CONTACT-DETECTION IN THE COMPUTER SIMULATION OF BIOPOLYMER NETWORKS

TERM PAPER

Author: Christian Roth
Technische Universität München

Supervision: Dipl.-Ing. Kei W. Müller
Institute for Computational Mechanics – Technische Universität München

Dipl.-Ing. Christian J. Cyron
Institute for Computational Mechanics – Technische Universität München

Prof. Dr.-Ing. W. A. Wall
Institute for Computational Mechanics – Technische Universität München
CONTACT-DETECTION IN THE COMPUTER SIMULATION OF BIOPOLYMER NETWORKS

AUTHOR: CHRISTIAN ROTH,
SUPERVISION: KEI MÜLLER¹ AND CHRISTIAN CYRON¹

ABSTRACT. In this term paper, we propose efficient parallel algorithms for contact detection in finite-element simulations of biopolymer networks using a linear octree. In this algorithm a cuboidal bounding volume is constructed for each polymer element upon which spatial decomposition is performed by the octree. Finally, contacting elements are detected running a bounding volume intersection test within the leaf octants. Present MPI-based implementation demonstrates low constant scalability over a wide actin filament concentration range between 0.1 and 8μM, perfectly interacting with our in-house finite element Code BACI. The results are consistent for different polymer phases such as bundles, layers, and clusters. Octrees arranging more than 16384 elements were constructed and intersection was executed on 4 processors in a time period two orders of magnitude below the contact algorithm itself. Like other existing algorithms for constructing octrees, our algorithms have O(n log n) work complexity resulting in low wall times for multi- and self-contact detection even in large biopolymer networks.

¹ Institute for Computational Mechanics
Technische Universität München
Boltzmannstr. 15
D-85748 Garching
## CONTENTS

1. Introduction ................................................................. 1

2. Octree Theory
   2.1 Index Encoding ........................................................................ 4
   2.2 Bounding Volume Employment.................................................... 4

3. Main Part – Algorithms
   3.1 Bounding Box Generation.......................................................... 6
   3.2 Octree Construction................................................................. 10
   3.3 Bounding Box Intersection....................................................... 12

4. Performance Evaluation
   4.1 Test Data....................................................................................... 14
   4.2 Comparison with the Existing Brute Force Algorithm.................. 14
   4.3 Behaviour for Different Polymer Phases................................. 16
   4.4 Governing Parameters of the Octree........................................... 18
   4.5 Complete Contact Simulation................................................... 20

5. Conclusions and Further Work ............................................. 23

6. Acknowledgements .................................................................. 23

7. References ............................................................................. 23
1. INTRODUCTION

The growing interest in a deeper understanding of microbiological processes in the human body such as cell division, cell motility and mechanotransduction has lead to various simulation methods for biopolymer networks, which crucially determine the mechanics of biological cells. While the classical bead-spring or bead-rod models are still widely used in this field, they have well known limitations in their representation of mechanical phenomena occurring in biopolymer networks such as the description of torsion. In this respect, [1] proposes a new approach by applying the finite-element method to polymer dynamics. Within this mathematically sounded framework various phenomena observed in polymer networks can be simulated including Brownian dynamics and statistical mechanics. However, present finite-element simulations allow interpenetration of filaments. Thus, they cannot capture physical phenomena like the diameter of polymer bundles or so-called trapped entanglements in a realistic manner. The understanding of such effects, which are assumed to be responsible for certain effects in human cells, may even be important for the understanding the origin of cancer and require an adequate contact simulation [3, 4].

In this term paper efficient parallel algorithms for contact-detection in polymer networks are presented. The developed methods are based on octrees, hierarchical tree-like structures widely used for spatial decomposition and searching tasks in 3-dimensional domains in many fields of research. Octrees use axis-aligned lines and planes to sort the input objects on the right places within the boundaries. All subsequent finite-element contact computation routines only consider objects which are near to each other, e.g. which are sorted to proximate positions within the boundaries. Before, the octree-implementation contact detection was realized by a brute-force algorithm checking all element pairs for intersection, resulting in \(O(n^2)\) complexity and limiting the whole contact computation process in terms of system size. These high costs for partitioning and searching in polymer networks have expedited the development of an octree-based contact detection method.

The main contribution of this work lies in the application of the octree data structure to simulations of polymer networks in order to achieve an efficient parallel contact detection algorithm. In the present work this is done by (1) setting up bounding volume geometry for each polymer element; (2) constructing a top down octree upon these bounding volumes; (3) performing an intersection test between neighbouring bounding volumes identifying contact pairs which are handed over to the contact routine. Stages one and three are parallel. Like many other existing algorithms for constructing octrees, our algorithms have \(O(n \log n)\) work complexity resulting in low wall times for multi- and self-contact detection.

This paper is organized as follows. In section 2 consistent terminology on octrees is introduced, which will be used in the rest of this work. In section 3 we describe the octree construction process. Section 4 contains numerical experiments including comparisons with the previous brute-force implementation and performance tests that demonstrate the efficiency of the octree approach. Finally, section 5 gives a brief outline of the developed algorithm and offers a few suggestions for future work. Table 1 summarizes the notation that is used in the following sections of this work. The standard physical parameters for a homogenous isotropic human cytoskeleton are presented in Table 2.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbf{m}_1, \mathbf{m}_2$</td>
<td>Position vectors to the centre of sphere 1, 2</td>
</tr>
<tr>
<td>$r_1, r_2$</td>
<td>Radii of sphere 1, 2</td>
</tr>
<tr>
<td>$V_{FE}$</td>
<td>Volume of a finite element</td>
</tr>
<tr>
<td>$R_{\text{filament}}$</td>
<td>Radius of a filament</td>
</tr>
<tr>
<td>$L_{FE}$</td>
<td>Length of a finite element</td>
</tr>
<tr>
<td>$V_{\text{SPB}}$</td>
<td>Volume of a bounding sphere</td>
</tr>
<tr>
<td>$V_{\text{AABB}}$</td>
<td>Volume of an axis-aligned bounding box</td>
</tr>
<tr>
<td>$\bar{V}_{\text{AABB}}$</td>
<td>Arithmetic average of an axis-aligned bounding box for elements with an equal angular orientation</td>
</tr>
<tr>
<td>$V_{\text{OBB}}$</td>
<td>Volume of an oriented bounding box</td>
</tr>
<tr>
<td>$D_{\text{max}}$</td>
<td>Maximum permissible treedepth</td>
</tr>
<tr>
<td>$N_{\text{max}}^p$</td>
<td>Maximum number of bounding boxes per leaf octant</td>
</tr>
<tr>
<td>$L_{\text{vol}}$</td>
<td>Length of control volume</td>
</tr>
<tr>
<td>$N$</td>
<td>Number of filaments in control volume</td>
</tr>
<tr>
<td>$l$</td>
<td>Length of an octant</td>
</tr>
<tr>
<td>$N^p$</td>
<td>Number of bounding boxes in octant</td>
</tr>
<tr>
<td>$L(n)$</td>
<td>Level of octant $n$; length of current treedepth</td>
</tr>
<tr>
<td>$n_p$</td>
<td>Number of processors</td>
</tr>
<tr>
<td>$(i, j)$</td>
<td>Pair of elements with IDs $i$ and $j$</td>
</tr>
</tbody>
</table>

**TABLE 1: Symbols and Notation**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{\text{filament}}$</td>
<td>5nm</td>
</tr>
<tr>
<td>$L_{\text{filament}}$</td>
<td>4μm</td>
</tr>
<tr>
<td>$N_{\text{discret}}$</td>
<td>Number of elements per filament</td>
</tr>
<tr>
<td>$L_{FE} = \frac{L_{\text{filament}}}{N_{\text{discret}}}$</td>
<td>0.125μm</td>
</tr>
<tr>
<td>$L_{\text{vol}}$</td>
<td>5μm</td>
</tr>
</tbody>
</table>

**TABLE 2: Standard physical parameters for the human cytoskeleton**
2. OCTREE THEORY

At the beginning of this section a brief definition of an octree is given. An octree is a tree-like data structure in which every node possesses eight children. A node of an octree stands for the cubic cell of the domain it represents. Octrees are similar to binary trees (two children per node) in one dimension and quadtrees (four children per node) in two dimensions. A node with no child is called a leaf and a node with one or more children is called an interior node. The only node with no parent is the root and all other nodes have exactly one parent. Nodes are called siblings, if they share the same parent. A node’s children, grandchildren, and so on are collectively referred to as the node’s descendants, and this node will be an ancestor of its descendants. A node along with all its descendants can be considered as a separate tree in itself with this node as its root. Hence, this set is also referred to as a subtree of the original tree. The depth of a node from the root is referred to as its level. As shown in Fig. 1, the root of the tree is at level 0, and every interior node is one level lower than its children [5].

Octrees and quadtrees can be used to partition cuboidal and rectangular regions (Fig. 1 (b)). These regions are referred to as the domain of the tree. A set of octants is said to be complete if the union of the regions spanned by them covers the entire domain. Alternatively, one can also define complete octrees as octrees in which every interior node has exactly eight child nodes, which together represent the same space as their parent. We will frequently use the equivalence of these two definitions.

---

**FIG. 1.** (a) Octree tree-like data structure; (b) Corresponding 3-dimensional cubic domain; and (c) Visualization as 2-dimensional quadtree, this is also used later for visual clarity
There are many different ways to define trees represented in [5]. In present work, we will use a linearized representation of octrees known as linear octrees. For this representation, we discard the interior nodes and only store the complete list of leaves. This is possible as our octree is complete, i.e. every interior node has exactly eight children and as the maximum treedepth $D_{\text{max}}$ is defined a priori (see section 3). The linearized representation is advantageous over other representations due to its lower storage costs. To use a linear representation, a locational code is needed to identify the octants [5]. This code contains information about the position and level of the octant in the tree. The following section describes the encoding method used in this work via a unique index-chain.

2.1 INDEX ENCODING

A lot of encoding schemes for linear octrees exist in literature, primarily using the Morton encoding scheme [5]. Here the entire domain is formed by a grid with $2^{D_{\text{max}}}$ cells in each dimension, where each cell is very efficiently encoded by an integer triplet representing its $x$, $y$ and $z$-coordinate (see Fig.2). Thus a three-dimension locational problem is transformed into a one-dimensional indexing problem. When a complete traversal of all octree-leaves is required, the Morton encoding turns out to a good advantage, as the space-filling Z-order curve allocates spatially nearby octants [5]. Further work on the Morton encoding can be found in [5] and [14]. However, in the case of polymer networks this encoding has two major drawbacks. The IDs (i.e. identification numbers) are not very informative regarding the current spatial location of an octant within the domain, and thereby exacerbate control and validation. Moreover, accessing a specific cell requires going through all of its predecessors. The encoding via index-chain is more appropriate, especially in polymer networks, when certain filaments close to octant boarders or in the centre of cluster formations are of particular interest.

In order to construct an index encoding, each child-octant is identified by an integer from 0 to 7 representing its $x$, $y$, and $z$ location relative to its parent as shown in Fig. 2. Those indices for each level are pushed back, forming an index-chain with its length being the treedepth $L(N)$. So any octant in the domain can be uniquely identified by following its index-chain (Fig. 1 (c)). More interesting characteristics of the index-chain encoding will appear when the octree is used for crosslinker search or search across neighbouring octants. In the following work the terms coarser and finer are used to compare cells based on the length of their index-chains, i.e., their levels in the octree.

![Locational position of the indices for the child-octants in three spatial dimensions](image-url)
2.2 BOUNDING VOLUME EMPLOYMENT

In many contact-detection applications, the runtime is determined by the necessary intersection tests between in-plane surfaces [7]. In our case we have a similar problem caused by the complex entanglements of polymers. As a remedy, we use cheap and fast intersection tests between simple 3-D geometries which hull our real filament or the elements respectively. These simple geometries are the so-called bounding volumes and they will appear in the first and third step of our algorithm. The octree consequently will only “see” bounding volumes as input objects that have to be spatially decomposed and delivers the IDs of intersecting bounding volumes as output for the next contact computation step.

Another advantage of the bounding volume approach is the high modularity of our implementation. If the octree has to be applied to different kinds of problems, such as crosslinker-search for example, only the bounding box geometry has to be varied, whereas the tree construction part remains fairly the same. More about bounding volumes can be found in the next section.
3. MAIN PART – ALGORITHMS

In this section we describe our algorithms for contact-detection in filamentous actin networks. In order to give a brief outline of our algorithm in detail, it is depicted in the following (Fig. 3).

![Diagram](image)

**Fig. 3.** (a) Two filaments of a network actin filaments of a human cell for example; (b) Generating a simple hull geometry, i.e. the bounding box; (c) Spatial decomposition of the bounding boxes using the octree; and (d) Intersection test within one leaf octant again upon the bounding box geometry; the marked boxes are handed over to the contact computation

3.1 BOUNDING BOX GENERATION

Two desirable qualities of intersection tests are high efficiency and correctness, which means that all contacting elements are delivered, but no others. As already written before, bounding volumes are simple geometric objects created in order to speed up intersection tests and hence are applied for efficiency reasons. However, this is not straightforward, as the hulling body always creates an “empty space” which is not filled by the original filament. One goal consequently is to create tight-fitting boxes. Tight-fitting, however, means a more complex geometry, complicating intersection tests. So there is always a trade-off between tight-fitting and easy-to-intersect boxes. In the following section, three very common box types are studied for application in filamentous biopolymer networks. A homogenous isotropic actin network described by the standard physical
parameters from Table 1 is used as reference. The main criteria for evaluation of bounding volumes are:

- low complexity in box building
- cheap box intersection tests
- high percentage of filled space $\eta_{filled}$

### 3.1.1 Spherical bounding boxes (SPB)

A spherical bounding box is basically represented by a sphere around the centre of each filament, where the diameter is given by the element length $L_{FE}$. This bounding volume type is very easy to construct and intersection between two spheres denoted with indices 1 and 2 only requires one operation.

$$\|m_1 - m_2\| = \sqrt{(m_1^x - m_2^x)^2 + (m_1^y - m_2^y)^2 + (m_1^z - m_2^z)^2} \leq r_1 + r_2$$ (1)

Furthermore, the spherical bounding box has very low storage costs being defined by one midpoint and its radius. However, in case of 1-D polymer elements the empty volume is very large. This leads to a huge number of contacting bounding volumes although the contained elements do not contact. Those non-contacting pairs must in turn be sorted out in the subsequent contact computation, which diminishes efficiency and is not appropriate in our case.

The equations read:

$$V_{FE} = \pi \cdot R_{filament}^2 \cdot L_{FE}$$ (2)

$$V_{SPB} = \frac{4}{3} \cdot \pi \cdot \left(\frac{L_{FE}}{2}\right)^3$$ (3)

$$\eta_{filled} = \frac{V_{FE}}{V_{SPB}} = 6 \cdot \left(\frac{R_{filament}}{L_{FE}}\right)^2 \xrightarrow{\text{parameters from Table 2}} \eta_{filled} \approx 1\%$$ (4)

![Fig. 4. Spherical Bounding Volume](image)

### 3.1.2 Axis-aligned bounding boxes (AABB)

Another easy to construct bounding volume is the axis-aligned bounding box. It is characterized by edges and faces which are parallel to the axes of the coordinate system. An axis-aligned bounding box is defined by two extreme points with three coordinates each. For example an AABB named $A$ is defined by $a_{\text{min}} = [x_{\text{min}}, y_{\text{min}}, z_{\text{min}}]^T$ and $a_{\text{max}} = [x_{\text{max}}, y_{\text{max}}, z_{\text{max}}]^T$, where $a_{\text{min},i} \leq a_{\text{max},i}, i \in x, y, z$. $B$ is defined by $b_{\text{min}}$ and $b_{\text{max}}$ respectively. Intersection requires tests along
the three coordinate axis, the boxes are disjoint if they do not overlap in all three directions, see Algorithm 1 [8, 9]. This will be used both for intersection between bounding boxes and octants (which is in fact an axis-aligned box also) in section 3.2 and between two AABBs in section 3.3.

![Axis-aligned bounding boxes with edges parallel to the global coordinate-system](image)

**Algorithm 1. Intersection Test for Two Cuboids — IntersectionTest.**

<table>
<thead>
<tr>
<th>Input:</th>
<th>Coordinates (a_{min} ; a_{max} ; b_{min} ; b_{max})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>“Intersection” or “Disjoint”</td>
</tr>
</tbody>
</table>

1. for \(i = x, y, z\)
2. if \((a_{min,i} > b_{max,i} \; \text{or} \; b_{min,i} > a_{max,i})\)
3. return Disjoint;
4. end
5. else
6. return Intersection;

Analogous to SPB we will also compute the empty volume for this type of box. The volume of an AABB is given by

\[
V_{AABB} = x \cdot y \cdot z
\]  

with

\[
x = L_{FE} \cdot \sin \theta \cdot \cos \varphi
\]

\[
y = L_{FE} \cdot \sin \theta \cdot \sin \varphi
\]

\[
z = L_{FE} \cdot \cos \theta
\]

and \(\theta, \varphi \in [0; \pi/2]\). For a homogenous isotropic network, the arithmetic average for the volume of an AABB yields

\[
\bar{V}_{AABB} = \frac{\pi^2}{2} \int_0^{\pi/2} \int_0^{\pi} V_{AABB} \; d\theta \; d\varphi = L_{FE}^3 \int_0^{\pi/2} \int_0^{\pi} \cos \theta \cdot \sin \theta \cdot \cos \varphi \cdot \sin \varphi \; d\theta \; d\varphi
\]

\[
= L_{FE}^3 \left(\frac{\pi}{2}\right)^2 \left(-\frac{1}{6} \cdot (\sin \theta)^3 \cdot (\cos \varphi)^2\right) \left(\frac{\pi}{2}\right)^2 = 2 \cdot \frac{L_{FE}^3}{\pi^2}
\]

(9)

With the volume of an element computed in (2) we get

\[
\eta_{filled} = \frac{V_{FE \; \text{parameters from Table 2}} \; \bar{V}_{AABB}}{\bar{V}_{AABB}} \eta_{filled} \approx 10\%
\]

(11)
3.1.3 Discrete Oriented Polytope (k-DOP)
The discrete oriented polytope is the generalization of an axis-aligned bounding box. It uses $k$ planes for huling the filament, compared to the axis-aligned bounding box, which uses only the six planes aligned to the axes of the coordinate system. As a consequence the k-DOP has a higher wrapping efficiency than the above mentioned boxes but also more complicated intersection tests along $k/2$ axes. For this reason we will not handle k-DOPs here in detail, and go directly to the refined oriented bounding boxes. For further particulars on k-DOPs we refer to [8, 9].

3.1.4 Oriented bounding boxes (OBB)
If a more tight-fitting box is needed, one can also use oriented bounding boxes. They are similar to axis aligned boxes, but can be arbitrarily rotated in filament direction. The geometry is still that of a rectangular box with six pairwise parallel faces. An OBB $C^i$ can be described by the centre point of the box and three normalised positively oriented vectors $n_x, n_y, n_z$ which describe the side directions of the box [8, 9].

If we compute the percentage of filled space when rectangular OBBs with the same edge length as the filament diameter are applied we obtain for our reference case with an isotropic homogenous network

$$V_{OBB} = 4 \times R_{FE}^2 \times L_{FE}$$

Again with the volume of an element computed in (2) we get

$$\eta_{filled} = \frac{V_{FE \text{ parameters from Table 2}}}{V_{OBB}} \Rightarrow \eta_{filled} \approx 78\%$$

However intersection-tests are relatively complicated for oriented boxes and require tests along 15 axes, which already turned out to be slower then AABBs in 1D-problems [10, 11]. Moreover, the octants remain axis-aligned and thus the intersection between a box and an octant remains the same, only the box-box intersection outlined in section 3.3 would change with this more complex bounding volume geometry.
3.1.4 **Bounding Volumes applied in this work**

In fact, we will use axis aligned bounding boxes for a first robust implementation, as they are relatively tight fitting and very easy to intersect; only tests along the three coordinate-axis are required. Spheres are applied for crosslinker search in future projects, as the physical background of a crosslinker radius is reasonably modelled by a spherical bounding volume. Oriented bounding boxes are possibly realized in a later, more refined model if the need occurs.

We already mentioned at the beginning of this section that correctness is crucial in contact-detection routines, which means that all possible contact pairs must be identified. In order to ensure this for pairs which are near to each other but not in direct contact we introduce the so-called extrusion factor $f_{extrusion} \approx 1.2$ to expand the boxes in all three spatial dimensions (details see Algorithm 2). Now, tangent boxes are also recognized as contacting. In future implementations one could also consider adaptive extrusion factors dependent on the filament orientation. Note that the minimum size of bounding boxes even when they are parallel to coordinate axes is twice the filament radius.

For physical reasons, our simulation possesses periodic boundary conditions at the faces of the control volume. This means that each filament, which exceeds one boundary, is shifted towards the opposite face and continued there. With this features the principle steps for generating axis aligned bounding boxes are visualized in the following scheme.

---

**Algorithm 2. Generation of Axis-Aligned Bounding Boxes for Each Element (Parallel)**

```
Input: Finite-element discretization
Output: EpetraMultivector allAABB of format [xmin, xmax, ymin, ymax, zmin, zmax, ID]
1. for (all elements in discretization)
2. get Nodes (GID) from discretization
3. get Coordinates from Nodes
4. if (currentElement = broken)
5. shift exceeding node towards corresponding end
6. calculate midpoint of currentElement
7. calculate sidelength of currentElement in each direction
8. if sidelength < filamentdiameter
9. sidelength = filamentdiameter
10. for (direction x to z)
11. lowerLimit = midpoint - 0.5* sidelength*extrusionfactor
12. upperLimit = midpoint + 0.5* sidelength*extrusionfactor
13. end
14. store lowerLimit, upperLimit and ID in EpetraMultivector
15. end
```

3.2 **Octree Construction**

We use a recursive top down approach for the construction of the complete linear octree from the set of bounding boxes we obtain from the above section. The root here is always the control volume which has to be given. For the tree construction process we define two abort criteria. (1) An octant contains less than $N_p^{max}$ bounding boxes; (2) The maximum allowed treedepth $D_{max}$ is reached. If none of those criteria is fulfilled, tree partitioning is recursively performed and each octant is divided into eight children (Fig. 7). The algorithm for the construction of the octree based on the number of bounding boxes within the domain is outlined in Algorithm 3.
and Algorithm 4. Algorithm 4 (LocateAll) handles all governing parameters such as $N_p^{\text{max}}$, $D_{\text{max}}$, the boundaries of the octree and initiates the first subdivision.

**ALGORITHM 3. RECURSIVELY CONSTRUCTION OF THE OCTREE – LocateBox.**

**Input:** OctreeLimits, $N_p^{\text{max}}$, $D_{\text{max}}$, allAABB_vec2  
**Output:** AABBinOctants

1. calculate the centre of the parent octant $[x_{\text{center}}, y_{\text{center}}, z_{\text{center}}]$ from OctreeLimits
2. calculate the limits of all eight sub-octants; Matrix $<1,6>$ sublim (see Fig. 2)
3. for sub-octant = 0 to 7
4. for all bounding boxes in allAABB_vec2
5. if current box is contained in sub-octant
6. add current box to temporaryAABB_vec2
7. raise treedepth of bounding boxes in current octant by +1
8. end
9. end
10. if ((Number of bounding boxes in current octant) > $N_p^{\text{max}}$) && treedepth < $D_{\text{max}}$
11. call LocateBox(sublim, $N_p^{\text{max}}$, $D_{\text{max}}$, temporaryAABB_vec2)
12. else
13. store the current octant and all its contained bounding boxes in AABBinOctants
14. end

**FIG. 7.** (a) Initial octant created by LocateAll, e.g. the bounding volume; (b) octree after first subdivision into eight children, treedepth=1; (c) further recursive decomposition of this octant where the abort criterion is not fulfilled; (d) further recursive decomposition at a finer level
Algorithm 3 (LocateBox) is the recursive algorithm, which is called until one abort criterion is fulfilled. Basically this is the octree-algorithm which first calculates the limits of the eight children, then assigns the bounding boxes to these eight children and evaluates the need for a finer spatial resolution. In the end we obtain a list of all existing leaf-octants in the domain and their completely or at least partially contained bounding boxes (AABBInOctants). This data structure is the input for the bounding box intersection algorithm described in the next section.

3.3 Bounding Box Intersection

In the last step of our contact-detection algorithm we are on leaf-level of the octree. This is either reached by having less than \( N_p^{max} \) boxes per octant or by reaching the maximum treedepth \( D_{max} \) in some octant. In both cases, we have to intersect all bounding boxes which lie completely or at least partially in the current leaf-octant. This has to be done for all leaf-octants by a brute force like algorithm within each leaf octant. However due to the typically small number of boxes per leaf time for this operation is very small (see section 4.2). The intersection test itself is similar to the one described in section 3.1 as it is based on the same bounding box geometry. However, the input and output parameters are different. Moreover, boxes of neighbouring elements of one and the same filament are automatically sorted out as they are always in contact and do not give any additional information. Boxes which belong to the same filament and are not subsequent are detected though as this so-called self-contact is a common phenomenon for example in the case of the DNA double helix. The box intersection algorithm in detail is outlined in Algorithm 5 and visualized in Fig. 7.


<table>
<thead>
<tr>
<th>Input:</th>
<th>EpentaMultiVector allAABB: ([x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}, ID])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>void</td>
</tr>
<tr>
<td>1.</td>
<td>get edge-length of control volume PeriodLength from discretization</td>
</tr>
<tr>
<td>2.</td>
<td>OctreeLimits = ([0, PeriodLength, 0, PeriodLength, 0, PeriodLength])</td>
</tr>
<tr>
<td>3.</td>
<td>set ( N_p^{max} ) and ( D_{max} )</td>
</tr>
<tr>
<td>4.</td>
<td>convert allAABB (\longrightarrow) allAABB_vec2 ([ID, x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}])</td>
</tr>
<tr>
<td>5.</td>
<td>call LocateBox(OctreeLimits, ( N_p^{max} ), ( D_{max} ), allAABB_vec2)</td>
</tr>
</tbody>
</table>

Algorithm 5. Intersection Test Within One Octant (Parallel) – IntersectionAABB.

<table>
<thead>
<tr>
<th>Input:</th>
<th>2-dimensional vector containing all octants and their bounding boxes AABBinOctant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>pairs of intersecting elements in format needed for the contact computation</td>
</tr>
<tr>
<td>contactpairs</td>
<td>[problem discretization, search discretization, dofoffset, ID(i), ID(j), coordinates(i), coordinates(j)]</td>
</tr>
<tr>
<td>1. for box i (\in) ([0; size(AABBinOctant)])</td>
<td></td>
</tr>
<tr>
<td>2. for box j (\in) ([i + 1; size(AABBinOctant)])</td>
<td></td>
</tr>
<tr>
<td>3. if ID(j) &gt; ID(i)</td>
<td></td>
</tr>
<tr>
<td>4. for k = (x, y, z)</td>
<td></td>
</tr>
<tr>
<td>5. if ((a_{min,k} &gt; b_{max,k} \text{ or } b_{min,k} &gt; a_{max,k}))</td>
<td></td>
</tr>
<tr>
<td>6. return Disjoint;</td>
<td></td>
</tr>
<tr>
<td>7. else if pair ((i,j)) not already an element of contactpairs</td>
<td></td>
</tr>
<tr>
<td>8. return pair ((i,j)) to contactpairs;</td>
<td></td>
</tr>
<tr>
<td>9. end</td>
<td></td>
</tr>
<tr>
<td>10. end</td>
<td></td>
</tr>
<tr>
<td>11. end</td>
<td></td>
</tr>
</tbody>
</table>
One important point is that pairs of elements \((i,j)\), which appear double in \(contactpairs\) cannot be handled by the contact computation routine. As we can see in line 7 of Algorithm 5, it is necessary that we sort out double pairs before generating the handed over construct \(contactpairs\). In order to solve this problem we propose two methods. The first is a brute force algorithm which loops through all proceeding entries in \(contactpairs\) and compares their IDs with the current ones \((i,j)\). The current pair is only added in case it is not contained yet. This method however has \(O(n^2)\) complexity and ruins efficiency of our contact-detection algorithm (see section 4.2).

So another method based upon a map sorting is applied. Here, we propose two methods. The first is a brute force algorithm which loops through all proceeding entries in \(contactpairs\) and compares their IDs with the current ones \((i,j)\). The current pair is only added in case it is not contained yet. This method however has \(O(n^2)\) complexity and ruins efficiency of our contact-detection algorithm (see section 4.2).

**Algorithm 6. Map Storage Method for Sorting out Double Pairs — MapStorage.**

**Input:** current contacting pair \((i,j)\)  
**Output:** \(contactpairs\)  
1. \(key = i \times \) (number of elements from discretization) + \(j\)  
2. \(contactmap.insert(key, \text{vector}(i, \text{coordinates}(i), j, \text{coordinates}(j)))\)  
3. \(\text{for} \ contactmap.begin \ \text{to} \ contactmap.end\)  
4. \(contactpairs.push\_back(key, \text{vector}(i, \text{coordinates}(i), j, \text{coordinates}(j)))\)  
5. \(\text{end}\)

In this section, we have successfully handed over all contacting element pairs to the contact computation algorithm in the required data format \(contactpairs\). Of course, this process described for one leaf octant here is executed for all leaves on all processors. The octree based spatial decomposition combined with the bounding box geometry and the map storage have contributed to a very efficient algorithm, which delivers the required data structure for all subsequent contact routines.

![Intersection test within one leaf octant based on axis-aligned bounding boxes](image_url)
4. PERFORMANCE EVALUATION

In order to determine the performance of the proposed algorithms, a set of four numerical experiments was conducted including scalability analysis and parameter sensitivity studies. The algorithms were implemented in C++ using the MPI library and Trilinos packages within our in-house finite-element code BACI (Bavarian Advanced Computational Ideas). All tests were performed on a Quad-core PC at the Institute for Computational Mechanics. Figures are visualized on Matlab R2008a with the following colour scheme: filaments are bold green lines covered by lilac cuboids symbolizing the axis aligned bounding boxes. Light green lines stand for the limits of the leaf-octants and finally, intersecting bounding boxes are marked pink. In this section, results of bounding box generation, octree construction and intersection tests for a number of different polymer phases such as homogenous networks, bundle, cluster, and layer phases are presented. Analyses are completed by comparisons with the existing brute force solution.

4.1 TEST DATA

The underlying test data and input-files (except those in section 4.4) were all generated by an input file generator. Filament concentrations vary from 0.1 to 8µM and filament-diameters from 5nm to 0.3µm with a domain period length 5µm and a filament length 4µm. Each filament is equally divided into 32 elements, resulting in an element-length of 0.125µm. An example of a 4.0µM homogenous network is visualized in Fig. 9. Beam contact and statistical mechanics are applied in these examples. However, only the homogenous networks are physically correct, all other network phases may contain overlaps from the beginning and thus do not converge in their finite element simulation. They just provide a basis to test the contact detection algorithm.

![Fig. 9: Visualization of the axis aligned bounding boxes for a homogenous network with a filament concentration of 4.0µMol in a domain with period length 5.0µm and a filament length of 4.0µm, which is equally divided into 32 elements](image)

4.2 COMPARISON BETWEEN THE OCTREE APPROACH WITH AND WITHOUT MAP-METHOD AND THE EXISTING BRUTE FORCE ALGORITHM

In order to assess the advantages of using an octree approach combined with the map-method (see section 3) this performance evaluation compares both variants. First the performance of the existing brute force contact search is tested for homogenous networks with different problem sizes. It is then compared with the linear octree implementation based on axis aligned bounding
boxes (bounding boxes are constructed in every time step for each element and then spatial decomposition takes places via the octree, followed by intersection tests within one leaf). The results are presented in Table 3. Octree construction times with and without map-method and the total time for contact detection are reported proportionally to the time for the whole contacting process. Overall the code performs quite well. Both octree construction and the time for bounding box generation and intersection are two orders of magnitude below those of the brute force solution. Simply by applying bounding boxes the simulation becomes a 100 times faster and especially the employment of the map for sorting out redundant contact pairs brings a crucial advantage. Selected results are visualized in Fig. 10. Both octree approaches demonstrate good scalability over a wide range of polymer concentrations, the map-method, although being a bit more memory consuming [12], has a lower absolute runtime and thus is the method used in all future tests.

### Table 3

<table>
<thead>
<tr>
<th>Problem size (elements)</th>
<th>Filament Concentration [µMol]</th>
<th>Brute Force Algorithm [s]</th>
<th>Octree Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Vector storage [s]</td>
<td>Map storage [s]</td>
</tr>
<tr>
<td>64</td>
<td>0.05</td>
<td>0.00712204</td>
<td>0.00800182</td>
</tr>
<tr>
<td>128</td>
<td>0.10</td>
<td>8.4867</td>
<td>0.01294996</td>
</tr>
<tr>
<td>256</td>
<td>0.17</td>
<td>36.7324</td>
<td>0.05050472</td>
</tr>
<tr>
<td>512</td>
<td>0.32</td>
<td>321</td>
<td>0.2011752</td>
</tr>
<tr>
<td>1024</td>
<td>0.65</td>
<td>&gt;500</td>
<td>0.79059552</td>
</tr>
<tr>
<td>2048</td>
<td>1.23</td>
<td>&gt;500</td>
<td>3.5256896</td>
</tr>
<tr>
<td>4096</td>
<td>2.46</td>
<td>&gt;500</td>
<td>13.5516032</td>
</tr>
<tr>
<td>8192</td>
<td>4.92</td>
<td>&gt;500</td>
<td>55.9903232</td>
</tr>
<tr>
<td>16384</td>
<td>9.84</td>
<td>&gt;500</td>
<td>207.4341888</td>
</tr>
</tbody>
</table>

Time to generate axis aligned bounding boxes, to construct the octree and to perform intersection tests on four processors for increasing problem sizes. All Octrees were created using the same governing parameters $D_{\text{max}} = 5$ and $N_{\text{max}}^p = 20$. Results are presented for two storage methods of contact pairs, one using a vector of vectors, the other using a standard map. We compare our method with the existing brute force algorithm for the same filament concentration (e.g. the same number of elements) and the same underlying discretization obtained by a fixed random seed.

**Fig. 10:** Comparison of the three different approaches for contact detection in homogenous networks for filament concentrations varying over three orders of magnitude. The y-axis is scaled logarithmic and displays wall-times in seconds, whereas the x-axis enumerates the elements.
4.3 **BEHAVIOUR FOR DIFFERENT POLYMER PHASES**

Here we survey the applicability of our algorithms to several phases appearing in the simulation of polymer networks such as bundles, clusters or layers which play a crucial role in polymer physics. Therefore, the octree must have a good capability to handle all those different kinds of polymer structures in order to work efficiently over the course of the simulation. This effectiveness shall be proven here.

First, two bundle networks of different size are generated with our input generator (as already mentioned before, those structures were not created in a physically correct manner in order to save time. In fact they develop within thousands of simulated time steps. The generated distribution here is valid for testing the contact detection algorithm). Whereas the bundle denoted by “bundle small” has a concentration of 2.0µM, a filament length of 4.0µm within a 5.0µm periodic domain, the network referred to as “bundle large” differs by its concentration of 4.0µMol. This will be the same for clusters and layers herein after referred to as “small” and “large”. In the following all types of phases are checked for contacting elements, while wall-clock times are illustrated in Fig. 11 to 13. The time for constructing both bounding boxes and the octree as well as the time for performing intersection of bounding boxes are compared with homogenous networks of the same element size (see previous section) and with the time needed by the contact solver. The detected number of contacting boxes is also shown, as it clearly has an effect on the time needed for box intersection.

![Fig. 11: Behaviour of the octree contact detection in the case of bundle phases. The left group of bars indicates the small bundle network in comparison with a homogenous network exhibiting equal concentration, whereas the right group symbolizes the large bundle. The last bar in a group symbolizes the time needed by the finite element solver, which does not depend on search performance and is just for the ratio between contact detection and finite element computation. The first two bars in each group represent from bottom to top the time taken for (1) bounding box generation, (2) octree construction over the domain, (3) bounding box intersection test within the leafs. Here the first bar stands for the homogenous, the second bar for the bundle phase.](image)

Although there is a tremendous increase in the number of contacting boxes, especially in bundle networks compared to homogenous networks, wall times are just slightly increasing for bundle phases. Compared to the contact computation, the time for contact detection is still one order of magnitude lower. The enormous number of contacting elements in bundles can be explained by its specific topology. As filaments are oriented almost parallel over their whole length, each of its elements is a possible contact partner for the neighbouring filament. This phenomenon can also be observed in the case of clusters and layers, but with minor effects. In clusters and layers, contact is detected only in single elements in the centre of clusters. Bounding
FIG. 12: Behaviour of the octree contact detection in the case of cluster phases. As before the left group of bars indicates the small cluster network in comparison with a homogenous network exhibiting equal concentration, whereas the right group symbolizes the large one. The last bar in a group symbolizes the time needed by the finite element solver, which does not depend on search performance and is just for the ratio between contact detection and finite element computation. The first two bars in each group represent from bottom to top the time taken for (1) bounding box generation, (2) octree construction over the domain, (3) bounding box intersection test within the leafs. Here the first bar in the group denotes the homogenous, the second bar the cluster network.

FIG. 13: Behaviour of the octree contact detection in the case of layer phases. As before the left group of bars indicates the small layer network in comparison with a homogenous network exhibiting equal concentration, whereas the right group symbolizes the large layer. The last bar in a group symbolizes the time needed by the finite element solver, which does not depend on search performance and is just for the ratio between contact detection and finite element computation. The first two bars in each group represent from bottom to top the time taken for (1) bounding box generation, (2) octree construction over the domain, (3) bounding box intersection test within the leafs. Here the first bar in the group denotes the homogenous, the second bar the layer network.
box generation time, however, is almost the same as it is for homogenous networks, as this figure only scales with the number of elements being equal for all compared cases. Octree generation time is even slightly smaller in the case of non homogenous distributed filaments as their majority is located in a restricted volume part of the domain. So, it is not necessary to build the octree for the entire domain but only for a part (see Fig. 14), which saves time. Intersection times are increasing though, as it was already observed in bundle phases because more boxes are intersecting than in the case of homogenous distributed networks.

![Image of octree boundaries for the large layer phase.](image)

*Fig. 14: Octree boundaries for the large layer phase. Note that only a small part of the entire domain is partitioned which saves time compared to a entire subdivision that is required for homogenous networks.*

### 4.4 Governing Parameters of the Octree

In the third experimental set, the sensitivity of the octree algorithm towards its governing parameters such as maximum treedepth $D_{\text{max}}$ and number of allowed bounding boxes in a leaf octant $N_{\text{max}}^P$ is examined. For that purpose, we were running the following homogenous network file for different $D_{\text{max}}$ and $N_{\text{max}}^P$. Filament concentration here remains constant at 4µM, the domain period length is set to 5µm, containing 4µm filaments with a standard 32 element subdivision. Results of this test are presented in Fig. 15. Normally, the governing parameters are strongly coupled as a certain treedepth stands for a certain octant size, which is interrelated with the number of contained bounding boxes. So, for the first test the governing parameters are decoupled by setting $N_{\text{max}}^P$ to 1, making the maximum treedepth the only abort criteria.

The wall-time increase for $D_{\text{max}} > 4$ is conditional upon octree construction time, which increases for a higher maximum treedepth, but it is more significantly dependent on the intersection time. With octant edge length

$$l(D_{\text{max}} = 5) = \frac{5\mu m}{2^5} = 0.15\mu m$$

and an element length of

$$l_{\text{ele}} = \frac{4\mu m}{32} = 0.125\mu m$$

the octants are already so small that many bounding boxes belong to more than one octant and thus are checked for intersection with all boxes in all leaf octants they belong to.
FIG. 15: Sensitivity of the octree when the treedepth is varied and $N_{\text{max}}^P = 1$. For a small $D_{\text{max}}$, the octree building time is very low, which is reasonable as for instance $D_{\text{max}} = 2$ means there are only two subdivisions made. However, the number of bounding boxes per leaf remains high in this case resulting in a long time taken for intersection similar to the brute force case. For $D_{\text{max}} = 4$, the octree generation time slightly increases whereas the intersection time significantly decreases.

FIG. 16: Sensitivity of the octree due to changes in the governing parameters $D_{\text{max}}$ and $N_{\text{max}}^P$. On the first axis, changes in the maximum allowed treedepth are made whereas over the second axis the number of bounding boxes per leaf is varied. The wide plane shows a relative good robustness against changes in these parameters. For very high $N_{\text{max}}^P$, the algorithm tends towards brute force within the leafs but is much faster than the older brute force approach due to the application of bounding volumes. Over the maximum treedepth, there is also a minimum at $D^* = 5$ because this is exactly the point where element size is equal to the length of an octant. For every smaller octant side length the filaments are in more than one child octant.

Note, that for cluster networks (see Fig. 17), these parameters are more essential since for a very low $N_{\text{max}}^P$ the abort criterion may only be reached at a high treedepth. In the case of very small octants elements are lying in every child when further subdivided. Thus the algorithm may lead into an infinite loop. This shall be prevented by introducing a redundant check with the maximum allowed treedepth. As a result of this section, we take a $N_{\text{max}}^P$ of 20 being in the middle of the range and a $D_{\text{max}}$ of 5 as standard parameters in our simulations.
4.4 Contact Simulation

In this last experimental set, we provide evidence of the functionality of our contact detection algorithm in interaction with a finite-element simulation in BACI. The response towards Dirichlet boundary conditions and statistical mechanics is presented. Therefore, two scenarios are tested. In the first scenario, an very stiff beam with diameter of 0.3µm and 2µm length, divided by 16 elements, lowers while bending a simply-supported beam. The scenario is visualized with GMSH for several time steps. The contacting boxes found by the octree are marked for the last time step. As it can clearly be seen in Fig. 18 and Fig. 19, the octree approach based on axis-aligned bounding boxes detects contacting elements correctly. Moreover, it is even more efficient compared to brute force as it sorts out directly adjacent boxes of the same filament, but nevertheless allows self contact detection within elements of the same filament.

In a second test run, behaviour of the proposed approach in the case of statistical mechanics shall be examined. Therefore, 9 filaments (diameter 0.3µm, length 4.0µm) are arranged in a network grid consisting of 3 planes. The planes are displaced by 90° to each other in order to achieve a perpendicular orientation between the filaments of two planes. Again, one filament, the upper middle one, is considered as inextensible and moves down and right deforming all filaments in this small network. Furthermore, statistical mechanics are applied to the element nodes causing random movement of the filaments within a certain range around their original position. The
filament diameter results in cross section areas and geometrical moments of inertia being orders of magnitude higher than in real problems (0.01272 instead of 5.7E-11). Correspondingly, the statistical mechanic forces need to be raised to have an effect on those stiff filaments (kT=0.5 instead 0.00404531). All these parameters though enabling significant tests of the contact detection algorithm eventually lead to divergence after approximately 4 simulation seconds (423 time steps).

**Fig. 18:** First contact example for several time steps showing a working contact simulation with applied Dirichlet boundary conditions when contact detection is executed by the octree.

**Fig. 19:** Bounding boxes, Octree boundaries and marked intersecting boxes in the last time step of the above plotted example.
FIG. 20: Plot for several time steps of the second contact example. Statistical Mechanics are applied here, producing additional contact partners in comparison to the first test case.

FIG. 21: Bounding boxes, Octree boundaries and marked intersecting boxes in the last time step of the above plotted grid network. While bounding boxes and octree boundaries remain similar to the above example, the number of intersecting boxes increases. The upper middle filament, which is moving downwards, is contacted from the left and from the lower side thus it is nearly completely given to the contact algorithm for computation.
5. **Conclusions and Future Work**

In this work, we have presented two parallel approaches for contact detection in polymer networks using axis aligned bounding boxes and a linear octree. Both variants base on different storage methods for contact pairs, one on a vector of vectors and the other on a map. The MPI-based implementations were tested with a comprehensive set of input-files and exhibit interesting features:

- Test on four different polymer phases demonstrate that the algorithms are well equipped to handle all kinds of different filament distributions.
- Interaction both with the existing contact computation algorithm applying Dirichlet boundary conditions and statistical mechanics flow smoothly.
- Experiments on the governing parameters such as maximum treedepth and maximum number of boxes per leaf octants basically show a insensitivity of the algorithm within a wide range of parameters.
- Experiments show that the proposed map storage method has a lower wall-time than the storage in a vector of vectors and both approaches have a significantly lower running time than the existing brute force algorithm.

In the next steps, following topics have to be addressed in order to improve the performance of the octree contact detection in polymer networks. In order to achieve a better adaption of bounding volumes to the filaments the use of oriented bounding boxes should be considered (see section 3). Besides, at the moment, the octree is only capable to shift filaments exceeding the domain to the correct position in the domain according to periodic boundary conditions. But for example filaments close to but non-intersecting the right border of the domain, do not check for contact across the periodic boundaries. One solution could be to choose an appropriate extrusion factor and to cut boxes near domain boundaries similar to the process applied for cutting filaments. Finally, by implementing these changes the algorithm could even be capable to handle crosslinker search, and reduce search time for this step.

6. **Acknowledgements.**

The support for the author (C. J. R.) by Kei Müller and Christian Cyron for their great supervision in every stage of this work is gratefully acknowledged. In addition to that, the author thanks Alexander Popp (Institute for Computational Mechanics, TUM) and Stefanie Schraufstetter (Informatics V – Scientific Computing, TUM) for fruitful discussions.

7. **References**

CHRISTIAN J. ROTH: CONTACT-DETECTION IN BIOPOLYMER NETWORKS


[10] Johannes Mezger, Collision Handling in Dynamic Simulation Environments, WSI/GRIS University of Tübingen


OWN WORK DECLARATION

I confirm that this work is my own and that I have referenced all sources and auxiliary means used and acknowledged any help that I received from others as appropriate.

(place and date)   (signature)