University of Amsterdam

Masters Thesis

Red Blood Cell Packing in HemoCell: An Analysis and Implementation of the Mesh-Based 3D Irregular Object Packing Algorithm

Author:
Maurits Bos

Supervisor:
Konstantinos Asteriou

Examiner:
dr. Gábor Závodszky

Assessor:
dr. ir. Ana Lucia Varbanescu

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Computational Science

in the

Computational Science Lab
Informatics Institute

June 2023

Declaration of Authorship

I, Maurits Bos, declare that this thesis, entitled 'Red Blood Cell Packing in HemoCell: An Analysis and Implementation of the Mesh-Based 3D Irregular Object Packing Algorithm' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the University of Amsterdam.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date: 19 June 2023

"Everything should be made as simple as possible, but not simpler"

Albert Einstein

UNIVERSITY OF AMSTERDAM

Abstract

Faculty of Science
Informatics Institute

Master of Science in Computational Science

Red Blood Cell Packing in HemoCell: An Analysis and Implementation of the Mesh-Based 3D Irregular Object Packing Algorithm

by Maurits Bos

The study of packing algorithms is crucial for numerous industrial applications due to their implications for space efficiency, cost reduction, and profit maximisation. This thesis explores the implementation and analysis of a mesh-based method, as proposed by Ma et al. in the paper entitled by "Packing Irregular Objects in 3D Space via Hybrid Optimisation". By doing so, this thesis aims to improve the efficacy of HemoCell, a computational tool used for simulating blood cell interactions that is developed by the University of Amsterdam's Computational Science Lab.

The existing algorithm used by HemoCell, packCells, employs ellipsoids for computational efficiency, but fails to capture the unique biconcave shape of red blood cells, leading to sub-optimal packing densities, computational wastage, and unrealistic hematocrit values. On the contrary, the mesh-based method examined in this thesis addresses irregular shapes more accurately and proposes a more precise packing approach.

Our analysis revealed that the mesh-based method, in its current incarnation, demonstrates less computational competitiveness compared to packCells, owing to the increased complexity and increased computational overhead it brings.

However, the research was successful in exhibiting the mesh-based method's potential for parallelisation, owing to its novel technique of isolating objects using Chordal Axis Transform (CAT) cells. Preliminary packing results, despite being less promising than that of packCells, indicate the basic functionalities of the mesh-based method and show its potential to achieve superior object configuration and a higher coverage rate, closer to real-world hematocrit values, with further development.

Scalability analysis shows a constant relation between the number of operations per cell and the Constrained Delaunay Triangulation (CDT) algorithm, indicating a computational complexity comparable to CDT.

This research serves as an initial step towards understanding and optimising a mesh-based method for irregular object packing in HemoCell, providing invaluable insights into the complexities of the approach and opening avenues for future research directed towards improving the efficiency and realism of cellular structure simulation. The result of this research is an open-source package that lays the foundations for future advancements and optimisation. Although not yet stable, the code base adheres to best practises and has been developed with scalability in mind.

Acknowledgements

This thesis would not have been possible without the support of the people around me. Leaving the faculty of applied physics at the TU Delft, was not the most logical step in my career, but it was the best one. I am grateful to the University of Amsterdam for giving me the opportunity to pursue my interests in computational science and for providing me with the resources to do so.

To my parents and sisters, whom have always supported me in my decisions, I am grateful for their unconditional love.

To my esteemed mentors, Gabor and Konstantinos, I am indebted for their guidance. I've felt like my work was appreciated from the first moment on. I would like to especially thank Kostis, for his support, his welcoming attitude, and for always being available to answer my questions. I'm happy to have had the opportunity to work with him. Also I would like to thank Gabor for the fact that, despite our early conversations about and our love for the beautifully typed programming language Rust, he steared me towards the more practical Python.

Then, I would like to thank my flatmates, who I consider dearest friends, for consuming the endless sprees of information and thoughts completely irrlevant to them.

Also, I would like to thank Daniel, the founder of the company I work for, for giving me the flexibility to work with him during this thesis and for showing me how to break in to the speakers dinner in Prague.

Finally, to all my friends. Im thankful for the fact that we remain friends despite the fact that I'm not always the best at keeping in touch.

Contents

List of Figures

List of Tables

Abbreviations

BFDH	**Best Fit. Decreasing Height**
CAT	**Chordal Axis Transform**
DT	**Delaunay Triangulation**
CDT	**Constrained Delaunay Triangulation**
FEM	**Finite Element Modelling**
FFDH	**First Fit Decreasing Height**
GPU	**Graphical Processing Unit**
IBM	**Immersed Boundary Method**
AOT	**Ahead Of Time.**
JIT	**Just In Time**
LBM	**Lattice Boltsman Method**
NP	**Non-Polynomial time**

Symbols

Symbol	Name	Unit
A_{avg}	average surface area	m^2
f_{init}	Initial scale	-
f	scale	-
$\theta_{\max}$	maximum angle of rotation, bound	rad
$(\theta_x, \theta_y, \theta_z)$	Angle of rotation	rad
(t_x, t_y, t_z)	Translation	m
$T_{4\times4}$	Homogeneous Transformation Matrix	-
$\vec{n}_j$	normal vector of face j	m
q_j	any point on face j	m
r	coverage rate, hematocrit	-
R_i	Radius of the outer encapsulating sphere	m
v_i	object vertex i	m
$w(v_i)$	constraint for v_i	-

Chapter 1

Introduction

Envision the everyday act of trying to fit your groceries into a limited cabinet space, or the fun challenge of aligning falling blocks in a game of Tetris. Perhaps, even recall the complex process of stuffing your luggage to capacity before a long trip. Now, transpose these experiences to a grander scale, and you have the world of 'packing'. This concept underpins numerous facets of our life and work, becoming particularly interesting in industries that rely on efficient use of space. In many of these, every centimeter saved translates to potential cost reductions and amplified profit. In addition to its practical applications, packing also intrigues researchers, particularly those involved in computational modelling. Significant effort is dedicated to studying and understanding the detailed methods for achieving the most effective packing strategies.

The packing problem, however complex, is essentially a question of optimisation. How can we fit the maximum quantity of objects into a given space? This is a relatively simple endeavor when the objects are uniformly shaped and equal of size, such as spheres or cubes. However, the challenge multiplies exponentially when we introduce irregularly shaped objects into the equation. Packing can be challenging for both 2D and 3D shapes, with the later being the more complex. Real-world objects, whether they are red blood cells, parcels or machine parts, rarely adhere to simple forms with high level symmetry. They tend to exhibit irregular and often convoluted shapes, turning the packing problem into a highly complex and computationally intensive task. In order to address this situation, various models and techniques have been developed in fields as diverse as computational geometry, operations research, and artificial intelligence, contributing to a robust, multidisciplinary body of knowledge that underpins our understanding of packing irregularly shaped objects.

The industrial interest in the packing problem, originates from the shipping industry, where the amount of cargo that is placed on the ships is directly related to the profit

margins. For ship loading, the complexity of the packing objects is rather simple, but finding a good solution still requires solid mathematical understanding. More modern industries like 3D printing and medical computational modelling, have lifted the interest from trivial shape packing to more complex shapes. In the field of computational biology, the packing problem is used to compute the initial configurations of red blood cells with a high coverage rate (medical term: hemocrit value) in the blood stream. Coverage rate refers to the percentage of volume that is covered by objects, in this case blood cells.

Developed by the Computational Science Lab of the University of Amsterdam, HemoCell [5, 6] stands as a prime example of computational modelling applied to the field of biology, particularly, the study of blood cells. This sophisticated computational model aspires to simulate how different types of blood cells and blood-related structures interact and behave within a fluid environment, an endeavor that is parallel to observing how red blood cells or platelets move in our bloodstream.

HemoCell does not merely provide an overview of these movements. It strives to recreate the physics of it in high detail. This is achieved by mimicking the intricate interactions these particles have with the fluid medium they are part of, providing a continual account of their movements and behaviours. This focus on interaction and behaviour allows HemoCell to encapsulate the dynamics of cellular objects within a fluid environment with extremely high accuracy.

To further understand how HemoCell operates, it's best to draw an analogy with a commonplace physical phenomenon. Picture dropping a pebble into a tranquil pond and witnessing the ripples it creates. HemoCell, in essence, employs a similar technique to model fluid dynamics. This technique is referred to as the Lattice Boltzmann Method (LBM)[7], which acts as the catalyst in driving the interactions between cellular structures and the fluid. Under the hood, LBM describes the time dependent behaviour of the fluid with the Navier-Stokes equations, and uses numerical approximations to solve them. The LBM is particularly well-suited for modelling fluid dynamics in complex geometries, such as the ones found in the human body. This is because the LBM is a mesoscopic method, meaning that it operates at a scale that is larger than the molecular scale, but smaller than the macroscopic scale. This allows the LBM to capture the complex interactions between the fluid and the cellular structures, while still being computationally efficient.

HemoCell goes a step further by coupling these fluid dynamics to the mechanical behaviour and simulation of the cellular objects. This is done by using the Immersed Boundary Method (IBM)[8], which allows the model to accurately track and predict the movement and behaviour of cells within the fluid. The IBM is a numerical approach

designed to handle the interaction between fluid and a flexible or rigid structure immersed in it. This could be the flow around an object, like a red blood cell moving in the bloodstream, or the movement of a flexible structure, like a heart valve opening and closing. This method approximates the immersed boundary — the cell or structure — by a series of discrete points, which communicate their influence to the surrounding fluid through a mathematical force distribution. This mechanism allows the model to accurately track and predict the movement and behaviour of cells within the fluid.

HemoCell is not just notable for its detail-oriented approach, but also for its capacity to manage substantial amounts of data and intricate interactions. It efficiently balances the computational workload, thereby making it a robust tool in the exploration of biological interactions and behaviours [6]. By managing and making sense of the complex data associated with blood cell behaviour, HemoCell brings us one step closer to understanding the microcosm that resides within the larger biological world.

HemoCell comes with a set of built-in, well-tested models that allow for the simulation of red blood cells, and it is flexible enough to allow users to add their own models. Currently, when setting up these simulations, HemoCell uses a tool called packCells to figure out the best way to fit these complex shaped objects into a space. PackCells is a C++ software package that aims to optimise the parallelisation of dense packing of specifically ellipsoids using the force-biased algorithm [9]. HemoCell leverages this for red blood cells, by approximating its donut-like shape with an ellipsoid, which allows for an computationally efficient packing implementation.

The underlying method used by packCells is the force-biased algorithm. This method is based on the idea that the objects are repelling each other, and that the repulsion force is proportional to the overlap between the objects.

The algorithm starts by placing the objects randomly in the space, and then iteratively moves the objects around until the repulsion forces are minimised. This is done by repeatedly calculating the repulsion forces between all the objects and moving the objects in the direction of the force, until the forces are minimised and the objects are packed as tightly as possible. The algorithm is efficient, and thanks to the packCells implementation it is also parallelisable, which allows for the packing of large amounts of objects in a relatively short amount of time.

packCells presents certain limitations that impact its effectiveness. First and foremost, the primary limitation lies in the approximation of red blood cells as ellipsoids that encapsulate the cells, displayed in Figure 1.1. Red blood cells, in reality, have a unique, biconcave shape that cannot be perfectly represented by an ellipsoid. The approximation is boundary preserving, meaning that the red blood cell is always contained within the

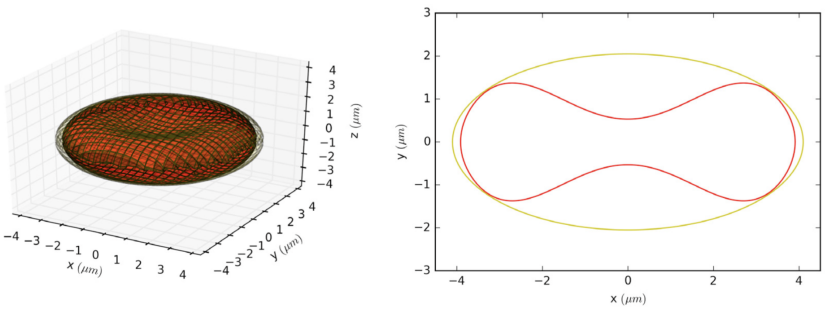


FIGURE 1.1: The shape of the red blood cell and its enclosing ellipsoid. In the left image is a 3D view of the ellipsoid (yellow) that encapsulates red blood cell (red) and right image shows the corresponding cross-sectional view.

ellipsoid, but this also means that the ellipsoid is taking up more space than actually needed. It a necessary simplification to make the computational model manageable, but it inevitably leads to a compromise in the system's fidelity. By representing the red blood cells as ellipsoids, the maximum density that can be achieved within the model is reduced. The red blood cell covers 60% of the ellipsoid volume. The maximum known ellipsoid packing density is around 80% [5], which means that the red blood cell packing with packCells using ellipsoids has upper bound of 48%. The shape of the red blood cell is visualised including its encapsulating ellipsoid in Figure 1.1.

A secondary, but still noteworthy, limitation pertains to the constraints of the packing container. The current method restricts the container's definition to a cuboid, which is not an accurate representation considering blood vessels bear a closer resemblance to cylinders. When packCells has computed a packed configuration, the desired container shape is mapped onto the cuboid. This results in cells overlapping the container, which are then subsequently removed. Regrettably, this process unnecessarily consumes a significant amount of computational resources on cells destined for removal.

The final, albeit less critical, issue stems from the force-biased approach used when packing a large number of objects. The iterative computation of repulsion forces based on overlap can leave the system stuck in a non-converging vibrational state. This creates a scenario where the objects are in constant motion, yet never reach a stable state.

The combination of these drawbacks leads to a maximum achieved packing of around 45%. The actual hematocrit value for humans ranges from 40% to 54% for men, and from 36% to 48% for women [10]. Furthermore, in case of certain haematologic diseases [11] and specialised microfluidic scenarios an even higher hematocrit is desirable. With the goal of HemoCell in mind, this is not nearly enough to create a realistic scenario, therefore there is a need for a better packing method.

The alternative way to approach this complex packing problem is not approximating the shapes with regular shapes and viewing them as completely irregular and therefore solving the packing problem for generic shapes. Of course, the shape definitions exist of a lot of data, and prevent us from using simplifying assumptions based on shape regularities, like symmetry. We would need a method that uses a high detail representation of an object and still be able to pack it efficiently. The standard way to efficiently represent a shape in data is by using a mesh, which is a combination of vertices, edges and faces, that can accurately represent the details of a 3D object.

The two main approaches to solving the irregular object packing use mesh representations of the shapes. The first approach uses 'voxelisation' to define the 3D domain and the objects in it [12], by discretising space into so called voxels, which stands for volumetric pixels. This advantage of this method is that it enables to easily compute whether 2 objects collide or not, but computing the voxelisation itself can be computationally expensive. The second approach uses the surface mesh representation of objects (vertices, edges and faces) and uses a multi-phase growth-based optimisation algorithm, where the items are initially shrunken and placed without overlap and then iteratively grown to their original size [13]. This method is doesn't have a specific name but we will refer to it as irop, which stands for irregular object packing.

This ability to tackle the most general form of the 3D shape packing problem aligns perfectly with the needs of HemoCell. Given its ambition to simulate all types of cells present in the blood, not limited to red blood cells, this approach shows potential as a fitting solution.

The aim of this study, is to implement and analyse the proposed method by Ma et al. for HemoCell as an alternative to packCells.

This study is primarily driven by the motivation to address several key challenges within the current system. The first issue is the use of an ellipsoid approximation for irregular shapes, which may not accurately represent their complex structures. Secondly, there is the hurdle of reaching maximal packing density, a crucial factor in efficient packing. Lastly, the issue of computational wastage due to an unaligned container definition originates from a clear in-efficiency in the current approach. Each of these challenges provides a focal point for our research, pushing us towards finding potential corrections and improvements. While this thesis will not attempt large scale distributed computation optimisation, we will conduct performance measurements and profiling to make a projection of the potential benefits of a fully optimised implementation. This exploration could pave the way for a more versatile and efficient packing methodology, aligning better with HemoCell's objectives.

The rest of this thesis is structured as follows. Chapter 2 will discuss the background of packing algorithms and some of the crucial methods of polygon-based packing. Chapter 3 will discuss the methods and implementation of the irregular object packing methods central to this thesis. The results of the implementation and its performance are shown in chapter 4. Chapter 5 will discuss the results and finally chapter 6 will conclusions and the future work.

Chapter 2

Background

In this chapter we will discuss the current state of the art in packing algorithms. We will start by discussing the different types of packing problems, and then move on to the different approaches to solve them. Finally, we will touch on the topic of Constrained Delaunay Triangulation (CDT), which is a key component in the algorithm we will be using in this thesis.

2.1 Packing Algorithms

The packing problem is known under different names, such as the shape stock cutting problem, nesting problem, knapsack problem[14], etc. However, the central question remains the same for all: How can one find the most efficient configuration of objects in a container. In Figure 2.1 three possible ways to configure different shapes into a container is shown. Still, the problem has many variations, which in turn may require different approaches. We can categorise them according to properties: non geometrical packing, regularly shaped polygonal packing, and irregularly shaped polygonal packing. Belonging to the first category, the knapsack problem aims to maximise the total value of items placed into a "knapsack" without exceeding its capacity. Each item has a specific weight and value, and the problem involves determining the most valuable combination of items to include given the weight constraint. This type of problem will not be discussed further.

Both regular and irregular shape packing exist in 2 and 3 dimensions. Even the most basic packing problem, the bin packing problem, is NP-hard, which is a class of mathematical problems for which any problem in Nondeterministic Polynomial time (NP) can be reduced to in polynomial time, making them at least as hard as the hardest problems

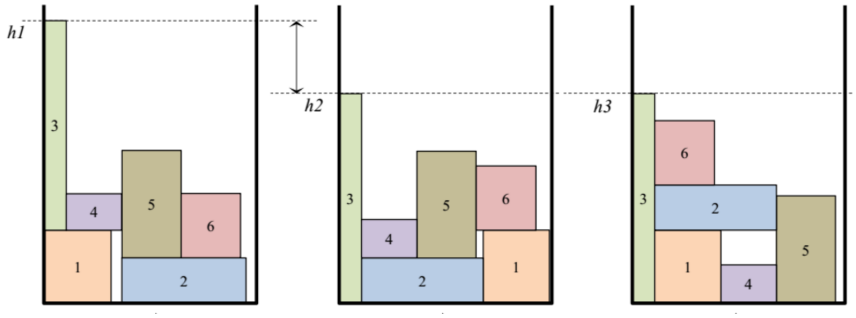


FIGURE 2.1: Example of a 2D strip packing problem [1].

in Nondeterministic Polynomial time. In the case of packing, as the number of items increases, the number of potential packing configurations grows exponentially, which makes it difficult to find the best solution in a reasonable amount of time. Nevertheless, Sophisticated algorithms[15] are able to produce optimal solutions for very large numbers of items and heuristic algorithms [16] can produce close to optimal solutions within application-relevant time frames.

2.1.1 Regular Shape Packing

For regular shape packing, we consider packing of shapes that have special geometric properties, like symmetry or being equilateral, that can be leveraged to compute the optimal packing. In 2D these are circles, ellipses, rectangles and regularly convex polygons (triangles, squares, hexagons, etc.). In 3D, these are spheres, ellipsoids, cubes, cuboids, tetrahedra, octahedra, cylinders and prisms. In 2 dimensions, The strip packing problem, which is arguably the most well-known among packing problems, will serve as *the trivial packing problem* as reference for the rest of this thesis. When we want to pack 3 dimensional objects, this problem becomes a lot harder due to the added degrees of freedom in translation and rotation.

2.1.1.1 Strip Packing Problem

The strip packing problem aims to pack a given set of rectangles into a strip of fixed width and infinite height in such a way that the overall height of the strip is minimised. Imagine you have a strip of material of fixed width (think of it like an infinite roll of wrapping paper). Your task is to pack a number of rectangular items onto this strip in such a way that you use as little of the strip's length as possible, while adhering to the fixed width. Each of the rectangles can have different dimensions, but they cannot overlap and they must be entirely contained within the strip. The rectangles can't be rotated, so their orientation is fixed. Figure 2.1 is an example of the 2D strip packing problem.

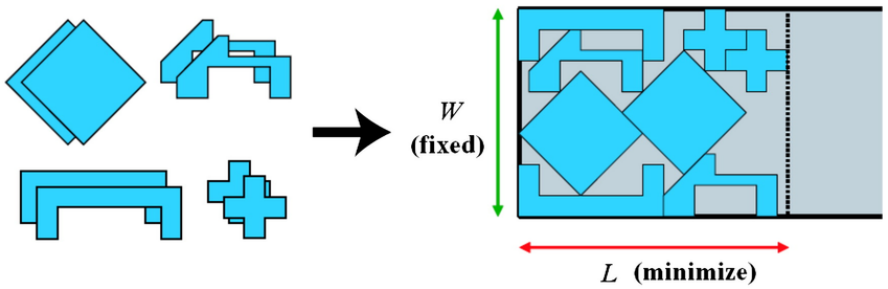


FIGURE 2.2: Example of a 2D irregular shape strip packing problem [2].

The problem is geometric in nature as it involves packing rectangular items onto a strip. Most of the attempts to find a solution to this problem, use heuristic algorithms, however there are a few that use an exact approach, that can reach optimality [16] or predict good lower bounds for heuristic approaches [17].

2.1.1.2 General Approaches

Approaches to tackle the strip packing problem include the First Fit Decreasing Height (FFDH) algorithm, the Best Fit Decreasing Height (BFDH) algorithm, and various evolutionary algorithms or local search methods [18]. The general approach is to first sort the items by size and (optionally) categorise them, before starting packing.

2.1.1.3 3D Problems

For 3D regular shape packing, it is common to start off from a 2D strip or bin packing problem and generalise this approach to d dimensions [19]. When the height of each item in the 3D strip packing problem is equal to 1, it is identical to the 2D bin packing problem, and when the width of each item is equal to 1, it is identical to the 2D strip packing problem. This are the basic building blocks for deriving multidimensional solutions.

2.1.2 Irregular Shaped Packing

Irregular shapes are those where the geometric properties of the shape are not trivial and therefore we cannot make any assumptions about them. An example of a 2D irregular shape packing problem is shown in Figure 2.2 For this reason, any algorithmic approach to solving this problem, must be shape-agnostic algorithm. This means that the algorithm must be able to handle any shape, regardless of its geometric properties. For shapes with just a few vertices, There are well performing algorithms proposed

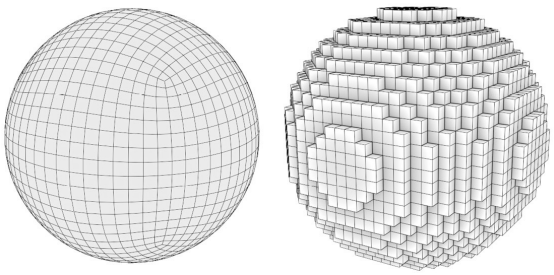


FIGURE 2.3: Voxelisation of a 3D surface mesh of a sphere.

by [20]. However, they do not scale well with shape complexity. For irregular shapes with many vertices there are 2 main approaches The voxel-based approach and the polygon-based approach. The implementation made for this thesis is categorised as the latter.

2.1.2.1 Voxel-based Approach

Voxelisation is a common approach in geometric computer simulations, as it allows for a high level of detail by the discrete representation of the shape as small uniform cubes, known as voxels (short for volumetric pixels). With regards to computation, this means that the data is well structured and easily accessible. These two properties are specifically useful when one is trying to detect collisions between objects, as the collision detection can be done by checking if the voxels of the two objects overlap. This is a very efficient way of detecting collisions, as it is a simple boolean operation that can be done in parallel. Nonetheless, while voxelisation has proven to be a useful approach in geometric computer simulations, it is not without its drawbacks. The process necessitates the storage of a uniform 3D grid of voxels, which imposes significant memory demands. Furthermore, transforming a high-resolution mesh into a voxelised representation is a computation-intensive process. Another inherent limitation of voxelisation is its block-shaped nature, which proves inadequate in accurately representing smooth surfaces. Yet, despite these hurdles, some successful solutions have emerged that employ voxelisation to solve the packing problem [12].

2.1.2.2 Polygon-based Approach

an alternative approach than the voxel-based techniques, has been proposed by Ma et al. [13], which utilises the surface mesh representation of objects. This method is based on a 2D surface mosaic method [21], which is then extended to 3D. The proposed method uses surface meshes for both the objects to be packed and the container. Unlike any other approach, this means that the container can also have an arbitrary shape, not just

cuboids. In irop's methodology, they combine volumetric mesh generation with method known as the 3D Chordal Axis Transform (CAT)[22]. The CAT creates separated regions within the container called 'CAT cells', opening the door for local optimisation for each object. This approach boasts several advantages. First, It requires a lot less memory, because we do not need to represent each voxel of the grid. Secondly, The use of a surface mesh representation enhances the accuracy of shape representation and, quite importantly, it holds the capacity to accommodate any shape that can be described by a mesh within any container, irrespective of their complexities. Briefly, the algorithm shrinks objects and places them in the container in a non-overlapping configuration. Then, the objects are grown iteratively in their corresponding independent space defined by the CAT cells, until the original size is reached. The detailed mechanics of the algorithm are described in Section 3.1. When it comes to computation, a key difference between the voxelisation approach and the currently presented polygon-based approach is that the latter doesn't need to perform any collision detection during the optimisation process. This is because the objects are grown in their own independent space, and therefore cannot collide with each other. This is a significant advantage, as collision detection is a computationally expensive process. We will look into this proposed method in more detail in this thesis.

2.2 Triangular Mesh Generation

Triangular mesh generation serves as the foundational stage of many 3D imaging techniques, material analyses, and 3D printing processes. It involves the creation of a two-dimensional surface mesh comprised of triangles, which underpins the structure of the more complex 3D mesh. The quality of this triangular mesh is paramount, with high-quality meshes largely devoid of sharp triangles. In essence, the fewer the minimum angle present in the mesh, the better the quality. However, achieving a balance between high quality and a low number of elements can be a challenging endeavor. This is due to the often conflicting goals of quality and simplicity in the generation process.

2.2.1 tetrahedral mesh

One core component of the algorithm is the generation of a tetrahedral mesh from a set of input constraints. A Tetrahedral mesh is a 3D volumetric mesh existing of tetrahedra only. These meshes are used in all sorts of 3D imaging techniques, structural material analysis and in 3D printing. The process of generating a tetrahedral mesh is far from trivial however. It has been studied for multiple decades[23] and is still an active area of

research[24]. The main challenge is to generate a mesh that is both of high quality[1] and has a low number of elements. This is a challenging task, as these two goals are often conflicting. For example, a mesh with a low number of elements can be generated by simply dividing the space into a grid of cubes, however this will result in a mesh with a very low quality. On the other hand, a mesh with a high quality can be generated by using basic Delaunay triangulation, however this will result in a mesh with a significantly higher number of elements than originally present in the provided input.

2.2.2 Delaunay triangulation

The Delaunay Triangulation (DT) algorithm, a cornerstone to many applications in the field of computational geometry, involves triangulation of a set of points while maximising the minimum angle across all triangles in the resulting triangulation, resulting in a high quality mesh.

In a 2D space, Delaunay Triangulation creates triangles such that for each triangle there is no exterior point inside its circumcircle. Figure 2.4 shows a schematic of the difference between a Delaunay Triangulation (a) opposed to a non-Delaunay Triangulation (b) of 4 points. The Delaunay property is a property that guarantees high quality meshes, thereby avoiding 'skinny' triangles which for example could degrade the accuracy of a finite element method (FEM) simulation or any similar numeric computations.

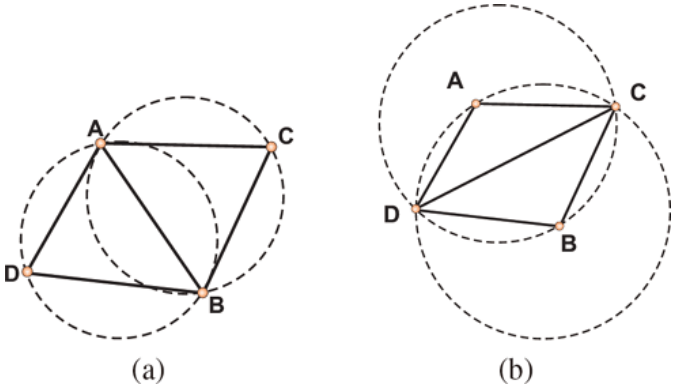


FIGURE 2.4: Delaunay (a) and non-Delaunay (b) triangulation of a set of points in 2D. in schematic b, point A is inside the circumcircle of the lower triangle and point B is in the circumcircle of the upper triangle, making them both non-Delaunay. Retrieved from [3].

In 3D, the concept extends to tetrahedra, with each tetrahedron's circumsphere containing no other point from the set, leading to the creation of a so-called Delaunay tetrahedralisation. This tetrahedralisation method typically results in high-quality meshes but may generate a large number of elements, thus increasing computational complexity.

[1]The quality of a mesh is defined by the shape of its cells. For triangular mesh, this means that a low quality mesh has a lot of sharp triangles, and a high quality does not.

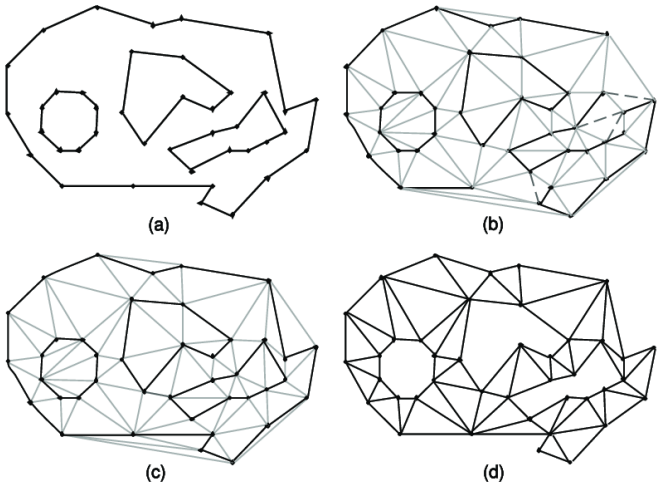


FIGURE 2.5: The four states of CDT in a simple vague region, with (a) the constraints, being the region boundaries, (b) Delaunay triangulation of the vertices, (c) insertion of the missing lines and (d) removal of triangles outside the boundary and inside holes [4].

2.2.3 Constrained Delaunay Triangulation

Constrained Delaunay Triangulation (CDT), a key adaptation of the initial Delaunay Triangulation, offers increased flexibility and use by adding the capacity to satisfy initial conditions.

With CDT, the triangulation is subject to constraints, being edges and facets (in 3D case) that are provided as part of the input to the triangulation. These predefined constraints can for example be the outline of shape that must appear in the final triangulation. This allows the algorithm to respect specific boundaries or interfaces between different materials, or to ensure that certain features of the geometry are captured accurately in the mesh. The process is displayed in Figure 2.5.

While the introduction of constraints complicates the process somewhat, as one must now ensure that the constraints and the Delaunay property are both satisfied, it offers a powerful method for controlling the mesh's structure and the placement of elements. This added control can be exploited to improve the mesh's quality or reduce its element count, helping balance the often conflicting goals of quality and efficiency in mesh generation. For example, constraints can be used to guide the partitioning process, so that elements are placed more strategically and efficiently, leading to a decrease in the total number of elements. Additionally, by carefully selecting and positioning the constraints, one can guide the triangulation process to produce elements with better shapes, improving the mesh's quality.

Chapter 3

Methods

In this chapter, we will elaborate the polygonal based approach described Section 2.1.2.2 and how it has been implemented. Then, we shortly describe how the data collection has been done. Finally, we will describe the evaluation metrics used to evaluate the results of the algorithm.

3.1 The Irregular Object Packing Algorithm

The heuristic optimisation algorithm described in Ma et al. [13] exists of a set of 6 steps, where the goal is to increase the coverage rate slightly by for each phase. The outline of the main steps of the algorithm are as follows. (1) A set of initial coordinates and rotations is generated based on the predefined volumetric coverage rate. Then, all the instances of shapes, which we will refer to as objects, are scaled down to the initial scale, f_{init} and are placed inside the container. (2) The continuous optimisation phase starts. Here, the scale is iteratively increased until the final original scale is reached. Each iteration, the volume encapsulated by the container is split into separate non-overlapping regions for each object, in which that object is then individually scaled. (3) Now, the algorithm finds and swaps objects, that fit better in the CAT cell of the peer. (4) it tries to individually grow objects to an a version of the CAT cell that extends to the surface of adjacent objects rather then the midsection. From here, the algorithm has a few different approaches to increase the coverage rate further. (5) The algorithm increasing the coverage rate by replacing objects that are not fully grown with other new shapes and, thereafter, by filling empty leftover spaces still present in the volume with a new matching object. The algorithm is described in more detail in the following subsections.

3.1.1 Initialisation

To start the algorithm, we need an initial configuration. For this, it is necessary to generate a set of initial coordinates and rotations of the objects. The algorithm starts of with a starting coverage rate, r, that determines the number of objects that are being initialised in this phase. The paper describes the use of the centroidal power-diagram method, which is a way to distribute the coordinates in space while considering a weight, in this case being the relative size, for each object. This creates initial configuration for shapes of different sizes, that do not overlap each other for the initial scale. The initial rotations are generated by randomly rotating the objects around the x, y and z axis.

3.1.2 Continuous Optimisation

The continuous optimisation phase is the main part which we will look at in this thesis. This phase starts by scaling all objects down to the initial scale, f_{init}. Then, the algorithm iteratively increases the scale of the objects until the original size is reached. This process works with an outer loop, where each iteration is referred to as a **scaling step**, and an inner loop, where each iteration is referred to as an **iteration**. The inner and outer loop are shown in the flowchart in Figure 3.1. The outer loop prevents one object growing faster than the other and leaving no space for other objects to grow. For each iteration of the inner loop, the algorithm splits the volume into separate non-overlapping regions for each object, in which the scale of the object is maximised. The objects that have surpassed the maximum scale of the scaling step, will be reduced to that maximum scale. Then, the scales of objects are evaluated. If all reached the scale limit, the algorithm continues to the next scaling step. If not, the algorithm continues to the next iteration. The algorithm stops when all objects have reached the final scale.

3.1.2.1 Compute CDT Volumetric mesh

This step starts with a specific configuration of the objects, either from the initialisation, or from the previous iteration. The first step in the continuous optimisation phase is to compute a tetrahedral mesh from all from the objects meshes. Here, we use CDT described in Section 2.2.3 In this step, the input to the CDT is the sum of all the object meshes, including their vertices, edges and faces. Based on these 'constraints' we can now assume that we have a mesh that contains all the objects and that their are no tetrahedra crossing multiple objects[potentially add schematic/figure here]. This is essential for the next step, where we will split the volume into separate non-overlapping regions for each object.

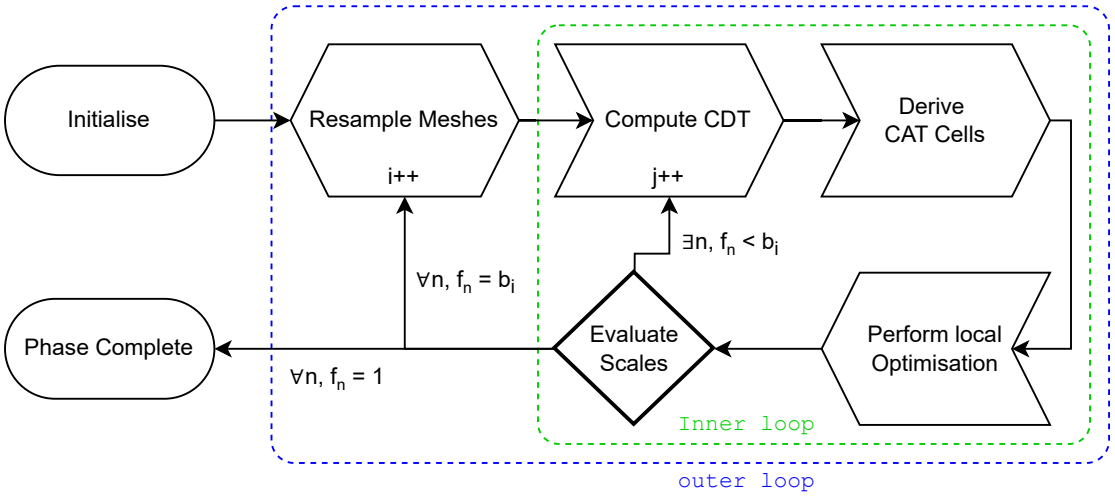


FIGURE 3.1: Flowchart of the continuous optimisation phase, where f_n refers to the scale of object n, b_i is the scaling limit of scaling step i, j is the inner-loop iterator and 1 is the final scale of the objects ($\forall k$ stands for 'for all' and $\exists k$ stands for 'for some'). The outer-loop has a max number of iterations which will also lead to a loop exit.

3.1.2.2 Derive CAT Cells

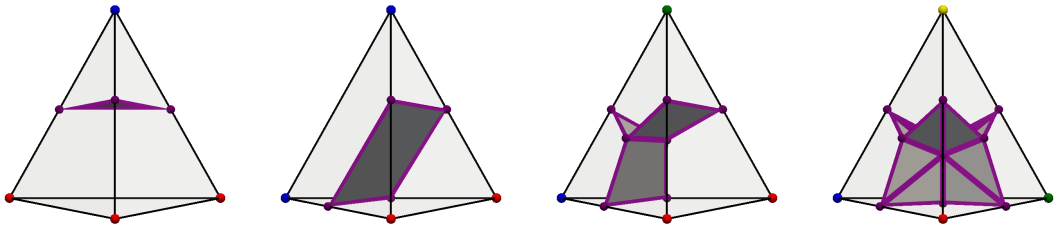
The volumetric mesh we've generated possesses some key characteristics: it's composed entirely of tetrahedra, all input shapes are completely encapsulated within the mesh, and no additional shapes beyond the input meshes have been added. These features allow us to generate distinct, non-overlapping *cells* for the objects using a process called the Chordal Axis Transform (CAT).

In essence, CAT involves creating segments separates vertices from different objects from each other. The faces separating these segments are termed CAT Faces. Combining all the CAT faces associated with the one object gives us a CAT cell that is manifold - a term that indicates a surface mesh is fully enclosed. This is a necessary condition to determine whether a point is inside a surface mesh.

An important detail to note is that during this process, it must be administrated which face correlates to which vertex. This association is necessary when the constraints of the local optimisation phase are computed, where only the faces related to a specific vertex are considered.

In 2D, the CAT process involves separating only 2 types of triangles, one where all the vertices belong to a different object, and one where two vertices belong to one object and the other vertex belongs to another object. In 3D, the process is more complex, as there are four combinations of vertices. These combinations are illustrated in Figure 3.2.

Let's examine the four cases:



(A) (B) (C) (D)

FIGURE 3.2: Chordal Axis Transform split variations. Each vertex of a tetrahedron is colored with respect to the object they belong to. The black edges are the original edges and the purple vertices and edges are new. Subfigure (a), (b), (c) and (d) refer to case (*aaab*), (*aabb*), (*aabc*) and (*abcd*) in table Table 3.1 respectively.

- Case *(aabb)*: Two vertices belong to one object, and the other two belong to a different object, leading to two segments separated by one face.
- Case *(aaab)*: Three vertices belong to one object, and one vertex belongs to another object, resulting in two segments separated by one face.
- Case *(aabc)*: Two vertices belong to the same object, and the remaining two vertices each belong to different objects. This arrangement yields three segments, separated by three faces.
- Case *(abcd)*: All four vertices belong to different objects, resulting in four segments separated by twelve faces.

Case	**Vertex Count**	**Segment**	**Faces**
(aaab)	(3,1)	2	1
(aabb)	(2,2)	2	1
(aabc)	(2,1,1)	3	3
(abcd)	(1,1,1,1)	4	12

TABLE 3.1: Summary of Chordal Axis Transform (CAT) Cells Segmentation Cases, where vertex count refers to the number of vertices per object.

3.1.2.3 Growth Based Optimisation

By modifying the rotation and translation, we maximise the scale of each individual object inside its CAT cell, without exceeding the CAT boundary. By describing this step as a non-linear constraints problem with bounds, we can use optimisation algorithms to find the best fit. The parameter to optimise for is the object transformation with the objective being maximum scale. The constraints are the distances of the vertices to their related CAT faces, and finally, the transformation is bounded by a maximum rotation angle, $\theta_{\max}$. The maximum allowed rotation deserves some explanation. As mentioned, the constraints for every point are computed for only a small set of the actual CAT

faces. Only those, that are connected to the point, are considered. This means that the constraints for a vertex of the object are not computed for all the faces of the CAT cell. This is reduces the amount of computation by not computing the constraint for all faces for each point, However if these points are allowed to be rotated without limit, this might lead to the transformation being valid, while the CAT cell is actually violated. This is illustrated in figure [FIG]. According to Ma et al. [13], limiting the rotation to a maximum guarantees that the CAT cell will not be violated. However, there is no mathematical proof of this method being watertight. The objective, constraints and bounds are described in equations 3.1, 3.2 and 3.3 respectively, where F is the function that computes the scale, f, of the object, T is the transformation matrix, v_i is the ith vertex of the object, q_j is the jth vertex of the CAT face, $\vec{n}_j$ is the normal vector of the jth CAT face, $w(v_i)$ is the set of vertices of the CAT face that is related to v_i and θ_x, θ_y, θ_z, t_x, t_y, t_z are the rotation and translation parameters of the transformation matrix T.

$$\max F\left(f, \theta_x, \theta_y, \theta_z, t_x, t_y, t_z\right) = f, \tag{3.1}$$

$$\text{s.t. } \frac{\left(T \cdot v_i - q_j\right) \cdot \vec{n}_j}{\|\vec{n}_j\|} \geq 0, i = 1, 2, \ldots, n, j = 1, 2, \ldots, w\left(v_i\right), \tag{3.2}$$

$$f, t_x, t_y, t_z \in R^+, \theta_x, \theta_y, \theta_z \in \left[-\frac{\pi}{12}, \frac{\pi}{12}\right] \tag{3.3}$$

Adaptive Mesh Sampling

As shown in Figure 3.1, each scaling step starts with re-sampling the meshes. The re-sampling itself, doesn't contribute to the mechanics behind the algorithm, however it is an important step that improves efficiency. The reason for this is that computing the tetrahedral mesh has a $O(n \log n)$ average run time and the number of vertices in the mesh is directly related to the number of CAT faces, which in turn is related to the number of constraints that are being computed in the local optimisation phase. On the other hand, down-sampling a mesh can result in loss of features[1], which may lead to incorrect optimisation results. In the first few iterations, the objects are small and not that close to each other so the loss of features will not lead to incorrect results. However, as the objects grow and the space between objects becomes tighter, the lack of represented features will result into more issues. To solve this, an adaptive mesh sampling method is introduced, where the re-sampling factor is more aggressive initially and becomes more conservative as the objects grow. The re-sampling factor is defined

[1] A feature is a part of the mesh that is important for the shape of the object. For example, a hole in the mesh is a feature.

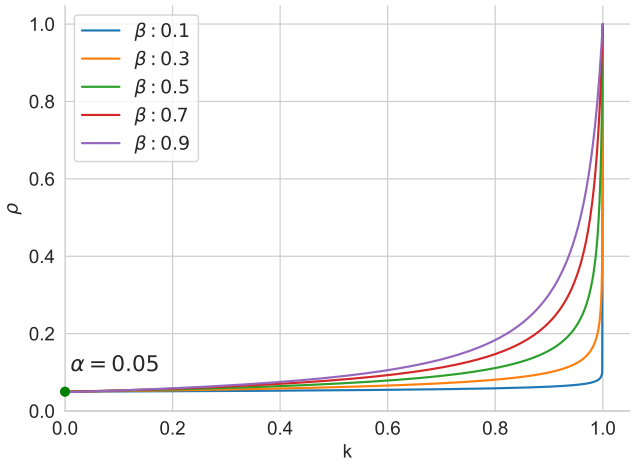


FIGURE 3.3: Re-sampling factor as a function of the iteration number.

as the ratio between the number of vertices in the original mesh and the number of vertices in the re-sampled mesh. The re-sampling factor is defined in (3.4) and plotted in Figure 3.3.

$$\rho(k) = \alpha(1 + \alpha^{\frac{1}{\beta}} - k^{-\beta}) \tag{3.4}$$

3.1.3 Escaping Local Maxima

In the continuous optimisation phase, the goal is to reach as close to the original scale as possible. As that phase has finished, the configuration of objects has reached a local maximum. The authors of Ma et al. [13] have described a few methods to escape from that local maximum, however this is outside of the scope of this thesis. A description of these steps is provided for completeness.

Object Swapping

The first method to step out of this local maximum is to swap objects. By including a permutation of objects the algorithm can explore more of the search space. This is a common approach to escape local maxima in packing problems. In this step, the isolated boundary is not the CAT cell, but rather the CDT cell, which extends itself to the boundaries of adjacent objects. For each object, a suitability score is computed with respect to the CDT cells of other objects, which is then used to build a bipartite graph. The bipartite graph is used to find the best swap. The suitability score is defined by the maximum scale that an object can reach in a CDT cell as a result of a slightly

modified version of the continuous optimisation. The optimisation is different in that it doesn't recompute the CAT cells in between growth iterations and that it is repeated for a set of random initial orientations. By splitting the objects into non-neighbouring groups and then performing the swap algorithm for that group, the algorithm prevents adjacent objects from overlapping.

Object Enlargement

Now, some of the objects may not have reached their original scale and for these objects, the modified optimisation is executed with the CDT cell as boundary. The CDT cells of adjacent objects overlap, so this step can only be performed for non adjacent objects.

Object Replacement

Just like the object swapping phase, the object replacement phase, swaps objects, however it does so with new objects that haven't been introduced into the container. The aforementioned suitability score is used to find the best replacement, whereafter the continuous optimisation phase is executed again. Here, the fully scaled objects are protected from scaling down.

Hole Filling

The last step of the algorithm is to fill the holes that are left in the packing. These so called holes are vacant spaces left inside the container. This step allows for the increase of the final coverage rate. To detect holes in the container, all the already packed objects are merged and subtracted from the container volume. The resulting volumetric mesh will contain only tetrahedra that contain free space. Now, the circumsphere for each tetrahedron is computed and based on the size if those spheres, a spherical non-overlapping space can be defined and a sphere is placed inside. The CDT cell of this sphere is then used to find unpacked objects with corresponding shapes and the new objects are then locally packed in the CDT cell. The objects that haven't reached their original scale after this step are discarded. The algorithm ends here and the final packing is returned.

3.2 Implementation

In this section, the implementation written for this thesis project is described. This implementation tries to follow the description given by Ma et al. [13], while taking implementation decisions where necessary. These choices and reasoning behind them will be explained. Given the aim of packCells of HemoCell, which is to pack red blood cells with a realistic hemocrit, the focus of this implementation is primarily centered on the optimisation phase, described in 3.1.2. These specific steps are visually set out in Figure 3.1. For this reason, this thesis does not cover the object swapping and object replacement phase described in Section 3.1.3 and 3.1.3, as they are only relevant for packing of multiple different shapes, and it does not cover the object enlargement and hole filling phase, as they are less critical.

The implementation is written with python 3.10 and it makes use of Numba [25] for Just-In-Time (JIT) compilation for specific speedups. The code is written in a modular way, which allows for easy extension. The code is available on github[2].

Framework and General Approach

Starting with a birds eye view, we are trying to perform a computer simulation on a large amount of detailed mesh objects. The storage of all these individual objects at full resolution, will require a lot of memory. In the context of HemoCell, the objects which we are trying to pack all have the same shape, so we can actually define a object state by solely storing the transformation array, described in Equation 3.5, where f is the scale and $\theta_{x,y,z}$ and $t_{x,y,z}$ define the rotations and the translations with respect to the axes. The translation (t_x, t_y, t_z) is equal to the coordinate of the center of an object. The parameters in this array are all scalars.

$$\vec{t} = \begin{bmatrix} f & \theta_x & \theta_y & \theta_z & t_x & t_y & t_z \end{bmatrix} \tag{3.5}$$

By doing this, we don't store the details of the objects during the optimisation process, and we only compute them when needed. This doesn't reduce the total memory requirements, as we need all the mesh details as input for the triangulation described in Section 2.2.3, however it is an important implementation detail for what follows.

[2] https://github.com/MbBrainz/irregular-object-packing

3.2.1 Initialisation

Considering HemoCell, the described initialisation phase is over-complicated, as we focus on objects with equal sizes and shapes. Therefore, we can use a simpler method to generate the initial configuration. The objects are initialised with a random rotation and the coordinates are generated by randomly generating coordinates within the container. For each newly generated coordinate, we check whether it is at least as far as the bounding radius of the object away from all other objects and the container. If this is not the case, the coordinate is discarded and a new one is generated. The condition is as follow:

$$|(c_i - c_j)| \geq \frac{1}{2} R_i * f_{\text{init}} \text{ for all } j, \tag{3.6}$$

where c_i and c_j are generated coordinates of the 'to be added' (i) and the already present (j) objects, R_i is the radius of the closest enclosing sphere of object i and f_{init} is the starting scale of the object. This process is repeated until the desired number of objects is reached. The necessity of this initialisation may not be obvious at first, However, it is required to ensure that the initial configuration doesn't have any overlapping objects, as this would cause the algorithm to fail. This method is far less involved and computationally expensive than the one described in Section 3.1.1 and it serves purpose of this project. 1

3.2.2 Continuous Optimisation

The continuous optimisation phase is the most important part of the algorithm and it exists of a set of steps described in Section 3.1.2. The implementation requires understanding of each step separately, as well as the interactions between the steps.

This phase starts with computing the 3D CDT. First, we combine all the surface meshes of the objects and the container, where it is essential to input not only the vertices, but also the faces and edges, hence *Constrained*, to get the desired input. From this combined mesh, we generate the volumetric mesh. CDT generation is a topic that has been analysed and worked on for many years. The industry standard implementation for CDT generation is the C++ library called TetGen [26]. TetGen is a highly optimised library and for the scope of this paper, it is considered optimal. The library is written in c++, however, it has a python wrapper, that is used for the implementation of this thesis.

Besides uniform sampling, We implement a custom container sampling method to improve the CDT mesh quality. The method sets the average face surface area of the container equal to that of the input mesh. The formula is shown in Equation 3.7, where A_{avg} is the average surface area for the faces container, subscript c, and object, subscript o.

$$N_{\textbf{target faces}} = N_{faces} * (A_{avg,c}/A_{avg,o}) \tag{3.7}$$

3.2.2.1 CAT Cells

We now have a volumetric mesh, existing solely of tetrahedra, which has all the vertices, edges and faces originating from the objects and the container, and the new edges and faces interconnecting them. Now, the goal is to create CAT cells, which are surface meshes that each encapsulates an object and do not overlap other CAT cells.

The first step is to identify those tetrahedra, that are connecting the vertices of more than one object. We create a list of vertex sets for each object. A vertex set is a finite set with unordered unique values. For such a set, we can check if any vertex v_i is part of a set or not in $O(1)$ time [27], which allows us to iterate over all tetrahedra and then for each vertex of a tetrahedron figure out efficiently which set, and therefore which object, it belongs to. Based on these findings, we can categorise the Tetrahedron in one of the cases mentioned in Table 3.1.

Once we know which object a tetrahedron belongs to, we can compute the faces that split it in separate segments. We need to register the faces for the corresponding vertex in order to compute the right constraints in the next step. In the initial implementation, a dictionary is used with the keys and values being the vertex id and an append-able list of of corresponding faces as the data-structure to store. This allows us to easily access the faces that belong to a certain vertex. However, there are 3 reasons which make this data format hard to efficiently compute with. Firstly, iteration over dictionary key-value pairs are convenient but not efficient. Secondly, the amount of related faces per vertex is not constant, and thirdly, the faces are defined by either 3 or 4 vertices, preventing us from storing this data in memory contiguously.

3.2.2.2 Growth Based Optimisation

A better understanding of Equation 3.2 allows for improvement of the previously used data-structure. The goal is to find a new transformation that increases the scale of the object, while staying within the boundary defined by the CAT cells. The current state of

the object is defined by the transformation array from the previous iteration and we have a list of all the related faces per vertex from the previous step. The optimisation will try to find a better value for v_i by changing T, which means that during the optimisation, v_i, q_j and n_j are constant. Where q_j can be any point on the surface of the object and n_j is the normal vector of the face, we can simply compute the unit normal vector and store that together with the first vertex of the face. Now, during the optimisation, we do not need to compute the unit normal vector for each vertex, or lookup a vertex from the face, but we can simply use the stored values. There is one drawback of this approach, which is that we store a (face, vertex) pair twice, once for each sides of the face, and instead of storing the vertex id, the vertex itself is stored. Nonetheless, this is a contiguous way of storing the data and allows for efficient computation. The final data structure used is shown below:

$$\begin{pmatrix} \vec{v_i} & \vec{q_j} & \hat{n_j} \end{pmatrix}_{3\times 3} \tag{3.8}$$

As the output is now a list of containing this data-structure, we can easily convert it to a contiguous numpy array, before we start the optimisation phase. Not only do numpy arrays benefit from general numpy operations, it enables Numba, a Just-In-Time (JIT) compiler for python, to compile the non-linear constraint computations and numpy operations so that they can be executed as compiled code during the optimisation phase.

With regards to the computation of the constraints, there is a subtle improvement that can be made. In Equation 3.2 the numerator is divided by the length of the normal, $|\vec{n_j}|$. As we are pre-computing the face normal during the creation of the CAT faces, we can also normalise compute the unit vector of this face normal, which simplifies the constraint computation to 3.9. This may seem like a neglect-able improvement, however the difference is significant, because the optimisation algorithm is an iterative process, and the constraints are computed for each optimisation iteration, while computing the normalised normal vector only once per scale iteration.

$$(T \cdot v_i - q_j) \cdot \hat{n}_j \tag{3.9}$$

3.3 Error Detection and Correction

There have been a set of interesting issues regarding our implementation of the proposed algorithm in Ma et al. [13], which are addressed in this section. Firstly, The input constraints were not correctly provided to the CDT algorithm, which led to incorrect

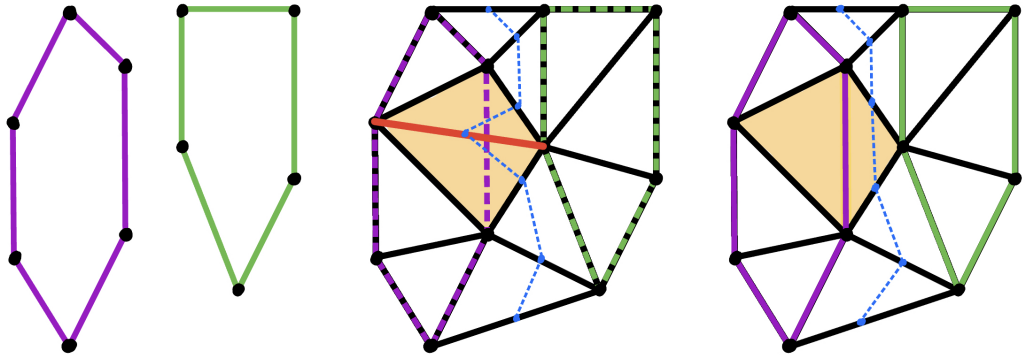


FIGURE 3.4: A schematic representation of results of DT and CDT. The purple and green polygon on the left are triangulated without and with constraints. The blue dashed CAT lines in the non-constrained version, crosses the original object boundaries and is therefore incorrect.

CAT cells. Secondly, the implementation is not fully correct, leading to unintentional violations of the CAT cells, during optimisation. For both of these issues, we will provide a description and a (partial) solution.

3.3.1 Input constraints for CDT

The construction of a high quality volumetric mesh from a set of input points has been a well studied topic. The most efficient way to do this is using CDT, which is described in detail in Section 2.2.3. When we provide initial edges to the algorithm, those will be considered as *constraints* which will be enforced in the triangulation. With regards to creating CAT cells from the resulting mesh, the constraints play an essential role, because without these, the boundaries of the input meshes will not be respected. Now the description of the algorithm in Ma et al. [13] states that "We first uniformly sample points on the surfaces of the container and objects, [...], we use these sampling points as the input to the CDT". However, following this exactly this description will lead to incorrect CAT cells, as the constraints are not correctly provided to the CDT algorithm. A 2D schematic of this in Figure 3.4, where the center mesh shows the CDT and the resulting incorrect CAT lines and the right mesh shows the correct result. The complete description would be to include all the mesh information of the objects and the container, which include the edges and faces. The CDT algorithm will then use these edges and faces as constraints, which will lead to the correct CAT cells.

3.3.2 CAT cell violation

Another issue that occurs during the implementation, is that the CAT cells are regularly exceeded by a few vertices of the objects after local optimisation. As long as this doesn't

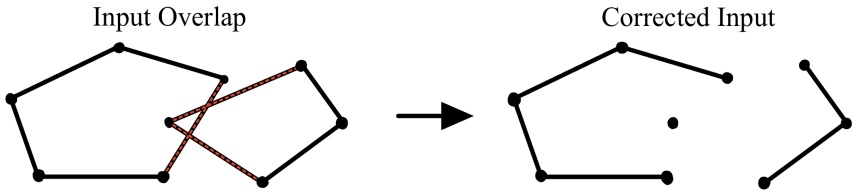


FIGURE 3.5: A 2D schematic representation of overlapping objects (left) and the input that is interpreted by CDT (right)

lead to objects intersecting with each other, this is, although incorrect behaviour, not problematic for the correctness of the configuration. However, when this does lead to intersecting objects, the configuration is incorrect and continuing the optimisation phase devolves into incorrect states.

If the input configuration is incorrect, the overlapping constraints, being faces or edges, will simply be ignored by the CDT algorithm, which leads to similar results as described in the previous subsection. Figure 3.5 shows a schematic representation if this situation. The CDT will generate a incorrect mesh, which corrupts the computation of the CAT faces.

According to the algorithm, this overlap should not happen, however in practice, tolerances and rounding errors may lead to this situation. The algorithm does not provide a solution for this, and therefore we need to implement a solution ourselves.

3.3.3 Post iteration correction

By implementing an error correction strategy, we can correct an intermediate wrong state, and continue with the computation without completely wasting the previously performed scaling steps. We do that by performing collision detection for all two combinations of objects and all objects with the container and then shrink those colliding objects. If we perform this action until there are no collisions left, we are back to a correct state. This step would be inserted in Figure 3.1 between *Perform Local Optimisation* and *Evaluate Scales*. The adjusted algorithm is shown in Algorithm 1. Even though this defeats the idea that collision detection, which is computationally intensive, is not necessary for the methods proposed by Ma et al. [13], this allows us to make a more educated estimation of the performance of this method with respect to the HemoCell packing for a completely correct implementation of the continuous optimisation at a later stage of this project.

Padding

The aforementioned validation method adds quite some additional computational overhead and will therefore lead to an longer execution time. To minimise the number of corrections, we implemented a padding factor to the algorithm. The padding should be of comparable size to the distance of the vertices, therefore we use the square root of the average surface area of the faces of the object mesh. By introducing this padding, we reduce the number of CAT cell violations per iteration, with the goal to reduce the overall execution time. Equation 3.9 is modified slightly by subtracting the padding from the constraints. In this way, A small positive constraint value will now become slightly negative, and will be rejected by the minimisation optimisation. The modified equation is written down in Equation 3.10 and a final overview of the parameters of the algorithm can be found in Table 3.2.

$$(T \cdot v_i - q_j) \cdot \hat{n}_j - p \tag{3.10}$$

Algorithm 1: Continuous Optimisation including error mitigation

Input : A container surface mesh C, a set of objects $\{B_1, B_2, \ldots, B_m\}$ in C, scaling factor barriers $\{b_1, b_2, \ldots, b_k | b_i < b_{i+1}\}$, the maximum iteration number for each b_j, $itn_{\max}$

Output: A compact layout for packed objects in C

Shrink objects to 10% of their original size;
for *each* b_i **do**
    $itn = 0$;
    while $itn < itn_{\max}$ **do**
        `ComputeCAT()`;
        for *each object* B_j **do**
            `ApplyGrowthOptimisation(`*compute the largest* B_j *in its CAT cell with a scaling factor* f_j*, a three dimensional rotation transformation* θ_j *and a translation* t_j`)`;
            `ProcessIteration(`*Find collisions and shrink the colliding objects until corrected*`)`;
            if *scale rate* $s_j f_j < b_i$ **then**
                `ApplyTransformation(`f_j, θ_j, t_j *to* B_j`)`;
            else
                `ApplyTransformation(`$b_i s_j, \theta_j, t_j$ *to* B_j`)`;
        $itn = itn + 1$;

TABLE 3.2: Scalar parameter overview. The 'Value' column indicates which value is used in the implementation. *indicates parameters added in this implementation.

Description	Parameter	Value
target coverage rate	r	$0.2 - 0.5$
Initial scale	f_{init}	0.1
scaling step size	b_i	0.1
alpha	α	0.05
beta	β	0.1
Max rotation angle	$\theta_{\max}$	$\frac{\pi}{12}$
max translation*	$t_{\max}$	${V_{mesh}}^{1/3}$
padding*	p	$0 \leq p < 1 \times {A_{avg,mesh}}^{1/2}$

Chapter 4

Results

In this chapter we will discuss the results of the implementation of the algorithms described in Figure 3.2. We will discuss the results of the implementation of the CAT cells, the trivial packing cases, the complexity of the algorithms and the results of the packing of red blood cells into a sphere.

4.1 Verification

4.1.1 CAT cells

The computation of the CAT cells involves filtering the cells of tetrahedral mesh, computing the individual CAT faces and administrating the faces with the right object. The result is a manifold cell that doesn't overlap any other CAT cell. Figure 4.1 shows the resulting CAT cell for a relatively simple configuration of 4 cubes inside a flat cuboid. Figure 4.2 shows the result of the computation of the CAT cells for more complex configuration of 4 spheres and 2 cubes inside a spherical container. Both figures show that the CAT cells are computed correctly. The cells do not overlap each other.

4.1.2 Trivial Packing Cases

Before packing multiple objects in one container, we should be able to pack a single object into a container. In Figure 4.3 the results of packing a single object into a container are displayed. The figure shows the result of the packing of a cube, sphere, tetrahedron and cone into a thin version of themselves. The result shows that the packing is computed correctly, as the objects are packed into the container without any overlap. In these results, the coverage rate is set to 0.8 because the object is packed into a shape equal to

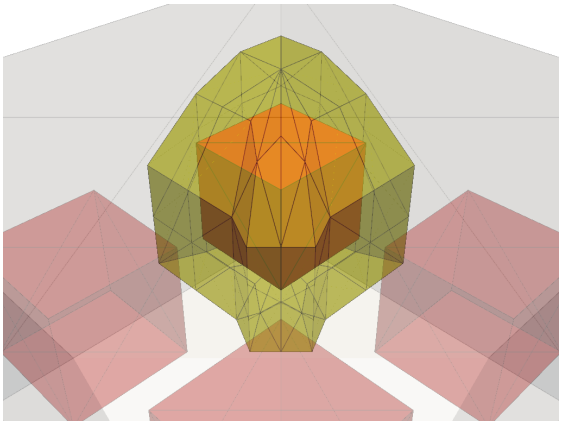


FIGURE 4.1: The CAT faces of one of four objects when placed inside a flat cuboid. The yellow faces are the CAT faces and the nearly transparent grey shape is a cut of of the cuboid container.

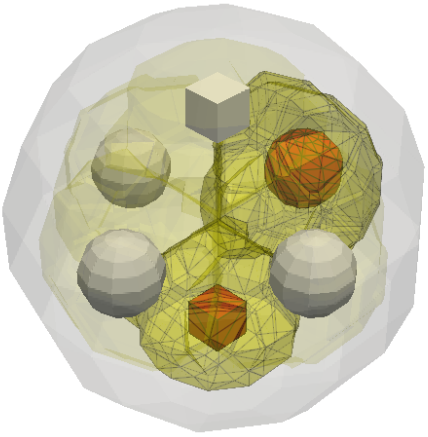
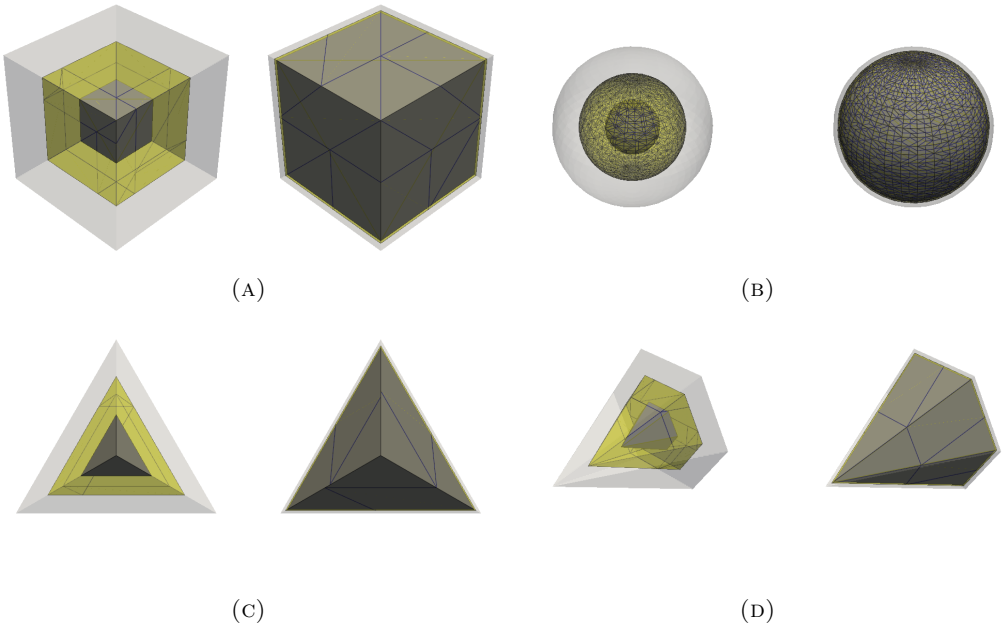


FIGURE 4.2: The CAT cells of cubes and spheres that are places inside the same container sphere. The CAT cell of the red cube and red sphere are highlighted and have their edges drawn. The CAT cells of the white spheres and cube are faded to show the contact points.

itself, therefore the object could entirely cover the space, unlike when packing multiple objects.

4.1.3 Blood cell Packing

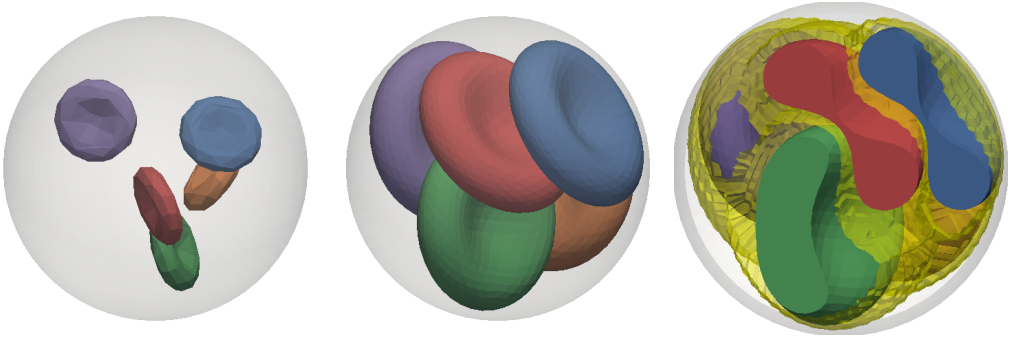
In Figure 4.4 the result of packing 5 red blood cells is shown, which had a run time of over 3 minutes. This performance is not competitive enough to be able to make a reasonable comparison with packCells for larger scale packing and the instability of the implementation doesn't allow for a larger number of objects to be packed. The maximum that has been reached within a few hours of computation is 30 red blood cells. Due to the phenomenon described in Section 3.3, the intermediate results of the computations



(A) (B)

(C) (D)

FIGURE 4.3: Single object packing of trivial shapes. This figure shows the result of the packing implementation



(A) (B) (C)

FIGURE 4.4: A demonstration of packing 5 coloured blood cells inside a sphere with a coverage rate of 30%. Figure (a) shows the initial configuration, and Figure (b) shows the final configuration and Figure (c) shows a slice through the xz-plane of the final configuration with CAT cells. This packing took 3 minutes and 25 seconds.

have to be corrected almost every iteration, which in the case larger number of red blood cells means that the solution will never reach full scale.

4.2 Performance

4.2.1 Profiling

The performance of the functional components in the implementation is measured by running an initial configuration with different stages of the implementation. The time

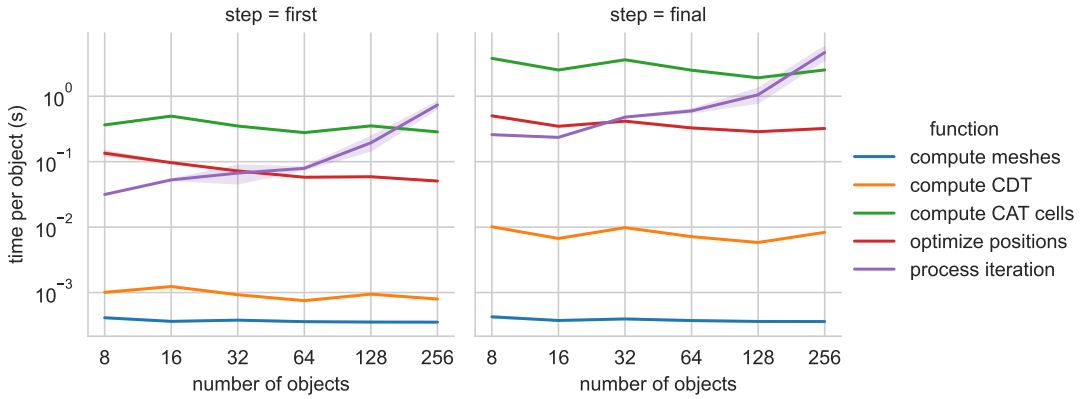


FIGURE 4.5: Performance of the functional components of the Irop Implementation for the first scaling step (left) and the last scaling step (right).

per object of each functional component are shown for the first scaling step and the last scaling step in Figure 4.5. Note that, as described in 3.1.2.3, the first scaling step uses meshes that are simplified to contain only 10% of the original number of vertices, opposed to 100% in the final step. The results show that the performance of the implementation is dominated by the computation of the CAT cells and that the cost of computing the meshes from the transform array described in Section 3.2 is neglect-able. Furthermore, the CDT algorithm costs orders of magnitudes less time than the other functions. A clear bottleneck to the performance for larger numbers of objects is caused by the post-processing described in Section 3.3.3. Apart from the high costs, the figure shows that the performance of the implementation scales well with regards to the number of objects, as we see a constant time per object for the different stages of the implementation.

The differences between the CDT and the other functions are summarised in Table 4.1. The table shows the computation of a single threaded execution for performing the operations on 128 objects.

Function	**Time (s)**	**CDT/Time**
Compute CDT	0.745152	-
Optimise Positions	4.688833	6x
Process Iteration	134.667990	181x
Compute CAT Cells	245.137027	329x

TABLE 4.1: Performance of the functional components for single core execution at the final scaling step of packing 128 red blood cells into a sphere.

The current implementation is limited to packing up to 30 cells within a few hours, while packCells is able to pack 128 cells in 0.5 seconds. The performance of packCells for up to 128 red blood cells with a single threaded execution is shown as a reference in Figure 4.6. As mentioned before, statistical comparison between the two methods does not provide further insights, because the performances are too different.

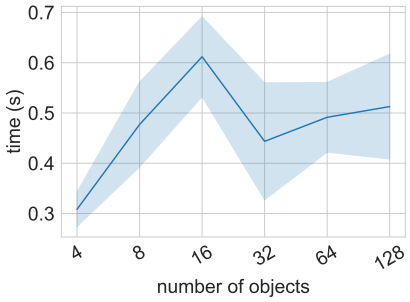


FIGURE 4.6: Performance of packCells for packing of up to 128 red blood cells on a single threaded application.

4.2.2 CAT face computation

In Figure 4.7 The relation between the number of input vertices and the number of relevant tetrahedra is shown. For CDT algorithms, the exact number of resulting tetrahedra cannot be computed from the number of vertices alone. In the use case of this thesis, only a limited set of features of the CDT is used, which allows us to provide a more accurate prediction of how many tetrahedra one will get. The figure shows that the number of relevant tetrahedra in the case of packing red blood cells into a sphere scales linearly with the number of input vertices with a factor of 2.9.

In Figure 4.8 we show the number of CAT faces per tetrahedron after CDT computation. The function used for the fit is not backed by any other fact than that it fits well and it is merely there to suggest an upperbound to the number of cat faces. It indicates that the number of CAT faces per tetrahedron is relatively low. This indicates that the the dominant split cases are the $(3, 1)$ and $(2, 2)$ split cases, that are described in Section 3.1.2.2. This is expected, as most of the tetrahedra will be in contact with only one other object relative to those that form an intersection between more than two objects.

Combining this with the number of relevant tetrahedra per vertex, we can estimate the number of CAT faces per vertex. In the case of the red blood cells, we get 2.9 relevant tetrahedra per vertex. The number of CAT faces per tetrahedron is roughly 1.2. This means that we get 3.5 unique CAT faces per vertex, which in case of a full scale red blood cell is $642 * 3.5 = 2247$ CAT faces per blood cell.

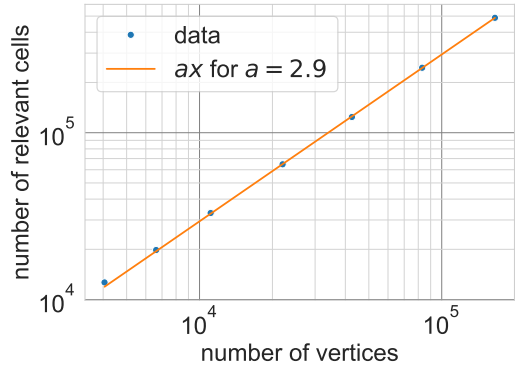


FIGURE 4.7: The number of resulting cells from the CDT that have connections to more than one object in the case of red blood cells in a sphere. For these tetrahedra, CAT faces will be computed. The line shows a linear relationship between the number of input vertices and the number of resulting relevant tetrahedra.

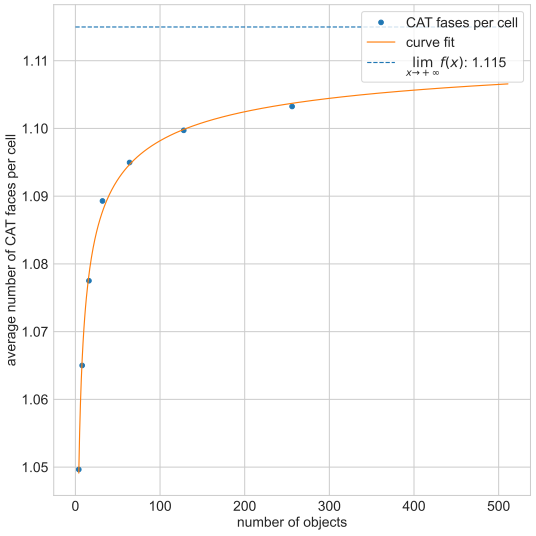


FIGURE 4.8: The average number of cat faces generated per tetrahedral cell. The data is fitted to a curve to show a potential upper bound.

Chapter 5

Discussion

From the profiling results and the performance analysis, it is clear that the current implementation of the algorithm is not competitive with packCellsẆhere packCells is capable of packing 128 objects within half a second, this implementation needs more than 3 minutes for 5 red blood cells.The complexity added by the mesh-based approach is high in terms of implementation and computational overhead. The ellipsoid approach of packCells benefits from its relative simplicity. Moreover, given that packCells is a highly optimised c++ library, whereas the implementation is merely a slightly tweaked python code, makes this basic comparison somewhat superficial.

The algorithm analysed in this thesis shows an interesting multi-phase approach to the packing problem. The paper by Ma et al. [13] does not describe the main algorithm in detail, but rather describes the algorithm in a high level. During this thesis, we have been able to fill most of the essential gaps and details necessary for the implementation. Starting with the description of the approach to the CDT. Here, the paper explicitly states that "We first uniformly sample points on the surfaces of the container and objects, [...], we use these sampling points as the input to the CDT". Now it has become clear that precisely following this description leads to incoherent and unusable results. The generated volumetric mesh will exist of tetrahedra that are crossing from the outside of one object trough the surface of the other. As these tetrahedra will be classified as multi-object tetrahedra, the resulting CAT cell will be far from correct. The paper does not describe how to avoid this problem, but it is likely that the authors used a different approach, where the input of the CDT exists not only of the vertices, but also edges and faces. However, there is no way to verify whether this is the case, as the code and implementation of the paper is not available publicly.

An other interesting parameter choice made in the algorithm description is the rotational bound of the growth-based optimisation phase. As mentioned in Section 3.1.2.3, the

constraints are computed for only a subset of the faces of a CAT cell for each vertex of an object. According to the paper, The rotational bound ensures that a vertex will not be placed outside the CAT cell while the constraints are still met. However, the paper does not provide a mathematical motivation for the rotational bound, nor does it provide an explicit reasoning for this assumption. This makes it extremely hard to determine whether unintended behaviour in the implementation is caused by an error in the code or because of the rotational bound. This proof has yet to be delivered.

One part that we have been able to confirm is the isolation of objects via CAT cells. The method for creating CAT cells in the proposed algorithm is innovative and has shown to be effective. The algorithm is able to create a mesh that isolates objects from each other, which is a valuable property with regards to parallelisability. The results in Figure 4.1 and 4.2 show a simple and more complex case of this isolation. Both cases include the complete set of tetrahedron types, discussed in Section 3.1.2.2. The later case also shows that the combination of objects with different shapes and detail are possible, as the cubes only have 8 vertices, whereas the spheres have 256.

When it comes to packing, the results displayed in Figure 4.3 show that the basic functionalities are in place. Furthermore, the results in Figure 4.4 show that the implementation is successful for 5 objects and reaching a coverage rate of 30% in 200 seconds. These results are not as promising as the coverage rates reported in the paper by Ma et al. or the coverage rates currently reachable using packCells. This is expected, as this implementation only covers one of the methods suggested by Ma et al.. However we cannot rule out if the proposed algorithm cannot reach higher coverage rates than shown here. From the result shown in Figure 4.4c, we can clearly see that the spaces that are ignored by packCells are filled. The concave notches of the red blood cells are filled by the tips of other cells. For a larger scale simulation this could lead to a more optimal configuration and a therefore a higher coverage rate.

The CAT analysis shown in Figure 4.7 and 4.8 provide further insight in how this method will scale when optimised. The number of CAT faces per cell of the generated tetrahedral mesh is constant and the number of tetrahedra that are required to compute the CAT cells scales linearly with the number of vertices. This intuitive result is nonetheless important, especially considering the fact that the number of tetrahedral cells as output of CDT is usually unknown. Based on these results, we can conclude that the number of operations performed per cell has a constant relation to the the underlying CDT algorithm. The results are based on a tetrahedral mesh with a fixed number of vertices per object and inside a spherical container. The results are therefore not representative for the general case, but they do provide a good indication of the complexity of the algorithm.

Also, the profiling results displayed in Figure 4.5 show a decreasing slope with respect to the time performance. We can conclude the individual methods scale well with the number of objects in both the first stage as the last stage, but we cannot say the same for the algorithm as a whole. These results originate from the isolated functional components and so they do not take into account situations where the optimisation algorithm takes longer to converge. In addition, Table 4.1 shows that compared to computing the CDT, the other methods are significantly slower. This is not surprising, as the CDT is a highly optimised algorithm, whereas the other methods are not. The computation of the CAT faces shows to be the most costly functional component of the implementation with a 329x time consumption relative to the CDT . This is explained by the fact that this method is fully implemented in python without external libraries outside of numpy and with no accelerations in place. Then, the post-processing shows to be 181x more costly than CDT is more than half of the CAT computation. In case of a stable implementation, this method should become redundant. Moreover, optimising positions, which, in contrast to the other methods, has been accelerated by improving the data structure and using JIT compilation, shows to be only 6x as costly as CDT . The profiling results of one of the first (semi) working implementation showed that this function was responsible for over 94% of the total run time. This probably says more about the quality of that early implementation, but its still a satisfying improvement. Overall, The profiling results show that there is a lot more room for improvement, before the CDT becomes the performance bottleneck.

With regards to implementation, this thesis project has successfully implemented an open-source package that is designed to be portable and easily extendable to a more optimised implementation. Parts of the code already have some optimisations in place and parts are yet to be optimised. The code base uses a static code checker, has ci/cd pipelines to ensure code quality and has a high test coverage. The as previously mentioned, the code is not stable, however the fundamentals for a mature code base are in place.

As we look to the future of this project, our immediate focus lies in fine-tuning the algorithm to achieve a more dependable, consistent performance. Despite many hours already dedicated to its development,the insights and understanding we've gleaned so far place us in an advantageous position to attain this goal.

Once we've achieved stability, our attention will then turn to optimising processes, in particular the calculation of CAT faces. By implementing similar improvements as seen in growth-based optimisation methods, we can more accurately compare our approach to packCells and truly showcase the potential of our mesh-based methodology. It's also

at this stage that we should evaluate the feasibility of distributed computing for our project.

Additionally, we're intrigued by the potential of broader-use optimisation libraries, such as Taichi, and Python super sets like Mojo. These tools could help us optimise our code for GPU, all while keeping the code readable and relatively simple.

Lastly, to elevate our performance even further, we're considering the prospect of performing CDT computations using GPUs. This area has already seen promising work done and could become a fruitful direction for our project's improvement.

In the broader context of packing, Another exciting avenue that could be explored is the voxel-based approach[12], a method that segments space into a grid of volumetric pixels or 'voxels'. The significant memory requirement to store voxels is the key hurdle here; however, we believe the use of specialised compilers for sparse data structures, like Taichi[28], could potentially overcome this obstacle.

Another interesting method to model 3D shapes other than mesh or voxelisation, is by the using a spherical-harmonic approach [29]. Then, spherical-harmonic functions can be used to efficiently compute statistics like collisions and overlap. There has been no developments on this method for packing, but it could be an interesting approach for future research.

Chapter 6

Conclusion

The work undertaken in this study aimed to introduce, implement, and analyse an irregular object packing method based on the mesh-based approach proposed by Ma et al. for HemoCell. This was seen as a potential solution to the challenges faced by packCells, including the approximation of red blood cells as ellipsoids, the attainment of maximal packing density, and computational wastage due to unaligned container definition. The findings of this study have highlighted some key aspects of the mesh-based algorithm, its implementation, and its potential for future improvement.

Our analysis demonstrated that the mesh-based approach, as it stands, is currently less competitive in terms of computational time compared to packCells. However, this is a rather superficial comparison, given that packCells is a highly optimised C++ library, and the mesh-based implementation examined in this study was based on minimally altered Python code. The complexity added by the mesh-based approach is high and imposes considerable computational overhead.

The study also brought forth the realisation that certain aspects of the algorithm were not well explained in the original paper, leading to implementation challenges. Issues arose with the Constrained Delaunay Triangulation and the rotational bound of the growth-based optimisation phase. This underscores the necessity for further investigation and understanding of the algorithm's internals.

Nonetheless, the algorithm's ability to isolate objects via Chordal Axis Transform (CAT) cells was confirmed and displayed innovative potential. We were able to create a mesh that isolated objects from each other, which makes it well suited for parallel computing. Additionally, preliminary packing results, while not on par with those achieved by packCells, indicated that basic functionalities are in place, and the method could potentially

lead to a more optimal configuration and higher coverage rate when employed on a larger scale.

From a scalability perspective, the complexity of the algorithm appears to be manageable. The results indicated that the number of operations performed per cell has a constant relation to the underlying CDT algorithm. However, the algorithm's current implementation has not yet shown effective scalability in practise. Profiling results highlighted that the computation of CAT faces and post-processing are significantly slower than computing the CDT, suggesting areas for further optimisation.

Overall, this research represents a first step towards the implementation and analysis of a mesh-based method for irregular object packing in HemoCell. The findings highlight the complexities and challenges of the approach, but also the potential that such an approach could hold if effectively optimised and understood. Further research and improvements are necessary to fully realise the benefits of the mesh-based packing approach.

As we move forward with this project, our primary focus will be on refining the algorithm for a consistent performance, and subsequently optimising the calculation of CAT faces. We are keen to leverage the advantages of distributed computing, general-purpose optimisation libraries such as Taichi, and Python supersets like Mojo to further our work. We are also considering implementations of Constrained Delaunay Triangulation (CDT) computations using GPUs and potential explorations of alternative approaches include voxel-based packing and spherical-harmonic methods.

With the learnings from this thesis, we are confident that the exploration and development of better packing approaches in terms of space efficiency and computational waste reduction is a fruitful one.

Bibliography

[1] A L N Júnior. *The Two-Dimensional Rectangular Strip Packing Problem.* PhD thesis, 2017.

[2] Gokula Vijaykumar Annamalai Vasantha, Ananda Prasanna Jagadeesan, Jonathan Roy Corney, Andrew Lynn, and Anupam Agrawal. Crowdsourcing solutions to 2D irregular strip packing problems from internet workers. *Int. J. Prod. Res.*, 54(14):1–22, November 2015.

[3] Yuanfeng Zhou, Feng Sun, Wenping Wang, Jiaye Wang, and Caiming Zhang. Fast updating of delaunay triangulation of moving points by bi-cell filtering. *Comput. Graph. Forum*, 29(7):2233–2242, September 2010.

[4] Arta Dilo, Pieter Bos, Pawalai Kraipeerapun, and Rolf de By. Storage and manipulation of vague spatial objects using existing GIS functionality. In *Flexible Databases Supporting Imprecision and Uncertainty*, volume 203, pages 293–321. unknown, June 2007.

[5] Gábor Zavodszky, Britt Van Rooij, Victor Azizi, Saad Alowayyed, and Alfons Hoekstra. Hemocell: A high-performance microscopic cellular library. *Procedia Comput. Sci.*, 108:159–165, 2017.

[6] Victor Azizi Tarksalooyeh, Gábor Závodszky, and Alfons G Hoekstra. Optimizing parallel performance of the cell based blood flow simulation software HemoCell. *Lect. Notes Comput. Sci.*, 11538 LNCS:537–547, 2019.

[7] Shiyi Chen and Gary D Doolen. LATTICE BOLTZMANN METHOD FOR FLUID FLOWS. *Annu. Rev. Fluid Mech.*, 30(1):329–364, January 1998.

[8] Charles S Peskin. The immersed boundary method. *Acta Numer.*, 11:479–517, January 2002.

[9] J Mościński, M Bargieł, Z A Rycerz, and P W M Jacobs. The Force-Biased algorithm for the irregular close packing of equal hard spheres. *Mol. Simul.*, 3(4): 201–212, May 1989.

[10] Henny H Billett. *Hemoglobin and Hematocrit*. Butterworths, 1990.

[11] C L Conley, R P Russell, C B Thomas, and P A Tumulty. HEMATOCRIT VALUES IN CORONARY ARTERY DISEASE. *Arch. Intern. Med.*, 113:170–176, February 1964.

[12] Thomas Byholm, Martti Toivakka, and Jan Westerholm. Effective packing of 3-dimensional voxel-based arbitrarily shaped particles. *Powder Technol.*, 196(2):139–146, December 2009.

[13] Y Ma, Z Chen, W Hu, and W Wang. Packing irregular objects in 3D space via hybrid optimization. *Comput. Graph. Forum*, 37(5):49–59, August 2018.

[14] Harald Dyckhoff. A typology of cutting and packing problems. *Eur. J. Oper. Res.*, 44(2):145–159, January 1990.

[15] Ethan L Schreiber and Richard E Korf. Improved bin completion for optimal bin packing and number partitioning. In *Twenty-Third International Joint Conference on Artificial Intelligence*. ijcai.org, 2013.

[16] Sándor P Fekete, Jörg Schepers, and Jan C van der Veen. An exact algorithm for Higher-Dimensional orthogonal packing. *Oper. Res.*, 55(3):569–587, June 2007.

[17] Jens Egeblad and David Pisinger. Heuristic approaches for the two- and three-dimensional knapsack packing problem. *Comput. Oper. Res.*, 36(4):1026–1049, April 2009.

[18] David S Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

[19] Nikhil Bansal, Xin Han, Kazuo Iwama, Maxim Sviridenko, and Guochuan Zhang. A harmonic algorithm for the 3D strip packing problem. *SIAM J. Comput.*, 42(2): 579–592, January 2013.

[20] Y G Stoyan, N I Gil, G Scheithauer, A Pankratov, and I Magdalina. Packing of convex polytopes into a parallelepiped. *Optimization*, 54(2):215–235, April 2005.

[21] Wenchao Hu, Zhonggui Chen, Hao Pan, Yizhou Yu, Eitan Grinspun, and Wenping Wang. Surface mosaic synthesis with irregular tiles. *IEEE Trans. Vis. Comput. Graph.*, 22(3):1302–1313, March 2016.

[22] Lakshman Prasad. Morphological analysis of shapes. *CNLS newsletter*, 139(1): 1997–1907, 1997.

[23] Richard Southern, Edwin Blake, and Patrick Marais. GeMS: Generic memoryless polygonal simplification. August 2000.

[24] Célestin Marot and Jean-François Remacle. Quality tetrahedral mesh generation with HXT. August 2020.

[25] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: a LLVM-based python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, number Article 7 in LLVM '15, pages 1–6, New York, NY, USA, November 2015. Association for Computing Machinery.

[26] Hang Si. TetGen, a Delaunay-Based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, 41(2), February 2015.

[27] Andrew Badr. TimeComplexity - python wiki. https://wiki.python.org/moin/TimeComplexity, 2023. Accessed: 2023-6-17.

[28] Yuanming Hu, Mit Csail, Tzu-Mao Li, U C Berkeley, Luke Anderson, Jonathan Ragan-Kelley, Berkeley Frédo Durand, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *dl.acm.org*, 38 (6):16, 2019.

[29] Xiang Wang, Zhen-yu Yin, Hao Xiong, Su Dong, and Yun-tian Feng. A spherical-harmonic-based approach to discrete element modeling of 3D irregular particles. *Int. J. Numer. Methods Eng.*, 122(5), June 2021.