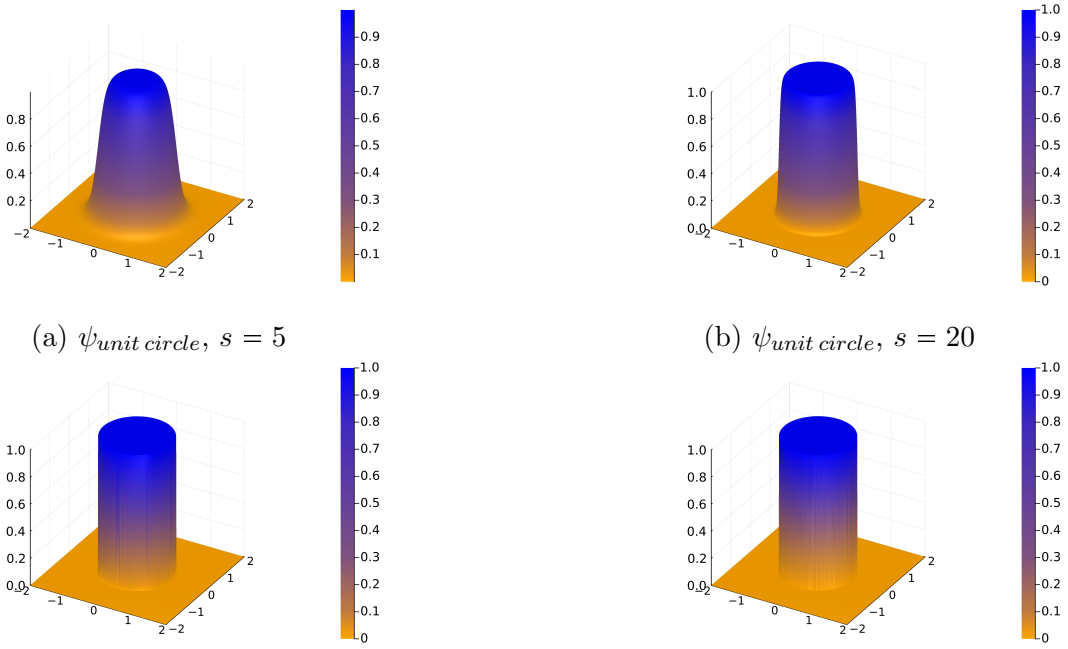
where $\phi_{\vec{x}}$ is the angle of rise of the line segment that connects the centre $\vec{c}$ with $\vec{x}$ in respect to the x axis. The scaling factor $s \gg 1$ determines how rapid the transition from inside and outside of the cell is if one draws the analogy to the normal indicator function.

If we take a look at the cell $((0,0)^T, \phi \mapsto 1)$ that is represented by the unit circle, we obtain

$$\psi_{unit\,circle}(\vec{x}) = \tfrac{1}{2}(1 + \tanh(s[1 - \|\vec{x}\|_2])).$$

Right down to the radius function of the cell, every other function that is composed in ψ is infinitely differentiable. That is why the smooth indicator function has the same smoothness as r.
In Figure 5 we get to see plots of $\psi_{unit\,circle}$ for different s and the indicator function $\mathbb{1}_{unit\,circle}$. As expected, the larger we choose s, the steeper the slope gets and the closer $\psi_{unit\,circle}$ gets to $\mathbb{1}_{unit\,circle}$. For smaller s the product $\psi_C \psi_\zeta$ can be greater than zero in areas near both cells C and ζ where they actually do not meet. This could be used to add neighbouring cell interaction, although these cells are not yet overlapping.
As mentioned before, we can now use the smooth indicator function to compute the area of a cell as we calculate the integral

$$A_C = \int_{\mathbb{R}^2} \mathbb{1}_C(\vec{x}) d\vec{x} \approx \int_{\mathbb{R}^2} \psi_C(\vec{x}) d\vec{x} = \tfrac{1}{2} \int_{\mathbb{R}^2} (1 + \tanh(s[r(\phi_{\vec{x}}) - \|c - \vec{x}\|_2])) d\vec{x}.$$



(a) $\psi_{unit\,circle}$, $s = 5$

(b) $\psi_{unit\,circle}$, $s = 20$

(c) $\psi_{unit\,circle}$, $s = 500$

(d) real indicator $\mathbb{1}_{unit\,circle}$

Figure 5: Plots of $\psi_{unit\,circle}$ with different $s \in \{5, 50, 500\}$ in comparison with the normal indicator function of the unit circle.

With these smooth indicator functions, we are also able to compute the overlap of different cells. We just have to draw the analogy to the standard indicator function once more to see

$$A_{C\cap\zeta} = \int_{\mathbb{R}^2} \mathbb{1}_{C\cap\zeta}(\vec{x})d\vec{x} = \int_{\mathbb{R}^2} \mathbb{1}_C(\vec{x})\mathbb{1}_\zeta(\vec{x})d\vec{x} \approx \int_{\mathbb{R}^2} \psi_C(\vec{x})\psi_\zeta(\vec{x})d\vec{x}.$$

Thus, we just have to multiply the smooth indicator functions of all cells that we want to determine the overlap from. The final numerical implementation and its properties will be discussed in the next paragraph.
For now, we want to take a look at the area calculation of the discrete model. Fortunately, there exists a direct formula for the area of simple polygons.

Proposition 2.8. ***Shoelace formula for DF cells***
Let $C = (\vec{x}_1, \dots, \vec{x}_N)$ be a positively orientated DF cell with $\vec{x}_i = (x_i, y_i)^T$. Then, the area A_C of C can be determined through the shoelace formula

$$A_C = \frac{1}{2}\sum_{i=1}^{N}(x_i y_{i+1} - x_{i+1}y_i),$$

where $\vec{x}_{N+1} := \vec{x}_1$.
Proof.
An illustration of the proof is given in Figure 6, which is where the idea of the proof comes from. We can assume without the loss of generality that all coordinates are positive. Otherwise we could move the figure in the positive coordinate directions

until all entries are positive without changing the area.
For every $1 \leqslant i \leqslant N$ *the edge* $\overline{\vec{x}_i\,\vec{x}_{i+1}}$ *gets assigned to the area* T_i *of the trapeze that arises when one connects the line segment vertically with the* x *axis. We get*

$$T_i = \tfrac{1}{2}(y_i + y_{i+1})(x_i - x_{i+1}).$$

Keep in mind that $\vec{x}_{N+1} := \vec{x}_1$.
The area T_i *has a positive sign if* $x_i \geqslant x_{i+1}$ *(green arrow in Figure 6) and a negative sign otherwise (red arrow). As one can see in the figure, the negatively signed* T_i*'s take up exactly the areas that are superfluous if one just adds all the positive* T_i*'s. Thus the total polygon's area is equal to the sum of all trapezes*

$$A_C = \sum_{i=1}^{N} T_i = \tfrac{1}{2}\sum_{i=1}^{N}(y_i + y_{i+1})(x_i - x_{i+1}) = \tfrac{1}{2}\sum_{i=1}^{N}(x_i y_{i+1} - x_{i+1}y_i).$$

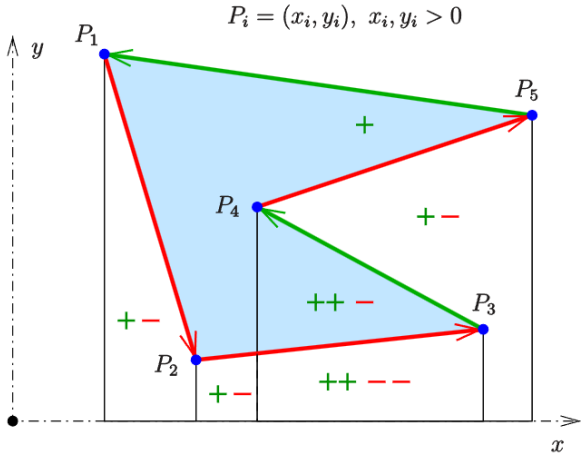


Figure 6: This figure shows a geometrical interpretation of the shoelace formula. In difference to the proposition, here the vertices are called P_i and not $\vec{x}_i$.
Source: [ShoelaceIllustration, 2022]

□

This method is sometimes also called Gauss's area formula. A derivation of the formula can be found in [ShoelaceFormula, 2014].
Thus, we have got a quiet easy but powerful way to obtain the area of a single discrete cell. In order to evaluate the overlap of two discrete cells, we have to determine new cells which exactly cover the overlapping area. Once these cells are computed, it is no problem to compute the overlapping area by just using the shoelace formula on them. How to do that will be answered in the consecutive paragraph.

2.3 Numerical approximation for the continuous area computation

Until now, we have discussed ways to find out cell areas and overlaps of different cells of both the continuous and discrete model.

For the continuous case, where we have got the cells in their CRF, we introduced the smooth indicator function which delivers the cell area via the integral

$$A_C \approx \int_{\mathbb{R}^2} \psi_C(\vec{x})d\vec{x} = \frac{1}{2}\int_{\mathbb{R}^2}(1+\tanh(s[r(\phi_{\vec{x}}) - \|c-\vec{x}\|_2]))d\vec{x}.$$

As this integral is really hard to solve explicitly, we rather want to solve it with a numerical method that we will now derive.
First of all, we need to find a domain over which we will integrate. It makes sense to choose a rectangle so that we can create a grid over it without much effort. Therefor, we iterate over the interval $[0, 2\pi)$ and compute the cell vertices at many angles with small constant shifts to find the coordinates x_0, x_M, y_0, y_N that represent the x coordinates most left and right and the y coordinates most up and down. The rectangle $R = [x_0, x_M] \times [y_0, y_N]$ will now cover the whole cell quite accurately. Further on, we have to choose the amounts $M, N \in \mathbb{N}$ that will represent how many times we divide the sides of the rectangle, to obtain the mesh

$$\begin{aligned} h_x &:= \frac{x_M - x_0}{M}, \qquad & h_y &:= \frac{y_N - y_0}{N}, \\ x_i &:= x_0 + ih_x, & 0 &\leqslant i \leqslant M, \\ y_j &:= y_0 + jh_y, & 0 &\leqslant j \leqslant N. \end{aligned}$$

Every point $(x_i, y_j)^T$ is a grid point of our mesh. We also need the midpoints of each sub rectangle which are given by the term

$$\vec{m}_{i,j} := \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \begin{pmatrix} ih_x \\ jh_y \end{pmatrix} - \frac{1}{2}\begin{pmatrix} h_x \\ h_y \end{pmatrix}, \qquad 1 \leqslant i \leqslant M,\ 1 \leqslant j \leqslant N,$$

for our numerical approximation. With these preparations we are now ready to use the summed up midpoint rule.

Proposition 2.9. ***Summed up midpoint rule***
Let $f : R \to \mathbb{R}$ be a function that is integrable over a rectangular domain $R \subset \mathbb{R}^2$. For given increments h_x, h_y and midpoints $\vec{m}_{i,j}$ like above, we can approximate the integral of f over R by the formula

$$\int_R f(\vec{x})d\vec{x} \approx h_x h_y \sum_{i=1}^{M}\sum_{j=1}^{N} f(\vec{m}_{i,j}) =: Q^R_{N,M}(f).$$

Although this quadrature formula will not give us the most precise estimation of the integral, it is simple and will still deliver a good intuition of the size of the cell. As we can see in the next proposition, the error can be reduced by lowering h_x and h_y, which leads to more grid points in each direction.

Proposition 2.10. ***Error estimation of summed up midpoint rule***
The summed up midpoint rule is exact for polynomials of degree 1. If $f \in C^2(R)$, the error that is made with this quadrature formula can be limited by

$$|\int_R f(\vec{x})d\vec{x} - Q^R_{N,M}(f)| \leqslant h_x h_y(\|f_{xx}\|_\infty \frac{h_x^2}{24} + \|f_{yy}\|_\infty \frac{h_y^2}{24} + \|f_{xy}\|_\infty \frac{h_x h_y}{16}),$$

where the subscript letters behind the function represent its partial derivative.
Proof. The proof is given in the source [Büsing, 2008].

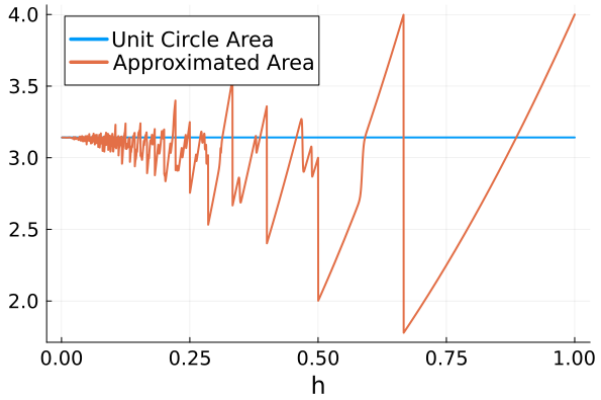
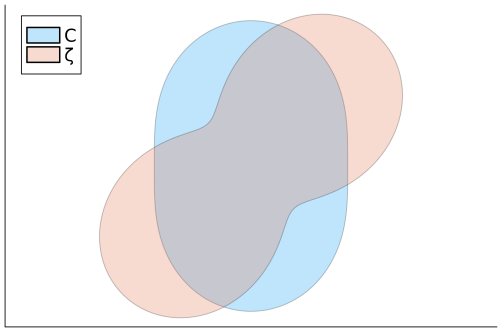


Figure 7: A plot of π (the real area of the unit circle) and the approximated area through the summed up midpoint rule dependent on the mesh step size h.

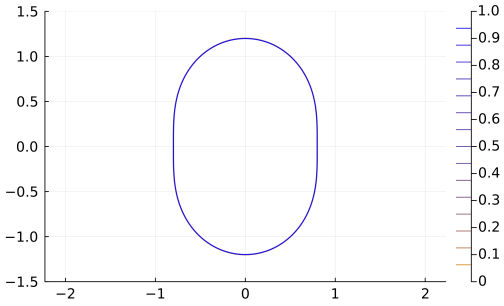
If we take a look at Definition 2.7, we can see that the smooth indicator function is in $C^2(R)$ if and only if the radius function of the cell is in $C^2(R)$, since all the other functions that occur in this definition are infinitely differentiable. Thus, the summed up midpoint rule is numerically convergent if the radius function is twice continuously differentiable over R.
Figure 7 pictures the error of the computation of the area of the unit circle using the summed up midpoint rule dependent on h. One can see that the error vanishes as $h \to 0$.
On this foundation we are also able to compute the overlap of two cells. As mentioned earlier, we can simply use the approximation

$$A_{C \cap C_2} \approx \int_{R_{overlap}} \psi_C(\vec{x})\psi_\zeta(\vec{x})d\vec{x}.$$

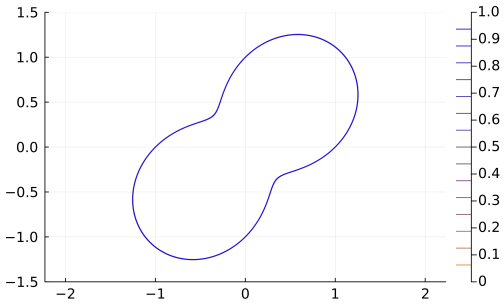
In the implementation of the overlap computation, we operate quite similarly to the computation of the area of a single cell. First, we determine the rectangles R_C and R_ζ which overlap their cells. Then, we choose the intersection $R_{overlap} := R_C \cap R_\zeta$ to be our operating domain. If the intersection of R_C and R_ζ is the empty set ,we can finish the computation and say that the overlap is zero, as both cells will neither overlap.
Otherwise, we lay a mesh over $R_{overlap}$ and determine the midpoints $\vec{m}_{i,j}$ as in the computation before. From there on, we are ready to apply the summed midpoint rule to approximate the area of the overlap. Correspondent to the approximation from above, we set our function f from Proposition 2.9 to be $f = \psi_C\psi_\zeta$.
Figure 8 demonstrates how the function $\psi_C\psi_\zeta$ represents the overlap of C and ζ.
For now, these are all important functionalities that we need for our CRF model.



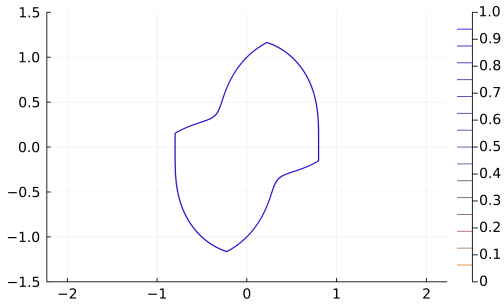
(a) First, the figure shows a plot in the plane of C in blue and ζ in red. The overlap of both cells is clearly noticeable.



(b) The next plot is a contour plot. It displays the function ψ_C.



(c) In the third sub figure, we can see the counterpart to (b). It shows a contour plot of ψ_ζ.



(d) A contour plot of $\psi_C\psi_\zeta$. One can notice the correspondence of this function to the overlap that is visible in (a).

Figure 8: The graphics show two possible CRF cells $C = ((0,0)^T, r_C(\phi) = \frac{1}{2}\sin(2\phi)+1)$ and $\zeta = ((0,0)^T, r_\zeta(\phi) = 1 - \frac{1}{5}\cos(2\phi))$, their smooth indicator functions ψ_C and ψ_ζ with $s = 500$ and demonstrate how the product $\psi_C\psi_\zeta$ represents the the overlap of the two cells. The axis show the spatial coordinates.

We are now able to measure a cell's area and whether or how much it interferes with other cells.

2.4 Implementation of the discrete overlap calculation

The last subsection of Chapter 2 deals with the problem of calculating the overlap of discrete cells. As we have already seen in previous parts, we must therefor determine all overlap areas. They can be represented as discrete cells by themselves. Having obtained all such overlapping cells, it is straightforward to compute their area with the shoelace formula from Proposition 2.8.
For the rest of the paragraph, we set the vertex $\vec{x}_{N+1} := \vec{x}_1$. The case in which one cell completely surrounds the other is assumed to be excluded.
In spite of the solution displaying to be obvious, the final implementation turns out to be quite effortful. First of all, let us formulate the problem precisely.

Problem 2.11.
For two cells given in their DF $C = (\vec{c}_1, \ldots, \vec{c}_N)$ and $\zeta = (\vec{\zeta}_1, \ldots, \vec{\zeta}_M)$, identify all discrete cells $O_1, \ldots, O_L$ such that any region where C intersects ζ is represented by exactly one O_l , $(1 \leqslant l \leqslant L)$, i.e.

$$C_i \cap C_j = \amalg_{l=1}^{L} O_l,$$

where $\amalg$ *symbolises a disjointed union and L denotes the number of overlaps that occur between C_i and C_j.*

In order to solve this problem, we first define the domain that will be operated on. Similarly to the continuous area computation, we want to approximate the important location through a rectangle in the following way.
For each of the two discrete cells C and ζ, iterate through all vertices and save those coordinates that are the farthest left, right, up and down. As a result, we obtain two rectangles $R_C \supseteq C$ and $R_\zeta \supseteq \zeta$ which completely cover their corresponding cell. As every overlap is located in both cells and thus in both rectangles, we then set our operation domain as the intersection

$$R_{overlap} = R_C \cap R_\zeta.$$

It is simple to control whether R_C and R_ζ intersect. Therefore, the maximum of the smaller x coordinates of both rectangles must be less than or equal to the minimum of the larger x coordinates of both rectangles. The same must also hold for the y coordinates. The corners of $R_{overlap}$ are then given by these maximum and minimum values. This yields

$$\begin{aligned} a = max(R_C^{x_{min}}, R_\zeta^{x_{min}}), \qquad & b = min(R_C^{x_{max}}, R_\zeta^{x_{max}}), \\ c = max(R_C^{y_{min}}, R_\zeta^{y_{min}}), \qquad & d = min(R_C^{y_{max}}, R_\zeta^{y_{max}}), \\ a > b \vee c > d \implies & R_{overlap} = \varnothing, \\ a \leqslant b \wedge c \leqslant d \implies & R_{overlap} = [a, b] \times [c, d]. \end{aligned}$$

An example of this computation can be seen in Figure 9. If performing the intersection delivers an empty set, we can end the calculation, since **no overlap** is possible in that case. So let us assume that $R_{overlap} \neq \varnothing$ from now on.
Now, we want to find every intersection point of the two cells. For the following, it is very useful to introduce some theory about our edges.

Definition 2.12. Edge parameterisation
The edge e_i of the cell $C = (\vec{x}_1, \ldots, \vec{x}_N)$ is the edge that connects the vertices $\vec{x}_i = (x_i, y_i)$ and $\vec{x}_{i+1}$.
The smallest and biggest x values of e_i are called x_i^{min} and x_i^{max}. Analogously, y_i^{min} and y_i^{max} represent the lowest and highest y values of e_i. They are called boundary coordinates.
An edge e_i is called parameterisable if the path $p_i : [x_i^{min}, x_i^{max}] \to \mathbb{R}^2$ that corresponds to e_i can be expressed as

$$p_i(x) = \begin{pmatrix} x \\ m_i x + a_i \end{pmatrix},$$

for some $m_i, a_i \in \mathbb{R}$.
Thus, all edges that are not parameterisable run completely vertically.

Since every edge is just a line, it is no problem to determine x_i^{min}, x_i^{max}, y_i^{min} and y_i^{max}.
If e_i is a not parameterisable edge, we know that it runs parallel to the y axis. In that case, it is clear that

$$\begin{aligned} x_i^{min} = x_i^{max} &= x_i = x_{i+1}, \\ y_i^{min} &= \min(y_i, y_{i+1}), \\ y_i^{max} &= \max(y_i, y_{i+1}). \end{aligned}$$

So let us now assume that e_i is a parameterisable edge. This means that there exist real numbers $m_i, a_i \in \mathbb{R}$ such that every y coordinate of each point $\vec{x} = (x, y)^T$ on e_i can be expressed as $y = m_i x + a_i$ for a $x \in [x_i^{min}, x_i^{max}]$. On this interval the edge is related to an affine linear function.
We can directly save x_i^{min} and x_i^{max} as the minimum and maximum values of x_i and x_{i+1}. Notice, that an affine linear functions on a compact interval always reaches its maximum and minimum values at the interval ends. Since we have got a parameterisable edge, the y values are given by an affine linear function $y = m_i x + a_i$. Thus, the extreme value are given at the interval ends $y_i = m_i x_i + a_i$ and $y_{i+1} = m_i x_{i+1} + a_i$. Consequently, the boundary coordinates of any edge are given by

$$\begin{aligned} x_i^{min} &= \min(x_i, x_{i+1}), & x_i^{max} &= \max(x_i, x_{i+1}), \\ y_i^{min} &= \min(y_i, y_{i+1}), & y_i^{max} &= \max(y_i, y_{i+1}). \end{aligned}$$

The determination of m_i and a_i of a parameterisable edge is a little bit more difficult, but can still be given through explicit formulas.

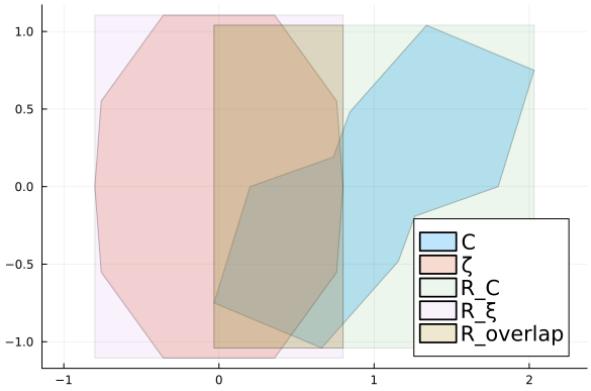


Figure 9: A sample of two discrete cells C (blue) and ζ (red), their rectangles R_C (green) and R_ζ (pink) and the intersection $R_{overlap}$ (brown) in the two dimensional space.

Proposition 2.13. ***Computation of*** m_i ***and*** a_i
Let e_i *be a parameterisable edge. Then, the parameters*

$$m_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}, \qquad a_i = \frac{x_{i+1}y_i - x_i y_{i+1}}{x_{i+1} - x_i},$$

deliver the edge parameterisation from Definition 2.12.
Proof.
The function $f(x) = m_i x + a_i$ *is uniquely defined by the equations*

(i) $f(x_i) = m_i x_i + a_i = y_i$,

(ii) $f(x_{i+1}) = m_i x_{i+1} + a_i = y_{i+1}$.

Subtracting (i) from (ii) and rearranging yields

$$m_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}.$$

Substituting this result into (i) and rearranging to a_i *provides*

$$a_i = y_i - x_i \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = \frac{x_{i+1}y_i - x_i y_{i+1}}{x_{i+1} - x_i}.$$

□

In order to reduce the computational effort, we do not check every edge for an intersection, but only those that could lie in the critical area $R_{overlap}$. We call these edges critical.

Definition 2.14. Critical edge
An edge $e_i = \overrightarrow{c_i\, c_{i+1}}$ of a discrete cell $C = (\vec{c}_1, \ldots, \vec{c}_N)$ is said to be critical for a rectangle $R_{overlap}$ if it could intersects with $R_{overlap}$, i.e.

$$[x_i^{min}, x_i^{max}] \times [y_i^{min}, y_i^{max}] \cap R_{overlap} \neq \varnothing.$$

All edges without this characteristic are called not critical. An illustration of critical edges can be found in Figure 10.

Whether the rectangles from the definition intersect or not can be decided in the same way as when calculating $R_{overlap}$.
In order to get ahead in the solution of Problem 2.11, we must calculate all edge parameters of all edges in both DF cells C and ζ. Then, we must determine which edges are critical. Next, we want to find all intersection points of C and ζ. We save all critical edges and their parameterisations in two sets, one for each cell.
For any not critical edge, we can say with certainty that it does not cross or lay inside of $R_{overlap}$ and therefore has no intersection from C and ζ on it. There

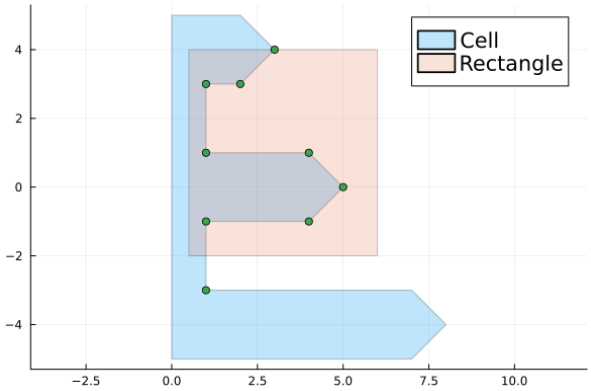


Figure 10: This diagram shows an example cell (blue), rectangle (red) and the critical edges (green dots) that occur in this scenario. If one traverses vertices counterclockwise, then each dot represents the edge coming afterwards. Every green vertex stands for a critical edge. If a vertex has no dot, the according edge is not critical.

could be critical edges that do not intersect $R_{overlap}$ and that thus, do not have an intersection on them as well. However, a further case distinction to decide whether an edge actually intersects with $R_{overlap}$ would be too expensive to serve the purpose of limiting the computational effort.
In order to find every intersection from C and ζ, every critical edge in C gets compared with all the critical edges in ζ. The following case distinction always indicates whether two edges intersect or not. The cases should be examined in the exact order now given. Let e_i be a critical edge in C and ξ_j be a critical edge in ζ.
Case 1: The edges are not nearby.
Similar to the intersection of the rectangles in Definition 2.14, we first want to check whether both edges are close to each other. Therefor, we compute

$$R_{intersection} := [x_i^{min}, x_i^{max}] \times [y_i^{min}, y_i^{max}] \cap [x_j^{min}, x_j^{max}] \times [y_j^{min}, y_j^{max}],$$

where the boundary coordinates with a subscript i belong to e_i and the ones with a subscript j belong to ξ_j.
If this intersection is equal to the empty set, we can end the computation with the result **no intersection**. Otherwise, we want to save $R_{intersection}$. To reduce the notation effort, we rename the rectangle

$$R_{intersection} = [a, b] \times [c, d] := [x_i^{min}, x_i^{max}] \times [y_i^{min}, y_i^{max}] \cap [x_j^{min}, x_j^{max}] \times [y_j^{min}, y_j^{max}],$$

and proceed with the next case.
Case 2: Both edges are not parameterisable.
If e_i and ξ_j are not parameterisable, the edges will **intersect** on the whole line $\{x_i\} \times [c, d]$. Thus, for every $y \in [c, d]$ the point $(x_i, y)^T$ will be an **intersection** of e_i and ξ_j.
Case 3: Exactly one edge is not parameterisable.
When only one edge, let us say e_i, is not parameterisable, we must use the parameterisation of the other edge to compute $s = (x_i, m_j x_i + a_j)^T$. If the y coordinate

$m_j x_i + a_j \in [c, d]$, we have found the **intersection** s, but if this is not true, there is **no intersection** between both edges.
Case 4: Both edges are parameterisable and $m_i = m_j$.
Whenever $m_i = m_j$, we must check if $a_i = a_j$. If yes, we again have two edges that intersect on the whole interval $[a, b]$, i.e. one could return the **intersection** $(x, m_i x + a_i)^T$ for any $x \in [a, b]$. If $a_i \neq a_j$, both edges do **not intersect**, because they are parallel, but not the same.
Case 5: Both edges are parameterisable and $m_i \neq m_j$.
Having reached the final case, we examine lines that always intersect at a $t \in \mathbb{R}$. This t is given by the formula

$$m_i t + a_i = m_j t + a_j \iff t = \frac{a_j - a_i}{m_i - m_j}.$$

In this case, e_i and ξ_j intersect if and only if $t \in [a, b]$. Then, the **intersection** is given by the point $(t, m_i t + a_i)^T$.
Otherwise, there is **no intersection** between e_i and ξ_j.

With the help of this case distinction, we are able to iterate through all critical edges in C and ζ to find all intersections that occur between these cells. We will call the set of all intersections I. Having obtained I, we are one step further in our computation.
If one takes a look at Figure 11 at the end of this chapter, one can see that each intersection changes whether the overlap cell uses edges from C or edges from ζ. This investigation is the key idea of the algorithm that finally computes the overlap cells. It looks as follows.

Algorithm 2.15. ***Computation of a discrete overlaps***

INPUT:

- *Discrete cells C and ζ*
- *List I of unused intersections of C and ζ*

```
function constructOverlap(C, ζ, I)
    usedIntersections = List{Intersection}(I[1])
    newOverlap = List{Vertices}(I[1])
    currentIntersection = I[1]
    for counter = 1 : length(I) do
        if counter is even then
            newPath, newIntersection = findPath(currentIntersection, C, I)
        else
            newPath, newIntersection = findPath(currentIntersection, ζ, I)
        end if
        append!(newOverlap, newPath)
        if newIntersection == I[1] then
            return newOverlap, usedIntersections
```

```
        else
            append!(newOverlap, newIntersection)
            append!(usedIntersections, newIntersection)
            currentIntersection = newIntersection
        end if
    end for
end function
```

OUTPUT:

- *A single intersection 'newOverlap' which occurs between C and ζ and which uses vertices from C and ζ as well as only intersections from I*
- *A list 'usedIntersections' of all intersection that are used in 'newOverlap'*

The algorithm takes the first intersection $I[1]$ in I as the starting vertex of the overlap cell 'newOverlap'. $I[1]$ is also added initially to the list 'usedIntersections'. Next, the function 'getOverlap' uses another function called 'findPath' which delivers the path on the discrete cell ζ from the starting intersection to the next intersection in I that appears when traversing through the edges of ζ. This new intersection will also be returned. The just mentioned path is a list of vertices in ζ that exactly occur between both intersections. It may be empty if the next intersection is already found on the same edge where the current intersection is located. The new path and the new intersection will then get appended to 'newOverlap'. The new intersection will also get added to the list 'usedIntersections'.
Since each intersection implies changing the cell from which the overlapping cell uses the edges, 'findPath' is now applied to the other cell. Again, it will deliver the next intersection as well as a list of the in between laying vertices. The vertex list always gets appended to 'newOverlap'.
If the new intersection is equal to the first intersection $I[1]$, we have already constructed the final discrete cell 'newOverlap'. Then, 'newOverlap' and 'usedIntersections' can get returned by 'constructOverlap'.
Otherwise, we can add the new intersection to 'newOverlap' and 'usedIntersections' and then, we will have to call 'findPath' on the other discrete cell once more and repeat the last part of the algorithm, until we find our starting intersection.
Once we are able to extract an overlap from C and ζ, we can delete all used intersections from I, because every intersection point is part of exactly one overlap. While I is not empty, we can make a new call to the function 'constructOverlap' with the adjusted I, to obtain another overlap. When I is empty, we have surely found every intersection of C and ζ and thus, have solved Problem 2.11.

Every time 'findPath' gets called, it is questionable in which direction the function should traverse through the vertices of the given cell. But it can be tested in the succeeding way.
Starting from the current intersection that is passed into the function, go a little bit in one direction on the edge of the passed cell where the intersection is located. For the then obtained point, examine whether it is located in the other cell, too. If the

point is located on both cells, we know that the chosen direction is the right one. Otherwise, we know that we must use the other direction.
A simple way to investigate whether a point is located in a polygon, is to draw a straight line from the point to the outside of the polygon. Then, the number of intersections of the drawn line and the wall of the discrete cell determines whether the point is inside the polygon or not. The point is located in the inside of the polygon if and only if the number of intersections is odd.

There are some requirements that C and ζ must fulfil in order for the Algorithm 2.15 to work properly.
First, the algorithm assumes that each intersection is exactly used one time by exactly one discrete cell overlap. If both cells share a vertex, it can happen that this specific vertex, which will then also be an intersection, is used in two different overlaps. Then, our algorithm will fail, because this intersection point will be deleted after the calculation of the first overlap and accordingly, will be absent from the calculation of the second overlap that uses it.
Second, the algorithm cannot handle the situation where two edges partly run on each other. This case can require different amounts of intersections to deal with it. The necessary distinctions are not considered in the presented algorithm.
In order for the Algorithm 2.15 to still work, we add the following procedure before we apply the algorithm: As long as one of the problem cases described occurs, move the whole discrete cell C slightly up and right. This will change the overlap area just a little bit, but causes the algorithm to work properly.
Finally, we have now fully derived a scheme which is able to compute all overlapping areas between two discrete cells. Thus, we have a solution to Problem 2.11. Figure 11 shows some scenarios of two DF cells and the overlap that is computed by the described algorithm.
The overlap cells determined by the algorithm are used for the overlap force in Chapter 4.4. There, a gradient for the area of the overlap cell gets computed. In order for this calculation not to produce an incorrect sign in the computation of the gradient and also in order to meet the requirement of a positively orientated cell for the shoelace formula, it is necessary that the cell is in a positive orientation, i.e. that the vertices are arranged counterclockwise. As [Bonvallet, 2009] writes, the orientation of a polygon is positive if

$$\sum_{n=1}^{N} (x_{n+1} - x_n)(y_{n+1} + y_n) < 0,$$

where N denotes the number of vertices of the polygon, $\vec{x}_n := (x_n, y_n)^T$ denotes the nth vertex of the polygon and $\vec{x}_{N+1} := \vec{x}_1$. After each overlap cell computation, we apply this formula to find out, whether the determined cell has a positive orientation, or not. If the orientation is negative, we reverse the order of the vertices so that the cell ends up positively oriented.
This method is designed to deal with all kinds of DF cells, also with all varieties of special cases, that may lead to problems in simpler algorithms. The associated price is that the computational effort is quite high, especially when the number of cell vertices increases.

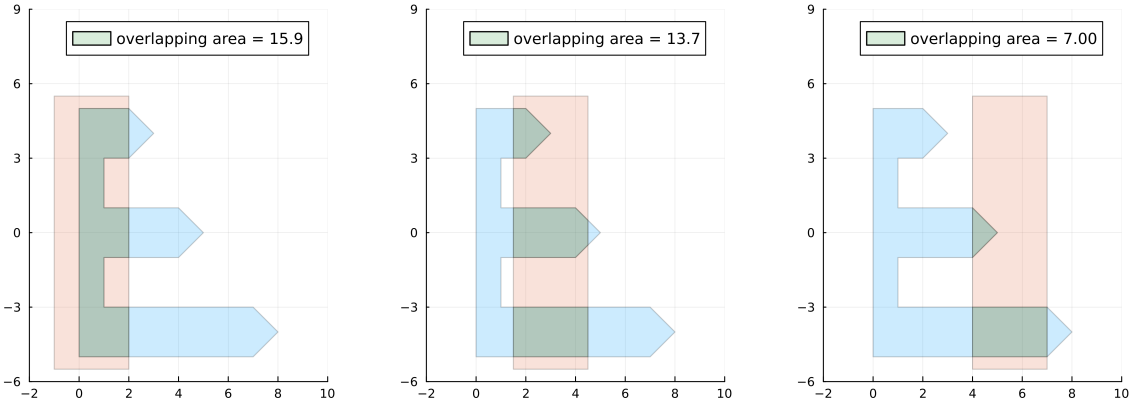


Figure 11: This figure shows 3 plots of 2 DF cells, shown in blue and red, each. One can see the calculated overlap in a green colour and the calculated area of the overlap is shown at the top of the plots. The red cell is shifted to the right in each subsequent plot, resulting in a change of the occurring overlap.

In order to optimize the computational effort, it may be useful to meet further requirements to the discrete cells, like for example convexity. An attempt to do so can be found in the Appendix.

So far, we have introduced two different models for cells with the CRF and DF model. We have also managed to develop methods to compute the area of a cell as well as the overlap of two cells of the same model.
For the rest of the thesis we will focus on giving the cells dynamics. Therefor, we will derive different kinds of differential equations that fulfil different purposes.

3 CRF shape recovery model

Having accomplished the development of the two cell models CRF and DF, we will now start the modelling of some cell dynamics.
The very first problem, that we want to regard, looks the following. We are considering one cell C that is given in the CRF $C = (\vec{c}, r)$. In order to give C the possibility to change in time, we make the centre point and radius function dependent on time. Thus, the shape of C at time $t \in [0, \infty)$ is represented by the CRF $C(t) = (\vec{c}(t), r(\phi, t))$. In order to keep the simplicity of ordinary differential equations, every angle ϕ gets assigned to its own radius function $r_\phi(t)$ such that for all $t \in [0, \infty)$ and $\phi \in [0, 2\pi)$ the equation $r(\phi, t) = r_\phi(t)$ holds. In order for the analysis to work, we assume $r_\phi \in C^1(\mathbb{R}, \mathbb{R}_{>0})$.
There are two specified states in the shape recovery model. The first state is the initial state which is expressed with the CRF $C^{(0)} = (\vec{c}^{(0)}, r^{(0)})$. Analogously, the second one is the desired state, described by $C^{(1)} = (\vec{c}^{(1)}, r^{(1)})$. To solve our problem, we need to find a model that takes the initial condition $C^{(0)}$ and makes it transform into the desired state $C^{(1)}$, as time goes by.
We set

$$\begin{cases} \vec{c}(0) = \vec{c}^{(0)}, \\ r_\phi(0) = r_\phi^{(0)}, \quad \forall \phi \in [0, 2\pi), \end{cases}$$

to hold our initial data.
After all necessary prerequisites are met, we are ready to introduce the first model provided by three ODEs.

Model 3.1. ***Shape recovery model***
With the data $C^{(0)} = (\vec{c}^{(0)}, r_\phi^{(0)})$ and $C^{(1)} = (\vec{c}^{(1)}, r_\phi^{(1)})$ from above, the shape recovery model is described by the initial value problems (IVPs)

$$\begin{cases} c_x'(t) = c_x^{(1)} - c_x(t), \\ c_x(0) = c_x^{(0)}, \end{cases}$$

$$\begin{cases} c_y'(t) = c_y^{(1)} - c_y(t), \\ c_y(0) = c_y^{(0)}, \end{cases}$$

$$\begin{cases} r_\phi'(t) = r_\phi^{(1)} - r_\phi(t), \\ r_\phi(0) = r_\phi^{(0)}, \end{cases}$$

where $\vec{c} = (c_x, c_y)^T$.

Note, that the preferred states, which are marked with a superscript (1), are just time independent real constants. It is easy to see that all individual IVPs have the same structure

$$\begin{cases} x'(t) = a - x(t), \\ x(0) = b, \end{cases} \tag{6}$$

with $a, b, t \in \mathbb{R}$ and $x \in C^1(\mathbb{R}, \mathbb{R})$. This IVP will now be referenced with (6).
Hence, the IVPs in the model will have the same analytical characteristics as (6). This is why we will now do the analysis on IVP (6).

Proposition 3.2. ***Solution of the IVP*** (6)
The unique global solution of the IVP (6) *is given by*

$$x(t) = (b - a)e^{-t} + a.$$

Proof.
At the beginning, we want to ensure the existence of a locally unique solution of (6) *that would be given if the Picard Lindelöf theorem could be applied. Therefor the ODE's rigth hand side $f(t, x) = a - x$ must be continuous in t and Lipschitz continuous in x.*
As a polynomial of t and x, f is in particular continuous in t and x. Direct application of the function yields

$$|f(t, x_1) - f(t, x_2)| = |a - x_1 - (a - x_2)| = |x_1 - x_2|, \ \forall x_1, x_2, t \in \mathbb{R}.$$

Hence, f is Lipschitz continuous in x with constant $L = 1$.
Now, with both conditions met, we can conclude that the Picard Lindelöf theorem can be applied to the IVP associated with the ODE $x'(t) = a - x(t)$ with the initial condition $x(0) = x_0 = b$. This means that there exists a unique local solution in some neighbourhood of $t_0 = 0$.
We now want to determine this solution. Therefor, let us first assume that $a = b$. In that case the right hand side will be zero at $t_0 = 0$

$$f(t_0, x_0) = a - x(0) = a - b = a - a = 0.$$

Consequently, the initial condition correspond to a steady state and the solution is given by the constant function $x(t) = a = b$.
Now, we would like to look at the remaining case $a \neq b$. The right hand side is then initially unequal to zero

$$f(t_0, x_0) = a - b \neq 0.$$

Since f is continuous, we can find a small environment $I = [0, \epsilon]$, $0 < \epsilon \ll 1$, of t_0 for which we can say that

$$\forall t \in I : f(t, x) < 0 \text{ or } f(t, x) > 0.$$

Under these assumptions, the solution can be determined by separating the variables for $t \in I$.

$$x'(t) = a - x(t) \iff \int_0^t \frac{x'(s)}{a - x(s)}\, ds = \int_0^t 1\, ds = t$$

$\iff \int_{x_0}^{x(t)} \frac{1}{a-v}\, dv = t$ *(Substitute $v = x(s)$)*

$\iff \left[-\ln(|a-v|)\right]_{v=x_0}^{x(t)} = t$

$\iff \ln(\frac{a-x_0}{a-x(t)}) = t$ *(The condition from before the calculation ensures that the numerator and denominator have the same sign.)*

$\iff \frac{a-b}{a-x(t)} = e^t$

$\iff x(t) = (b-a)e^{-t} + a$

For $a = b$, we regain our first solution $x(t) = a = b$.
All in all, we can conclude that $x(t) = (b-a)e^{-t} + a$ is the locally unique solution of (6).
It is even a global solution, because it fulfils the ODE for any $t \geqslant 0$. Since f is continuously differentiable on $\mathbb{R}^2$, it is the unique maximal solution.

□

So the IVP (6) has an explicit solution, as does every IVP from the shape recovery model.

Corollary 3.3. ***Explicit solution to the shape recovery model***
For $C^{(0)} = (\vec{c}^{(0)}, r_\phi^{(0)})$ and $C^{(1)} = (\vec{c}^{(1)}, r_\phi^{(1)})$ given, the explicit solution to the shape recovery model is given by the functions

$$\begin{aligned} c_x(t) &= (c_x^{(0)} - c_x^{(1)})e^{-t} + c_x^{(1)}, \\ c_y(t) &= (c_y^{(0)} - c_y^{(1)})e^{-t} + c_y^{(1)}, \\ r_\phi(t) &= (r_\phi^{(0)} - r_\phi^{(1)})e^{-t} + r_\phi^{(1)}. \end{aligned}$$

So, we do not need a numerical implementation to approximate the solution. Figure 12 shows some shape developments for different initial and desired states.

Proposition 3.4. ***Steady states of IVP*** (6)
The only steady state of (6) *is given by $x^* = a$. It is an asymptotically stable steady state.*
Proof.
The right hand side $f(x) = a - x$ is equal to zero if and only if $x^* = a$. To proof that this state is asymptotically stable, we will use the theorem of linearised stability. Therefor, we must compute

$$f'(x) = -1 \implies f'(x^*) = -1 < 0.$$

Since the value of x^* in f' is less than zero, linearised stability yields that $x^* = a$ is an asymptotically stable steady state.

□

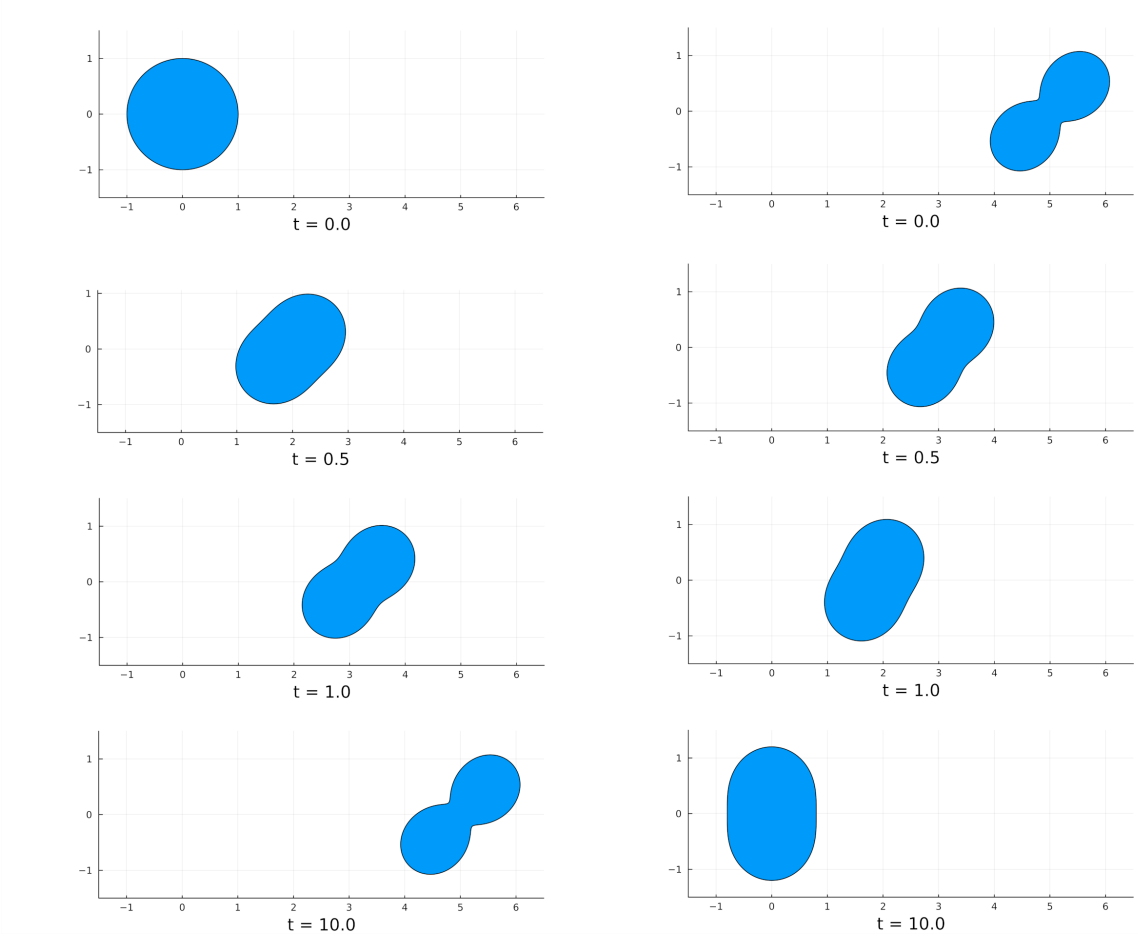


Figure 12: Here, we can see two different cell developments associated with the shape recovery model.
In the left column the initial state is given by the CRF cell $C = ((0,0)^T, \phi \mapsto 1)$ and the desired state is $D = ((5,0)^T, \phi \mapsto \frac{1}{2}\sin(2\phi) + \frac{4}{5})$.
The right column shows another transformation. The initial state is given by D which is the desired state in the first column. The desired state in the right column is defined by the CRF $E = ((0,0)^T, \phi \mapsto 1 - \frac{1}{5}\cos(2\phi))$.

It can be deduced from this that in the shape recovery model, the desired state $C^{(1)}$ will always be reached, as $t \to \infty$, because a is here always associated with the according desired state. This is the first dynamic model. It is the only one in this thesis that is explicitly solved. The following systems become more complex and can no longer be solved explicitly. One advantage of an explicit solution is that one can calculate the solution straight away at any wanted time. Furthermore, no numerical approximation of the solution is required, which prevents many possible errors.

4 Energy based dynamics for the DF model

In the now following models, we want to take a look at a different approach. The next models are based on the application of energy functionalities on vertex models, similarly to a model that is explained in the paper [Fletcher et al., 2014]. We consider a set of M DF cells, each with N vertices. The vector $\vec{C} = (C_1, \dots, C_M)^T$ holds all cells in the system where $C_i = (\vec{x}_{i1}, \dots, \vec{x}_{iN})^T$ describes the vector of all vertices of the ith cell, $(1 \leqslant i \leqslant M)$. As previously assumed, $\vec{x}_{N+1} := \vec{x}_1$.
An **energy function** is a mathematical function used in various fields, including physics, chemistry or material science. Its primary purpose in this thesis is to describe the potential energy associated with a single DF cell or a system of DF cells. We use energy functions to measure whether certain cell parameters, like area, edge lengths or interior angles have the right value. Each coming energy is identified by a capital Latin letter.
Once we have found an appropriate energy, we can then directly conclude a force which can get applied on the cells' vertices.

Definition 4.1. Force function
For a given energy E_i of the ith cell, the force $F_j^{(E_i)}$ that describes the impact of E_i on the jth vertex of the ith cell is given by

$$F_j^{(E_i)}(\vec{C}) := -\alpha_{E_i}(\vec{C}) \nabla_{\vec{x}_{ij}} E_i(\vec{C}),$$

where $\alpha_{E_i} > 0$ denotes a positive scaling factor and $\nabla_{\vec{x}_{ij}}$ is the gradient for the jth vertex of cell i.

Remark 4.2.
Our energy functions have the structure $E = |$ desired state - current state $|$. For every coming force, we choose the value of the energy function as the scaling factor such that a high energy results in a stronger force.
Another possible approach could be to write the energy functions in another form $E = \frac{1}{2}|$ desired state - current state $|^2$. Then, one would just have to compute the negative signed derivation of E in order to get the same scaled force function as in the first approach, because of the chain rule.
In both cases the energy E would not be truly differentiable, because of the absolute value. However, we will use the derivative of E to calculate the forces anyway, since the points at which E is not differentiable form a null set.

The application of these forces in a differential equation results in a so called gradient flow. This will lead to a deformation of the cell in such a way that the energy gets minimized. This corresponds to one of the most basic physical laws: For a closed system, with constant external parameters, the internal energy will decrease and approach a minimum value at equilibrium. To be able to implement the coming energies and forces, we must introduce the cells desired states. As the name suggests, the desired states are representative of the shape and size that a cell wants to develop

into. The DF $C_i^{(d)}$ is the desired state of the ith cell in the system. In the following, we will derive different energy functions that will help our cell to develop to a desired cell state. To approximate the solutions to the differential equations, I used an Explicit Euler integrated in Julia, or an Euler Maruyama Scheme with a fixed step size of $\Delta t = 2^{-8}$. Thus, one time step in the coming figures is equivalent to a value of 2^{-8}.
In some plots, one can see arrows reaching from the cell vertices into the domain. These arrows are exactly the from the algorithm computed forces that act at the specific vertex at that time in the simulation.

4.1 Area energy

The first energy which we want to use is the area energy. Other forces, that will get introduced afterwards, will usually lead to a reduction of the area. We want to ensure that the cell area remains almost constant over time.
We assume that all considered polygons have a positive orientation so that the shoelace formula can be applied. This is not a hard restriction, because all DF cells are positively orientated initially and the change from a negative orientated polygon into a positive orientated polygon is quite simple. The progress is already described at the end of Section 2.4. Fortunately, we have already derived a method to compute the current area of a positively orientated polygon in Section 2. Since every DF cell is also a polygon, we can just apply the shoelace formula to obtain

$$a(C_i(t)) := \frac{1}{2}\sum_{j=1}^{N}(x_{i,j}(t)y_{i,j+1}(t) - x_{i,j+1}(t)y_{i,j}(t)), \tag{7}$$

with $a \geqslant 0$ being the functional that maps a cell C_i to its area and $\vec{x}_{i,j} = (x_{i,j}, y_{i,j})^T$ being the jth vertex of C_i. Since we want all of our cells to reach their desired states $C_i^{(d)}$, the desired cell area must be equal to the areas of the desired states.
Thus, we can save them as a vector

$$\vec{a}^{(d)} := (a_1^{(d)}, \dots, a_M^{(d)})^T,$$

where $a_i^{(d)} = a(C_i^{(d)})$ denotes the area of $C_i^{(d)}$, $(1 \leqslant i \leqslant M)$.

Definition 4.3. Area energy
The energy A_i, used to keep the cell i at a constant volume, reads

$$A_i(C_i) := |a_i^{(d)} - a(C_i)|. \tag{8}$$

This energy reaches its minimum value whenever $a_i^{(d)} = a(C_i)$. In order to derive the gradient flow for this energy, we must first compute its gradient.

Proposition 4.4. ***Area force***
The gradient of A_i with respect to the jth vertex of cell i is given by

$$\nabla_{\vec{x}_j} A_i(C_i) = \operatorname{sgn}(a(C_i) - a_i^{(d)}) \frac{1}{2} \begin{pmatrix} y_{i,j+1} - y_{i,j-1} \\ x_{i,j-1} - x_{i,j+1} \end{pmatrix}.$$

As a scaling factor, we choose

$$\alpha_{A_i}(C_i) := |a_i^{(d)} - a(C_i)|.$$

Thus, the area force reads

$$F_j^{(A_i)}(C_i) = \frac{1}{2}(a_i^{(d)} - a(C_i)) \begin{pmatrix} y_{i,j+1} - y_{i,j-1} \\ x_{i,j-1} - x_{i,j+1} \end{pmatrix}. \tag{9}$$

Proof.
To reduce the notation effort, we neglect the subscript i, because we just consider a single cell. In order to compute $\nabla_{\vec{x}_j} |a^{(d)} - a(C)|$, let us first assume that $a^{(d)} \geqslant a(C)$. Then, one can calculate

$$\begin{aligned} \nabla_{\vec{x}_j} |a^{(d)} - a(C)| &= -\nabla_{\vec{x}_j} a(C) = -\tfrac{1}{2} \sum_{j=1}^{N} \nabla_{\vec{x}_j} (x_j y_{j+1} - x_{j+1} y_j) \\ &= -\tfrac{1}{2} \sum_{j=1}^{N} \begin{pmatrix} \partial_{x_j} (x_j y_{j+1} - x_{j+1} y_j) \\ \partial_{y_j} (x_j y_{j+1} - x_{j+1} y_j) \end{pmatrix} = -\frac{1}{2} \begin{pmatrix} y_{j+1} - y_{j-1} \\ x_{j-1} - x_{j+1} \end{pmatrix} \end{aligned}$$

Remember that $a^{(d)}$ is just an independent constant. In the other case, where $a^{(d)} < a(C(t))$, there is just a change in the sign. The combination of both cases yields the expression above.
Since $-|a^{(d)} - a(C)| \operatorname{sgn}(a(C) - a^{(d)}) = a^{(d)} - a(C)$, we can conclude the area force. □

For the computation of the gradient of the cell area, it is important that the cells has a positive orientation, i.e.the vertices must be ordered counterclockwise. For cells with a negative orientation, this formula will produce the wrong sign.
Having derived the area energy and force, we are ready to start the first simulations in order to see if the force does its job correctly. Therefor, we choose a setting with just one cell.
The initial condition is given by a regular hexagon. In the first simulation, the distances of each vertex to the centre will be 1 unit of length. This results the DF cell $C(0) = ((1,0),(0.5,0.87),(-0.5,0.87),(-1.0,0),(-0.5,-0.87),(0.5,-0.87))$ which can be seen in the first diagram of Figure 13 and in an initial cell area of $a(C(0)) = \frac{3\sqrt{3}}{2} \approx 2.6$. As a desired cell area, we choose $a^{(d)} = 10$. Having dropped the index i for the ease of notation, this results in the following ODE for each vertex.

Model 4.5. ***Area model***
The area model with $a^{(d)} = 10$ is given by the IVP

$$\begin{cases} \dfrac{d\vec{x}_j(t)}{dt} = F_j^{(A)}(C) = \frac{1}{2}(10 - a(C(t))) \begin{pmatrix} y_{j+1} - y_{j-1} \\ x_{j-1} - x_{j+1} \end{pmatrix}, \\ \vec{x}_j(0) = \vec{x}_j^{(0)}, \end{cases}$$

with $\vec{x}_j^{(0)}$ *being the* j*th vertex of* $C(0)$.

The result can be seen in Figure 13. Further models for different energies have the exact same structure and are therefor not written down explicitly. As we can see,

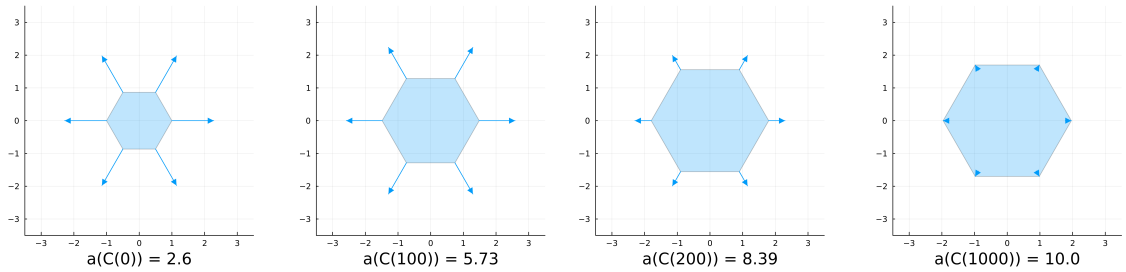


Figure 13: This figure shows the solution at the times $t \in \{0, 100, 200, 1000\}$ to the Model 4.5, which uses the area force to bring the cell area to the desired state $a(C) = 10$. The cell has an initial area of approximately 2.6. The area force causes the vertices to move away from the cell centre, causing an increase of the cell area. The areas at the times shown can be seen under each diagram. One can see from the length of the arrows that the closer the current area gets to the desired state, the weaker the forces become. This corresponds to the scaling factor $|a^{(d)} - a(C)|$. As soon as $a(C) = 10$, the forces stop acting and a steady state is reached.

the area force works right if the current area is less than the desired area. Figure 14 shows a similar simulation, but with the difference that the cell area is initially larger than the desired cell area. Also this case works.

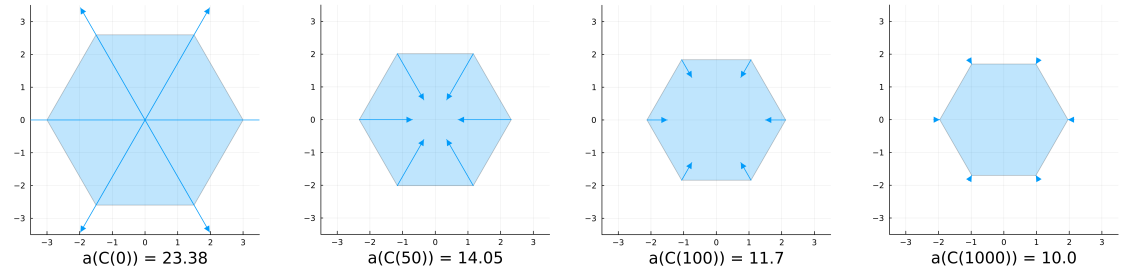


Figure 14: Similar to Figure 13, this image shows a cell that develops according to the area force. In contrast to the last illustration, we assume an initial area of approximately 23.38. That means that the area force must act in the reversed direction in order to let the cell develop into the desired state $a^{(d)} = 10$. As we can see in the four diagrams, this is exactly what happens.

4.2 Edge energy

We have already secured a constant cell area system. In the following two paragraphs, we want to focus on the cell developing into its desired shape. Our first step

to ensure this, is the edge energy. Remember, that the edge between the vertices j and $j+1$ is the jth edge. In a similar fashion to the area energy, we will use desired edge lengths $e_{ij}^{(d)} = \|\vec{x}_{i,j} - \vec{x}_{i,j+1}\|_2$. The functional that returns the edge length of edge j is given by

$$e_j(C) := \|\vec{x}_j - \vec{x}_{j+1}\|_2. \tag{10}$$

Having obtained this functional, we can introduce the edge energy.

Definition 4.6. Edge energy
The edge energy E_{ij} for the jth edge of the ith cell reads

$$E_{ij}(C_i) := |e_{i,j}^{(d)} - e_j(C_i)|. \tag{11}$$

The change of the position of the vertex j will change the lengths of both edges $j-1$ and j. This results in a change in both energies $E_{i,j-1}$ and Ei,j. Thus, we must consider these two energies in the edge force.

Proposition 4.7. ***Edge force***
The edge force is given by the formula

$$F_j^{(E_{ij})}(C_i) = \frac{e_{j-1} - e_{i,j-1}^{(d)}}{e_{j-1}(C_i)} \begin{pmatrix} x_{j-1} - x_j \\ y_{j-1} - y_j \end{pmatrix} + \frac{e_j - e_{i,j}^{(d)}}{e_j(C_i)} \begin{pmatrix} x_{j+1} - x_j \\ y_{j+1} - y_j \end{pmatrix}. \tag{12}$$

We choose $\alpha_{E_{ij}} := |e_{ij}^{(d)} - e_j(C_i)|$.
Proof.
Since this force acts on each cell individually, we can neglect the subscript i. *The searched term has the following structure*

$$F_j^{(E_j)}(C) = -\alpha_{E_{j-1}} \nabla_{\vec{x}_j} E_{j-1}(C) - \alpha_{E_j} \nabla_{\vec{x}_j} E_j(C),$$

with the scaling factors α_{E_j} *already defined in the proposition.*
The partial derivatives of the edge length e_j *are*

$$\partial_{x_j} e_j(C) = \partial_{x_j}((x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2)^{\frac{1}{2}} = \frac{x_j - x_{j+1}}{e_j(C)},$$
$$\partial_{y_j} e_j(C) = \frac{y_j - y_{j+1}}{e_j(C)}.$$

This yields

$$\nabla_{\vec{x}_j} E_j = sgn(e_j^{(d)} - e_j(C)) \nabla_{\vec{x}_j} - e_j(C) = sgn(e_j^{(d)} - e_j(C)) \frac{1}{e_j(C)} \begin{pmatrix} x_{j+1} - x_j \\ y_{j+1} - y_j \end{pmatrix},$$

and analogously

$$\nabla_{\vec{x}_j} E_{j-1} = sgn(e_{j-1}^{(d)} - e_{j-1}(C_i)) \frac{1}{e_{j-1}(C)} \begin{pmatrix} x_{j-1} - x_j \\ y_{j-1} - y_j \end{pmatrix}.$$

This yields the edge force

$$F_j^{(E_j)}(C) = \frac{e_{j-1}(C) - e_{j-1}^{(d)}}{e_{j-1}(C)} \begin{pmatrix} x_{j-1} - x_j \\ y_{j-1} - y_j \end{pmatrix} + \frac{e_j(C) - e_j^{(d)}}{e_j(C)} \begin{pmatrix} x_{j+1} - x_j \\ y_{j+1} - y_j \end{pmatrix}.$$

□

The application of this force is illustrated in Figure 15. For now, this is everything we want from the edge energy. We can conclude the edge model.

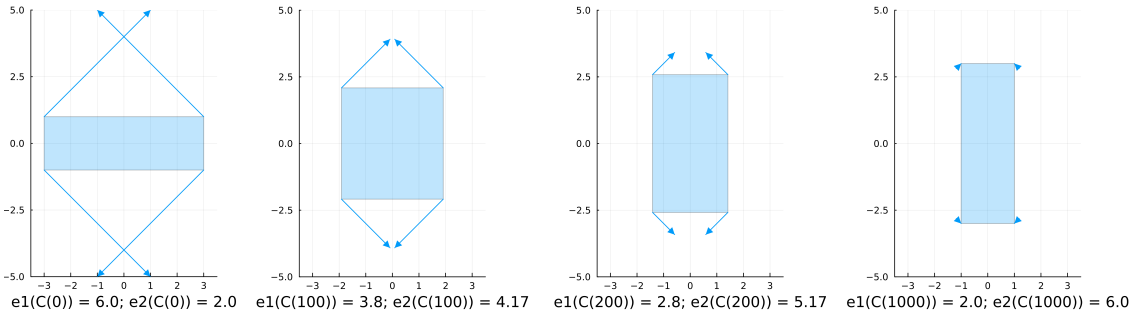


Figure 15: Here, we can see the edge force applied to a DF cell. At time $t = 0$ the cell equals a rectangle $[-3, 3] \times [-1, 1]$. Thus, the horizontal edges have a length of 6 and the vertical edges have a length of 2. These values at the according times $t \in \{0, 100, 200, 1000\}$ are written under the diagrams. The desired state is equal to the rectangle $[-1, 1] \times [-3, 3]$. That means that both horizontal edges must shrink and the vertical edges must expand. This behavior can be seen in the diagram sequence.

4.3 Interior angle energy

In order to complete the shape recovery in our model, we will also concentrate on the now coming interior angle energy. As the name already tells, it is used to ensure that the interior angles at each vertex evolve into their desired states. Simulations without this energy sometimes lead to constrictions at certain vertexes where the interior angle goes to 360°. The interior angle energy shall also prevent this problem. There are different ways to compute the angle that is given between two vectors. We choose the coming approach.
First of all, we want to be able to determine the angle θ between a given vector $\vec{v} = (v_1, v_2)^T$ and the x axis. In the first and fourth quadrant, we can easily apply the arctan function to compute

$$\theta = \arctan(\tfrac{v_2}{v_1}).$$

Unfortunately, it gets more complicated if $\vec{v}$ is located in the second or third quadrant. The usual arctan is not able to distinguish between these cases, because the fraction $\frac{v_2}{v_1}$ conceals the information of the signs of v_1 and v_2. However, there exists

an extension of the common arctan function. It takes v_1 and v_2 as arguments and is hence able to make a case distinction for the different signs of the arguments.

Definition 4.8. arctan2
The function

$$\text{arctan2} : \mathbb{R}^2/\{0\} \to (-\pi, \pi],\ (x, y) \mapsto \text{arctan2}(x, y)$$

computes the angle θ between $(x, y)^T$ and the x axis.

$$\text{arctan2}(x, y) = \begin{cases} \arctan(\frac{y}{x}) & x > 0 \\ \arctan(\frac{y}{x}) + \pi & x < 0, y > 0 \\ \arctan(\frac{y}{x}) - \pi & x < 0, y < 0 \\ \pi & x < 0, y = 0 \\ \frac{\pi}{2} & x = 0, y > 0 \\ -\frac{\pi}{2} & x = 0, y < 0 \end{cases}$$

This method provides high accuracy and robustness at all angles.
In order to obtain an interior angle at a certain vertex of a DF cell, we can use the interior angle functional.

Definition 4.9. Interior angle functional
The function $\iota_j(C_i)$ that computes the interior angle at vertex j of cell i is given by

$$\iota_j(C_i) := [\text{arctan2}(\vec{v_1}) - \text{arctan2}(\vec{v_2})]_{[0,2\pi)}, \tag{13}$$

where $\vec{v_1} := \vec{x}_{i,j-1} - \vec{x}_{i,j}$, $\vec{v_2} := \vec{x}_{i,j+1} - \vec{x}_{i,j}$ and $[\,\cdot\,]_{[0,2\pi)}$ describes the modulo operator, that maps the argument into the interval $[0, 2\pi)$ by adding or subtracting 2π correspondingly often.

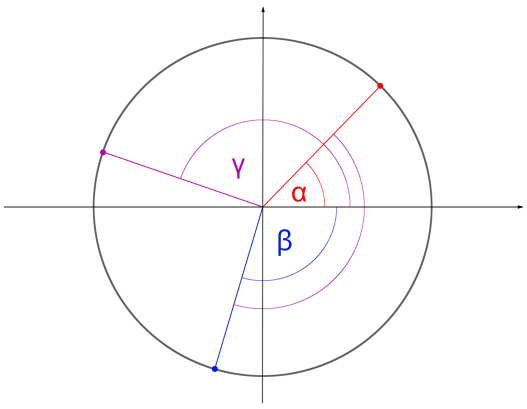


Figure 16: For two given vectors $\alpha \approx 45°$ and $\beta \approx -115°$, we can apply the formula from the interior angle functional to compute $\gamma = [45° - (-115°)]_{[0,2\pi)} = 160°$, what is also the angle between α and β.

Figure 16 illustrates the computation of the searched interior angle. In order to tell how each interior angle should evolve, we will again save the desired states $\iota_{ij}^{(d)} = \iota_j(C_i^{(d)})$.
Then, we can write down the interior angle energy.

Definition 4.10. Interior angle energy
The interior angle energy I_{ij} at vertex j of cell i is given by

$$I_{ij} := |\iota_{ij}^{(d)} - \iota_j(C_i)|. \tag{14}$$

Proposition 4.11. *Interior angle force*
The interior angle force is given by

$$F_j^{(I_{ij})}(C_i) = (\iota_{ij}^{(d)} - \iota_j(C_i)) \left(\frac{1}{\|\vec{v}_1\|_2^2} \begin{pmatrix} v_{1,y} \\ -v_{1,x} \end{pmatrix} + \frac{1}{\|\vec{v}_2\|_2^2} \begin{pmatrix} -v_{2,y} \\ v_{2,x} \end{pmatrix} \right),$$

where $\vec{v}_1 = (v_{1,x}, v_{1,y})^T := \vec{x}_{j-1} - \vec{x}_j$ *and* $\vec{v}_2 = (v_{2,x}, v_{2,y})^T := \vec{x}_{j+1} - \vec{x}_j$.
The scaling factor is defined as $\alpha_{I_{ij}} := |\iota_{ij}^{(d)} - \iota_j(C)|$.
Proof.
Again, we neglect the i, because we just consider one cell. The goal is to determine the interior angle force

$$F_j^{(I_j)}(C) = -|\iota_j^{(d)} - \iota_j(C)| \nabla_{\vec{x}_j} I_j(C)$$

Just like in the last forces, we use the sgn *function to get rid of the absolute value, yielding*

$$\nabla_{\vec{x}_j} I(C) = \operatorname{sgn}(\iota_j^{(d)} - \iota_j(C)) \nabla_{\vec{x}_j}(-\iota_j(C)),$$

since the desired state is just a constant number. Since we have a minus in front of the gradient at the end of the equation, the searched force can be written as

$$F_j^{(I_j)}(C) = (\iota_j^{(d)} - \iota_j(C)) \nabla_{\vec{x}_j} \iota_j(C).$$

The gradient of $\iota_j(C)$ *is still missing. We will neglect the not differentiable modulo operator and must then compute*

$$\nabla_{\vec{x}_j}(\operatorname{arctan2}(\vec{v}_1(C)) - \operatorname{arctan2}(\vec{v}_2(C))),$$

where $\vec{v}_1(C) = (x_{j-1} - x_j, y_{j-1} - y_j)^T$ *and* $\vec{v}_2(C) = (x_{j+1} - x_j, y_{j+1} - y_j)^T$.
The function arctan2 *is partly defined and not truly differentiable. We still want to compute a gradient to use it for our interior angle force. Since* $\operatorname{arctan2}(x, y) = \arctan(\frac{y}{x})$ *+ constant almost everywhere, we will use the function* $g(x, y) = \arctan(\frac{y}{x})$ *for the derivation, because the different constants do not matter in the derivation. With g, we can rewrite* $\iota_j(C) = g(x, y) \circ \vec{v}_1(C) - g(x, y) \circ \vec{v}_2(C)$.
Thus, we need to determine

$$\nabla_{\vec{x}_j}(g(x,y)\circ\vec{v}_1(C)-g(x,y)\circ\vec{v}_2(C)).$$

The partial derivatives of g are

$$\partial_x \arctan(\tfrac{y}{x}) = -\frac{y}{x^2}\frac{1}{1+(\frac{y}{x})^2} = -\frac{y}{x^2+y^2},$$
$$\partial_y \arctan(\tfrac{y}{x}) = \frac{1}{x}\frac{1}{1+(\frac{y}{x})^2} = \frac{x}{x^2+y^2}.$$

It is easy to see that

$$\partial_{x_j}\vec{v}_1(C) = \partial_{x_j}\vec{v}_2(C) = (-1,0)^T, \partial_{y_j}\vec{v}_1(C) = \partial_{y_j}\vec{v}_2(C) = (0,-1)^T.$$

This implies

$$\begin{aligned}\partial_{x_j}\iota_j(C) &\triangleq \partial_{x_j}(g\circ\vec{v}_1(C)-g\circ\vec{v}_2(C))\\ &= (g'(\vec{v}_1(C))\partial_{x_j}\vec{v}_1(C)-g'(\vec{v}_2(C))\partial_{x_j}\vec{v}_2(C)\\ &= -(\partial_x g)(\vec{v}_1(C))+(\partial_x g)(\vec{v}_2(C))\\ &= \frac{v_{1,y}}{v_{1,x}^2+v_{1,y}^2}-\frac{v_{2,y}}{v_{2,x}^2+v_{2,y}^2},\end{aligned}$$

using the multidimensional chain rule. In a similar fashion, we obtain

$$\partial_{y_j}\iota_j(C) = -\frac{v_{1,x}}{v_{1,x}^2+v_{1,y}^2}+\frac{v_{2,x}}{v_{2,x}^2+v_{2,y}^2}.$$

Together this yields

$$\nabla_{\vec{x}_j}\iota_j(C) = \frac{1}{\|\vec{v}_1\|_2^2}\begin{pmatrix}v_{1,y}\\-v_{1,x}\end{pmatrix}+\frac{1}{\|\vec{v}_2\|_2^2}\begin{pmatrix}-v_{2,y}\\v_{2,x}\end{pmatrix},$$

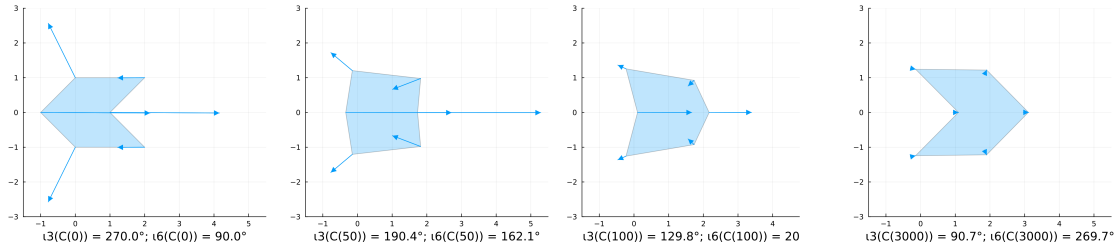


Figure 17: This figure shows how the interior angle force acts on the vertices of a DF cell. The initial state can be seen in the first diagram. The desired state is the horizontally mirrored version of the initial state. Below each chart, we can see the current interior angles at $t \in \{0, 50, 100, 3000\}$ of the two vertices that have the y value zero. The desired states are 90° for the right and 270° for the left considered vertex. The interior angle force ensures that each interior angle transitions to the desired state over time, as we can see in this figure.

which corresponds to the term from the proposition.

□

As in the last two paragraphs, we can see the effect of $F_j^{(I_{ij})}(C)$ in an example in Figure 17. The interplay between the edge- and inter angle forces result in a quite nice ability of the cell to develop into the desired cell shape. Figure 20 in Subsection 4.5 illustrates this quite well.

4.4 Overlap energy

While all previous energies are operating on each cell individually, the overlap energy is the first one that considers multiple cells and thus, adds the first cell interaction into the simulation. The overlap energy has quite a lot of similarities to the area energy, since in both cases the goal is to achieve a certain cell area.
But in difference to the area energy, it is not a cell from the considered system $\vec{C}$ whose area gets optimized, but new DF cells that are exactly equal to the overlap that occurs between two cells $C_i, C_j \in \vec{C}$. The new DF cells in the set of all overlaps between the original cells C_i and C_j $\Omega_{ij} = (D_1, \ldots, D_{k_j})$ fulfil the relation

$$C_i \cap C_j = \amalg_{k=1}^{k_j} D_k,$$

where $\amalg$ symbolises a disjointed union and k_j denotes the number of overlaps that occur between C_i and C_j. As already explained in Section 2, every overlap that appears between two cells C_i and C_j can be nicely expressed through a new DF. Every vertex from an overlap cell is either included in C_i, C_j or in the set I_{ij} that holds all intersection points of C_i and C_j. The determination of those overlap cells is also derived Section 2. Having all requirements stated, we can now introduce the overlap energy.

Definition 4.12. Overlap energy
Let C_i and C_j be two cells from the system $\vec{C}$ and Ω_{ij} be the set of all overlap DF that appear between C_i and C_j, like explained above. Then, the overlap energy of C_i is given by the formula

$$O_i(\vec{C}) := \sum_{j=1, j\neq i}^{M} \Big(\sum_{D_j \in \Omega_{ij}} a(D_j) \Big), \tag{15}$$

where a is the area functional (7).

The first sum iterates over all other cells in the system, while the second sum iterates over all overlaps that occur between both cells. We do not need an absolute value, because the area functional a always computes non negative values.
An important assumption for the overlap force is that the vertices $\vec{x}_{ij}$ of C_i that are also included in the overlap D are identified with the according vertices in D. Let $\omega_{ik} = C_i \cap D_k$ be the set of the vertices that are in $C_i = (\vec{x}_{i1}, \ldots, \vec{x}_{iN})$ and

$D_k = (\vec{d}_{k1}, \dots, \vec{d}_{kL})$. Then, we can find an overlap vertex $\vec{d}_l$ that corresponds to $\vec{x}_j$, whenever $\vec{x}_j \in \omega_{ik}$. In the computation for the overlap force, we can then use $\nabla_{\vec{d}_l} a(D_k)$.
If a vertex $\vec{x}_{ij}$ is not part of the overlap D_k then this overlap will not have an impact on this vertex.

Proposition 4.13. ***Overlap force***
The overlap force $F_j^{(O_i)}$ that acts on $\vec{x}_{ij}$ is given by

$$F_j^{(O_i)}(\vec{C}) = \sum_{m=1, m\neq i}^{M} \Big(\sum_{D_k \in \Omega_{im}} -\mathbb{1}_{\omega_{ik}}(\vec{x}_{ij}) a(D_k) \nabla_{\vec{d}_l} a(D_k) \Big), \tag{16}$$

with $\nabla_{\vec{d}_l} a(D_k)$ given as

$$\nabla_{\vec{d}_l} a(D_k) = \frac{1}{2} \begin{pmatrix} d_{l+1}^y - d_{l-1}^y \\ d_{l-1}^x - d_{l+1}^x \end{pmatrix},$$

with $\vec{d}_l$ being the corresponding vertex to $\vec{x}_{ij}$ in the according overlap and $\vec{d}_{l-1}$ and $\vec{d}_{l+1}$ being the vertices before and after $\vec{d}_l$.
Proof.
Instead of just using one scaling factor for each vertex, we will use a scaling factor $\alpha_{O_i, D_k} = a(D_k)$ for each individual overlap D_k.
For all vertices $\vec{x}_{ij} \notin \omega_{ik}$, that are not included in the overlap D_k, the force is zero, because a change of position would not impact the area of the overlap in this case. This produces the indicator function $\mathbb{1}_{\omega_{ik}}(\vec{x}_{ij})$ in the formula, that makes the force vanish for $\vec{x}_{ij} \notin \omega_{ik}$.
Per definition of ω_{ik}, we can always find an overlap vertex $\vec{d}_l$ that corresponds to $\vec{x}_{ij}$ if $\vec{x}_{ij} \in \omega_{ik}$. In this case, we can apply the gradient with respect to $\vec{d}_l$ on the area functional of D_k, to compute the direction of the fastest descent.
Thus, we must solve

$$F_j^{(O_i)}(\vec{C}) = \sum_{l=1, l\neq i}^{M} \Big(\sum_{D_k \in \Omega_{il}} -\mathbb{1}_{\omega_{ik}}(\vec{x}_{ij}) a(D_k) \nabla_{\vec{d}_l} a(D_k) \Big).$$

We can use the gradient computation shown in the area force, to determine

$$\nabla_{\vec{d}_l} a(D_k) = \tfrac{1}{2} \begin{pmatrix} d_{l+1}^y - d_{l-1}^y \\ d_{l-1}^x - d_{l+1}^x \end{pmatrix}.$$

□

In order to compute the gradient correctly, it is necessary that the overlap cell has a positive orientation. Otherwise, we would get a wrong sign, because $\vec{d}_{j-1}$ and $\vec{d}_{j+1}$ would be swapped. But this is also guarantied in the computation of the overlap.
Figure 18 shows the cell interaction between two overlapping cells with the overlap force acting on the vertices.

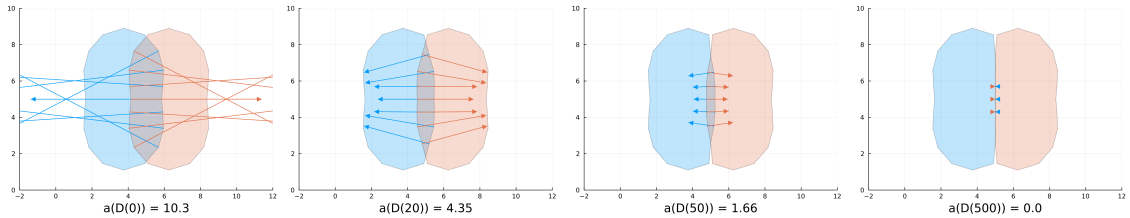


Figure 18: Here, we can see how the overlap force acts on two overlapping DF cells. The blue arrows represent the forces acting on the blue cell, the red arrows the forces of the red cell. The cells are shown at the times $t \in \{0, 20, 50, 500\}$, as well as the current area of the cell overlap D. On each consecutive diagram, we can see that the overlap gets reduced, until the area of the overlap is zero at $t = 500$.

4.5 Discrete shape recovery

We have now introduced all the forces that we want to use in this thesis. The next step is to combine the forces to see how the interactions between the forces affect the cell developments.
First, let us take a look at the first three forces, that shall make the cell develop into a certain desired state. Since these forces act on each cell separately, it is sufficient to consider a single DF cell.
In order to make it quite difficult for the forces to ensure the desired state, we choose a desired state, that has a completely different shape than the initial cell. Both states are shown in Figure 19.

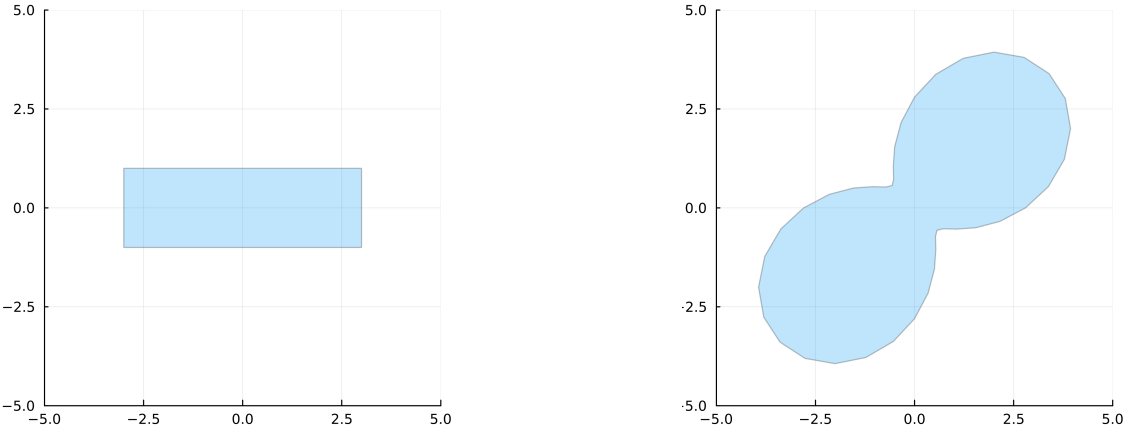


Figure 19: The left figure shows the initial state and the right figure shows the desired state for the ODE, which tests the three overlap, edge and interior angle forces for their ability to restore a cell shape. Both cells have an amount of 40 vertices.

On the one hand, the cell is initially identical to the rectangle $[-3, 3] \times [-1, 1]$ with an area of 12. We have 11 vertices on each edge, which are evenly distributed in relation to the individual edges. The total amount of vertices is still 40, because

each vertex at a rectangle corner is located on two different rectangle edges. This results in edge lengths of 0.6 on the horizontal cell edges and 0.2 on the vertical cell edges. Except for the corner points of the rectangle, where the interior angles are 90°, every other vertex has an interior angle of 180°.
On the other hand, we have got the desired shape that looks like a peanut. The area is roughly 29.2. The edges near the centre point $(0, 0)$, with a minimum length of ca. 0.16, are smaller than the peripheral edges, with a maximal edge length of ca. 0.80. The two vertices closest to the centre have an interior angle of about 243°, while the other interior angles decrease rapidly toward the outer cell regions, reaching a minimum value of 156°.
So, almost all cell parameters are quite different between the two states.
The ODE, that we will now apply on the initial state's vertices, reads

Model 4.14. ***Discrete shape recovery model***
The discrete shape recovery model is given by

$$\frac{d\vec{x}_j}{dt} = F_j^{(A)}(C) + F_j^{(E_j)}(C) + F_j^{(I_j)}(C). \tag{17}$$

It uses the first three energies that operate on single cells.

with the first three forces from the previous paragraphs. The solution can be seen

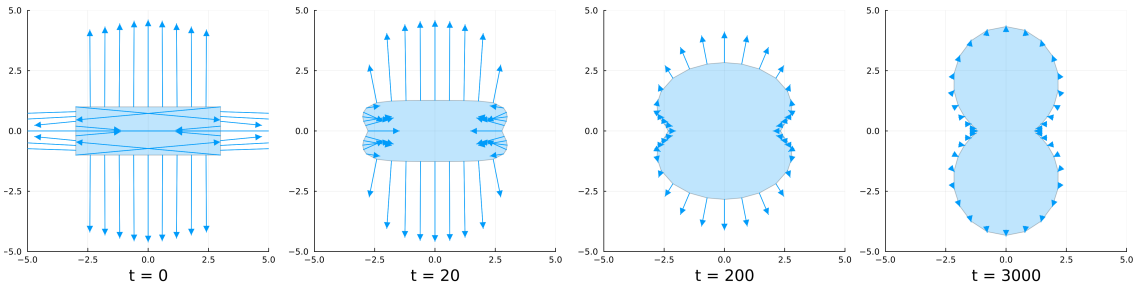


Figure 20: This series of plots shows the solution to the ODE just explained. For $t = 0$, we can see the initial condition from Figure 19 and the forces that initially act on the vertices. The following three diagrams show the solutions at the times $t \in \{20, 200, 3000\}$. They show a good transition from the starting shape into the desired state.

in Figure 20. The result at $t = 3000$ is congruent to the desired state, but it took a lot of time steps to get there. For $t > 500$, the acting forces are really weak, although the desired state is not finally reached. The transformation is quite slow at the end. The reason therefor could be that the scaling factors in the area-, edge- and interior angle terms got smaller and smaller, because the shape got closer to the desired shape.
It is also noticeable that the final cell shape from the ODE solution is congruent, but rotated by 45°. The reason for this is, that no force has an impact on the cells rotation, but only on the characteristics area, edge lengths and interior angles. As long as these values are met as required, no further force is exerted.

All in all, it is a good result that the forces can manage the cell transformation into a specific shape, even though the initial situation is quite different.

4.6 Interactive system

Now, we can start the consideration of the interactive system with overlap energy. Therefor, we will now consider a system of $M = 9$ DF cells with $N = 20$ vertices each. Since the focus is now set on the overlap energy and not on the shape recovery, we set the desired cell shape equal to the initial shape. Thus, only cell deformations caused by the overlapping forces should be regenerated into the desired shape.
We choose the initial condition as shown in the first diagram of Figure 21. All cells have the same shape and are concentrated on the bottom left corner of our domain G. The cells are initially arranged in such a way that there is a lot of overlap to show how the overlap energy affects the dynamics.
There are some terms that we must add to the ODE, in order to get a useful simulation.
First of all, we want every cell to move around. Thus, every cell should perform a random Brownian motion like in the papers from Bruna and Chapman in the introduction. This turns our previously deterministic ODE into a stochastic differential equation (SDE). In every time step, a normally distributed vector is generated for each cell, that gives it a random motion. Every vertex from the same cell gets the same Brownian motion applied such that the cell shape as a whole is not influenced, but only the cell position. We choose a diffusion constant of $D = 2$.
On top, we will do a rescaling for our forces. Simulations without a rescaling do not show a big impact. But since we especially want our overlap force to have an impact as soon as two cells overlap, we multiply its force by a factor of ten. In summary, we obtain the SDE

Model 4.15. ***First interacting system***
The sum of all derived forces yields the first interacting system

$$d\vec{x}_{ij}(t) = F_j^{(A_i)}(C_i) + F_j^{(E_{ij})}(C_i) + F_j^{(I_{ij})}(C_i) + 10F_j^{(O_i)}(C_i) + \sqrt{2D}dB_t^{(i)}, \tag{18}$$

where $\vec{x}_{ij}$ again stands for the vertex j of the ith cell, $(1 \leqslant i \leqslant 9,\ 1 \leqslant j \leqslant 20)$.

Except for the last addend, which makes up the stochastic part of the SDE, all other addends belong to the deterministic part. Figure 21 shows snippets of the solution. It is hard to show the important processes in this simulation with just a few frames, but this should give a good intuition. One can see that in situations where the Brownian motion causes heavy collisions between different cells, the overlap force starts acting and minimizes the overlapping area. Despite of this working well, we can see in this simulation, that the cells seem to be too static. It looks like, it is not just the area in the overlap, that gets minimized, but the cell as a whole. The reason therefor is probably that the interior angle force is too large, causing a lack of cell flexibility, because the shape cannot change, since the interior angles stay the same. We can also see that the influence of the area force is too small.

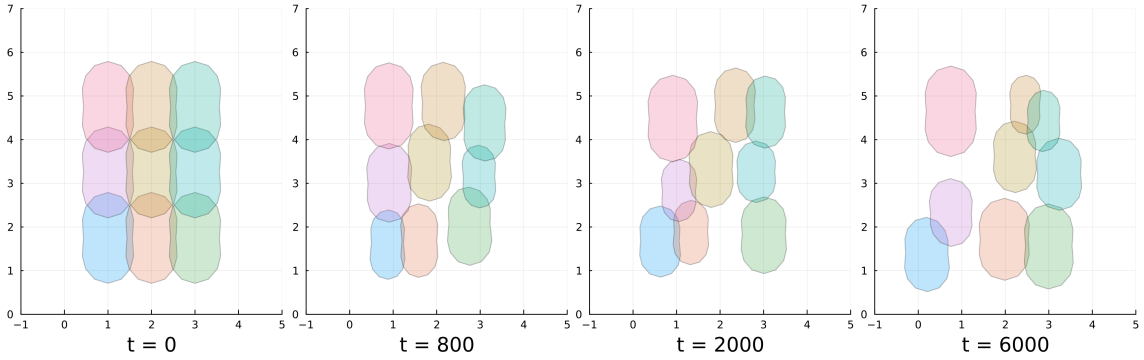


Figure 21: Here, we can see a solution to the described SDE (18) with a rescaled overlap force by the factor of 10. The solution is shown at the times $t \in \{0, 800, 2000, 6000\}$.

So let us take a look at another simulation with a different scaling for the force functions. We consider the SDE

Model 4.16. ***Rescaled interacting system***
A rescaling of Model 4.15 yields the rescaled interacting system

$$d\vec{x}_{ij}(t) = 50F_j^{(A_i)}(C_i) + 1.5F_j^{(E_{ij})}(C_i) + F_j^{(I_{ij})}(C_i) + 80F_j^{(O_i)}(C_i) + \sqrt{2D}dB_t^{(i)}. \tag{19}$$

A solution to this SDE is illustrates in Figure 22. In fact, we can see some differences

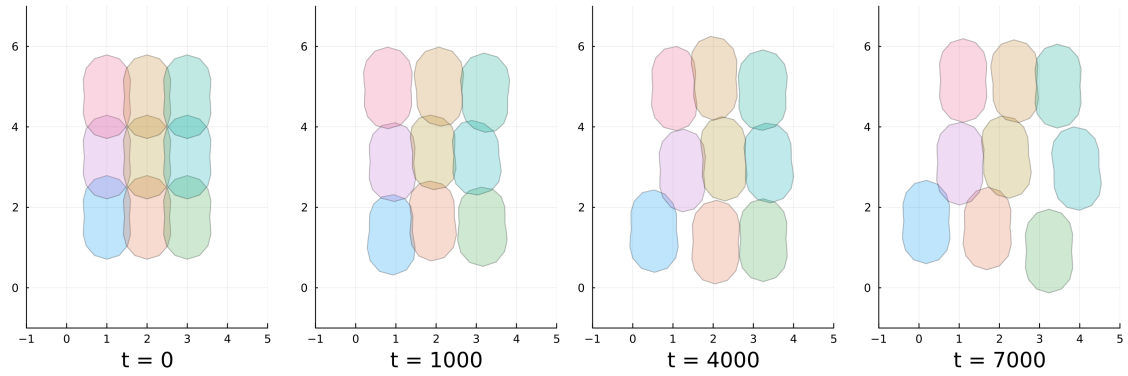


Figure 22: This figure shows different plots of a solution to the explained SDE (19) with the rescaled force configuration. We can see the cells at the times $t \in \{0, 1000, 4000, 7000\}$.

to the first SDE and exactly the desired effect occurred. Each cell is more flexible and has a slightly different shape compared to the others. It is not just the general cell size that changes. The flexibility makes it possible to resolve the overlap more quickly, as the free spaces are better used by the cells to expand into. The interaction effect between the cells looks much smoother, because the different cell walls fit very tightly together.

We can conclude that a higher scaling factor for the edge and interior angle forces implies that the cells get more static and less adaptable to changes in the cell shape. In summary of the interactive system, we can say that we have succeeded in creating a dynamic that is able to resolve overlaps while also maintaining a desired cell structure. In the absence of the overlap, the simulation from Subsection 4.5 has shown that even strong shape changes can be regenerated back to the desired state by the cell-internal area, edge and interior angle forces.

5 Conclusion

This bachelor thesis can be divided into two sections. The goal of the first part is to develop cell models that are able to represent complex shapes.
In Chapter 2, we derive the two cell forms that are the centre radius form (CRF) and the discrete form (DF).
The CRF is continuous in the sense that it contains the information for each point of the cell wall. Using the CRF one is able to represent every star shaped cell form. For the computation of the cell area, we introduced the smooth indicator function. It uses the data from a CRF and can determine if any point $\vec{x} \in \mathbb{R}^2$ is inside of the cell or not. It has similarities to the phase field model from [Happel and Voigt, 2023]. As the name suggests, the smooth indicator function of a cell can be used to calculate the cell area or the cell overlap of several cells via an integral using the product of their smooth indicator functions, just as it would be possible with the normal indicator function.
However, DF cells are a sequence of $N \in \mathbb{N}$ vertices that form a polygon when connecting adjacent vertices. As simple as the approach is, the possibilities for establishing cell dynamics are just as great. While calculating the area is easy using the shoelace formula, determining the overlap between 2 DF cells requires a complex algorithm.
After introducing the cell models and deriving area and overlap calculations, the foundation for the development of cell dynamics is laid, which leads us to the second main section of this work.
For the CRF model, we have found an ODE that can smoothly transform a cell from a certain initial state $C^{(0)}$ into a desired CRF $C^{(1)}$ in Section 3. There are no further extensions developed for CRF dynamics.
Instead, the focus in the following is on the dynamics of the DF cells in Section 4. Here, we examine various energy functions that measure specific energy potentials within a cell or between different cells. With the usage of gradient flows, we could derive forces from the energy that got applied onto the cells vertices in order to reduce the energies. This procedure is similar to that from the paper [Fletcher et al., 2014]. After examining each force individually, we begin to combine the forces, resulting in a serviceable dynamic with interactions of an entire cellular system.
Of course there are some meaningful extensions for my thesis.
In order to compute solutions to the energy dynamics, I have just used an explicit Euler method for the deterministic differential equations and Euler Maruyama schemes for the stochastic differential equations with a fixed time step size of $\Delta t = 2^{-8}$. Other methods, e.g. methods with adaptive time stepping, are suited better for computing more accurate solutions at times when rapid changes to the solution occur.
If needed, it is also possible to introduce new energies and forces into the energy based system. For example, if the cell's rotation plays an important role, one could implement a force, that makes the cell rotate into a certain position.
The rescalings, showed at the end of Chapter 4, are just possible examples of scalings. Depending on the desired cell properties, like for example deformability, other scalings may be preferred.
Another possible use case could be to define a domain Ω such that cells are not

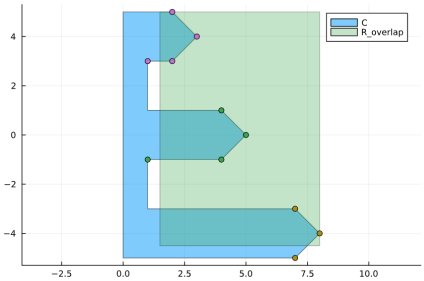
allowed to leave it. This could be implemented with a further force function $F^{(B)}$ 'boundary push' that pushes the whole cell back into the domain, if it touches the boundary $\partial\Omega$.
In [Bruna et al., 2023] Bruna, Chapman and Schmidtchen studied a macroscopic model for Brownian hard needles. Similar to the hard sphere model in the paper [Bruna and Chapman, 2012] from the introduction, there are exclusion effects for the different needle particles. The authors managed to derive an effective PDE that describes the probability density $\rho(t, \vec{x}, \theta)$ of finding a needle with rotation θ at position $\vec{x}$ and time t. An interesting next step for the development of the theory in this thesis would be to derive an effective PDE for our energy dynamic model of the DF cells.
Another problem could be the elaborate computation of the discrete cell overlap. Here, one could try to find faster algorithms to solve this. If the setup allows it, assuming convex polygons could simplify some calculations. An approach, that I tried to implement, does not determine every edge of the cell, but only those at critical areas. It is explained in the appendix 5. The approach worked in most cases, but unfortunately there are also configurations where the algorithm failed if one does not take further assumptions. Since these situation are mostly made up, it could be possible to deploy this ansatz.

Appendix

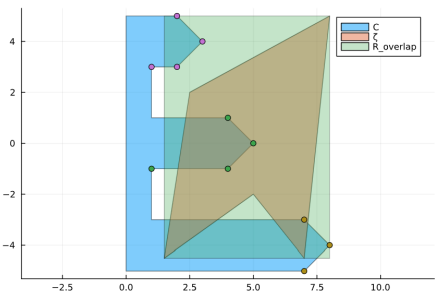
An attempt to simplify the discrete overlap computation

While researching the implementation of calculating the overlap of two DF cells in the Chapter 2.4, I also focused on an idea to simplify the Problem 2.11 and reduce the computational effort.
Unfortunately, I discovered counterexamples that contradicted my idea. However, since these counterexamples are rather artificial, this method may still have a right to exist. Perhaps, one can figure out how to circumvent the problem of the counterexample. One could also add constraints to the model, e.g. that only convex cells are regarded or that all occurring overlaps are convex.
In difference to the showed algorithm in the Chapter 2.4, not every cell edge is critical. We choose the following definition.
Definition. Critical edge
Let C and ζ be two overlapping DF cells and $R_{overlap}$ the smallest rectangle that covers all overlap areas. Than any edge is considered to be critical if it intersects with $R_{overlap}$.
The concept is only to consider lists of consecutive edges that are located in the critical area $R_{overlap}$.

Definition. Critical list
A critical list is an ordered, maximal list $L = (l_1, \ldots, l_L) = (e_i, e_{i+1}, \ldots, e_M)$ of critical edges such that all edges in the list are consecutive. Maximal means one cannot add another consecutive critical edge before or at the end of L.



(a) Here, we only concentrate on the cell C and $R_{overlap}$. Every vertex is associated with the counterclockwise next edge. The cell intersects the rectangle multiple times, causing several critical lists. Each of the three list is marked with its own colour. Only critical edges are marked with a dot.



(b) Now, we can also see the second discrete cell ζ. There are three areas where both cells overlap. Note that although the orange critical list takes part in two overlaps, it is does not happen that two different critical lists of the same cell involve in the same overlap.

Figure 23: Here, we can see a pattern of two cells C and ζ and the domain $R_{overlap}$. This figure gives a perception of critical lists and what they are used for.

I was hoping that each intersection of the two cells is always build with edges of just one critical list per cell. Then, the original Problem 2.11 could have been simplified to the following problem.

Problem.
For given critical lists L from C and λ from ζ, determine all discrete cells $O_1, \ldots, O_K$ such that any overlap that uses edges of L and λ is represented by exactly one O_k $(k \in 1, \ldots, K)$.
Figure 24 shows an example of two DF cells that have an overlap which uses edges of two different critical lists of the same cell. After finding that out, I decided to consider every edge to be critical and developed the procedure that is found in Chapter 2.4.

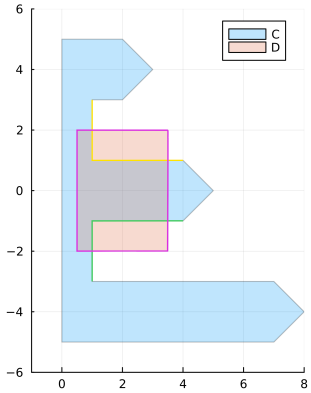


Figure 24: This plot shows a configuration of two DF cells C and D. The rectangle $R_{overlap}$ is identical to D in this case. Thus, each edge of D is critical and we obtained one critical list for D that is shown in pink. For C we get two critical lists that are shown in yellow and green. We have got one overlap between C and D and it uses edges from each the green and the yellow critical list of C.

References

[Bonvallet, 2009] Bonvallet, R. (2009). How to determine if a list of polygon points are in clockwise order? URL: https://stackoverflow.com/questions/1165647/how-to-determine-if-a-list-of-polygon-points-are-in-clockwise-order. Last accessed on 23.11.2023.

[Bruna and Chapman, 2012] Bruna, M. and Chapman, S. J. (2012). Excluded-volume effects in the diffusion of hard spheres. *Phys. Rev. E*, 85:011103.

[Bruna et al., 2017] Bruna, M., Chapman, S. J., and Robinson, M. (2017). Diffusion of particles with short-range interactions. *SIAM Journal on Applied Mathematics*, 77(6):2294–2316.

[Bruna et al., 2023] Bruna, M., Chapman, S. J., and Schmidtchen, M. (2023). Derivation of a macroscopic model for brownian hard needles. *The Royal Society*.

[Büsing, 2008] Büsing, H. (2008). Multivariate numerische Integration und Anwendungen in der Peridynamik. *TU Berlin*.

[Fletcher et al., 2014] Fletcher, A., Osterfield, M., Baker, R., and Shvartsman, S. (2014). Vertex models of epithelial morphogenesis. *Biophysical Journal*, 106(11):2291–2304.

[Happel and Voigt, 2023] Happel, L. and Voigt, A. (2023). Coordinated motion of epithelial layers on curved surfaces. *Cluster of Excellence, Physics of Life, TU Dresden*.

[ShoelaceFormula, 2014] ShoelaceFormula (2014). Green's theorem and area of polygons. blogoverflow. Published by: apnorton. URL: https://math.blogoverflow.com/2014/06/04/greens-theorem-and-area-of-polygons/. LAst accessed on 23.11.2023.

[ShoelaceIllustration, 2022] ShoelaceIllustration (2022). Deriving the trapezoid formula. URL: https://commons.wikimedia.org/wiki/File:Trapez-formel-prinz.svg. Published by user 'Ag2gaeh'. Last accessed on 23.11.2023.

Statement of authorship

I hereby declare that I have written this thesis (*Modelling of Cells and their Dynamics*) under the supervision of Jun.-Prof. Dr. Markus Schmidtchen independently and have listed all used sources and aids. I am submitting this thesis for the first time as part of an examination. I understand that attempted deceit will result in the failing grade „not sufficient“ (5.0).

Tim Vogel
Dresden, November 27, 2023
Technische Universität Dresden
Matriculation Number: 4930487