Calculating near-densest lattice packings of non-convex objects to minimize computational box volumes in Molecular Dynamics simulations

Jur van den Berg

Supervisor: Dr. H. Bekker

RuG

Computer Science

Masters thesis

Calculating near-densest lattice packings of non-convex objects to minimize computational box volumes in Molecular Dynamics simulations

Jur van den Berg

Supervisor: Dr. H. Bekker

Rijksuniversiteit Groningen
Computer Science
Postbus 800
9700 AV Groningen

August 2002

Master's Thesis in Computing Science:

Calculating near-densest lattice packings of non-convex objects to minimize computational box volumes in Molecular Dynamics simulations

Jur van den Berg

February 2003

Abstract

In chemistry, much CPU time is spent nowadays on Molecular Dynamics simulations to gain insight in the functioning of biophysical processes. In many of these simulations, the simulated system consists of a computational box with a single macro-molecule in a solvent. Usually, one is not interested in the behaviour of the solvent, so the CPU time may be minimized by minimizing the amount of solvent. For a given molecule this may be done by constructing a computational box with minimal volume. It has been shown that the problem of finding minimal computational boxes can be reformulated as the mathematical problem of finding densest lattice packings of non-convex objects. In this paper, a method is presented to approximate such densest lattice packings. We use this method to calculate near-minimal computational boxes for a significant number of macro-molecules. These boxes prove to have typically 50% less volume than conventional boxes, and as a result the simulation time also decreases with typically 50%. For oblong molecules, this ratio can even be 80%.

Contents

Chapter 1

Introduction

In the old days of chemistry, a lot of work was being done using flasks, test-tubes, spectroscopes etc., which involved a slow and expensive process of doing research. Nowadays, much chemistry is done on computers. An important class of chemical experiments on a computer is Molecular Dynamics (M.D.) simulations. The basic concept of M.D. simulations is simple: the time development of a many atom system is evaluated by numerically integrating Newton's equations of motion. As such, M.D. has been used to study simple gases, liquids, polymers, crystals, liquid crystals, proteins, proteins in liquids, membranes, DNA-protein interactions, etc.

At many places in the world, M.D. is being used to study the behaviour of macro-molecules. In many cases, the molecules are simulated in their natural environment, mostly being water. Because simulations can not be done in an infinitely large ocean, the molecule is simulated in a computational box filled with water. The behaviour of the water is not interesting, it only serves as a natural environment for the macro-molecule. Therefore, to minimize the CPU time an M.D. simulation requires, the amount of water should be minimized. The goal of this master's thesis project is to minimize the computational box volume, without affecting the results of the simulation.

A natural way to construct a computational box is by fitting a cube as tightly as possible around the molecule to be simulated. To prevent finite system effects, the box is periodic, which means that an atom that leaves the box on one side will enter the box again on the opposite side. This is equivalent with tesselating space with identical copies of the box. Because of the box' periodicity, the box has to be space filling packable, as is a cube. In 3D, there are five convex space filling shapes, namely the triclinic box (parallelepiped), the hexagonal prism, the rhombic dodecahedron, the elongated dodecahedron and the truncated octahedron. These more complex boxes might fit tighter around a molecule than a cube, reducing the amount of water. So in contemporary Molecular Dynamics Simulation packages (such as GROMACS [15]), a complex box is used in most cases.

No matter what box shape is used, is has been shown [6] that it can always be transformed into a simple triclinic box. That is because, when space is tesselated with a box of one of the types discussed above, a lattice is defined. The principal vectors of this lattice span a triclinic box that can be used just as well as the original box for the simulation. In fact, a simulation in the triclinic box is equivalent to a simulation in the original box.

This observation opens a new view towards the construction method of the computational box. To find a box with minimal volume, it is not necessary to construct a complex box shape

tightly around a molecule, before transforming it into a triclinic box. The optimal box can be found directly by calculating the *densest lattice packing* of the molecule. The lattice vectors of this packing span the minimal volume triclinic box for the molecule.

Having reformulated the minimal volume box problem as a densest lattice packing problem, we have to look for a method that determines the densest lattice packing for a given object. For polyhedral convex objects, such a method exists [9]. However, the shape of typical macro-molecules is non-convex. For non-convex objects there exists no densest lattice packing algorithm, and in the computational geometry community, this problem is considered as hard, so it is unlikely that such an algorithm will be devised in the foreseeable future. For that reason, we devise a heuristic method that approximates the densest lattice packing of any non-convex object. We will call such an approximation a *near-densest lattice packing*.

In this paper we present an algorithm to find near-densest lattice packings of arbitrary non-convex 3D objects. We apply this algorithm to minimize computational box volumes in Molecular Dynamics simulations. This yields a significant speedup of M.D. simulations in terms of required CPU time. On average, more than 50% of the simulation time is saved when a near-minimal box is used.

This paper is organized as follows. In chapter 2 we develop a mathematical framework underpinning the method for finding a near-densest lattice packing of a general object. This method is presented in chapter 3. It assumes that we have calculated the *difference body* of the object. In chapter 4, we show how such a difference body can be constructed.

As expounded above, the presented method has a major application in the field of Molecular Dynamics simulations. This application is discussed in detail in chapter 5. In chapter 6, we integrate the results from the preceding chapters and analyze the results we achieved in M.D. simulations. In the concluding chapter 7 we summarize the principal contributions of this paper.

Chapter 2

Mathematical framework

In this chapter, we develop a theoretical framework underpinning the method we present in this paper. A number of notions we use in later chapters are defined precisely here, and we derive some key properties of these notions that are essential for our method.

2.1 Objects and Difference bodies

2.1.1 Objects

In this paper, we discuss packings of *objects*. An object can be regarded as a special instance of a subset of $\mathbb{R}^d$ (see fig. 2.1):

Definition 2.1.1 A d-dimensional *object* K is defined as a nonempty compact connected set $K \subset \mathbb{R}^d$, $d > 1$.

The *boundary* and the *interior* of an object K are denoted by ∂K and $\Im K$, respectively. We denote the object K translated over a vector a by K_a. Note that an object does not need to be convex. We define $\mathcal{K}^d$ as the set containing all d-dimensional objects.

For the precise definitions of the notions *compact*, *connected*, *boundary* and *interior*, we refer to several introductions in topology, for instance [16]. The definition of *convex* can also be found there. For this paper, however, the intuitive meaning of these notions suffices.

An important property of objects is that two identical objects have points in common, iff their boundaries also have points in common (see fig. 2.2):

Lemma 2.1.2 $K_a \cap K_b \neq \emptyset \Leftrightarrow \partial K_a \cap \partial K_b \neq \emptyset$, *for any* $a, b \in \mathbb{R}^d, K \in \mathcal{K}^d$.

It is beyond the scope of this paper to prove this lemma; the proof requires arguments from basic topology. It explicitly uses the fact that objects are connected and two- or higher dimensional.

2.1.2 Operations on objects

One of the basic operations on objects that is used repeatedly in this chapter is the *Minkowski sum*, or *sum* for short, of two sets (see fig. 2.3):

Definition 2.1.3 The *Minkowski sum* of two sets $K_1, K_2 \subset \mathbb{R}^d$, denoted by $K_1 \oplus K_2$ is defined as $K_1 \oplus K_2 = \{k_1 + k_2 \mid k_1 \in K_1, k_2 \in K_2\}$.

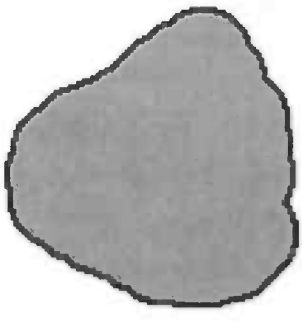

Figure 2.1: A 2D object K with boundary (dark) and interior (light).

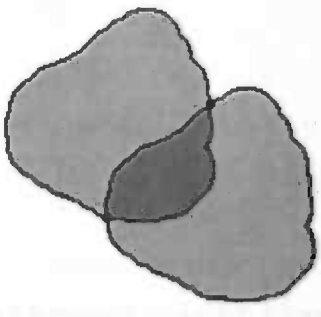

Figure 2.2: When two objects intersect, then their boundaries also intersect (dark dots).

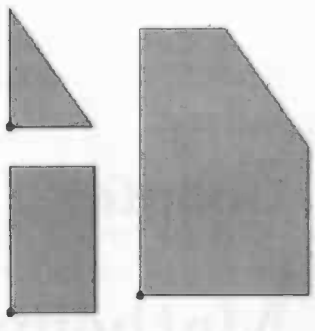

Figure 2.3: A triangle and a rectangle (left) and their Minkowski sum (right).

For example, a translation of a set K over a vector a, denoted by K_a can be written as $K \oplus \{a\}$.

The *scalar product* of is defined as follows:

Definition 2.1.4 A *scalar product* of a scalar $\alpha \in \mathbb{R}$ and a set $K \subset \mathbb{R}^d$, denoted by αK, is defined as $\alpha K = \{\alpha k \mid k \in K\}$.

Note that when K is an object, αK is also an object for any $\alpha \in \mathbb{R} \setminus \{0\}$. Also, when K_1 and K_2 are objects, then the sum of these objects form a new object. Taking these two together, we get the following lemma:

Lemma 2.1.5 $\alpha K_1 \oplus \beta K_2 \in \mathcal{K}^d$, *for any* $K_1, K_2 \in \mathcal{K}^d$, $\alpha, \beta \in \mathbb{R} \setminus \{0\}$

This lemma can be proved using basic topology arguments.

2.1.3 Difference bodies

The most important concept we use in this paper is the *difference body* $\mathcal{D}(K)$ of an object K (see fig. 2.4). Intuitively, the boundary of the difference body of K defines the region where a translated copy of K can be placed such that it touches K exactly. In this section, we define difference bodies precisely and prove some important properties of difference bodies.

Definition 2.1.6 The *difference body* $\mathcal{D}(K)$ of a set $K \subset \mathbb{R}^d$ is defined as $K \oplus -K = \{k_1 - k_2 \mid k_1, k_2 \in K\}$.

$\mathcal{D}(K)$ has a number of interesting properties. To start with, the difference body of any object is centrally symmetric in the origin:

Theorem 2.1.7 $\mathcal{D}(K)$ *is centrally symmetric in origin* $\mathbf{0}$, *for any* $K \subset \mathbb{R}^d$.

PROOF. $\mathcal{D}(K) = \{k_1 - k_2 \mid k_1, k_2 \in K\}$, so for any point $p \in \mathcal{D}(K)$, there exist $k_1, k_2 \in K$ such that $k_1 - k_2 = p$. But $k_2 - k_1 = -p$ is also element of $\mathcal{D}(K)$, so $\forall (p \in \mathcal{D}(K) :: -p \in \mathcal{D}(K))$, hence $\mathcal{D}(K)$ is centrally symmetric in $\mathbf{0}$. ⊠

An important property of the difference body is that $\mathcal{D}(K)$ consists of all points a for which holds that K and K_a have points in common (see fig. 2.6). Otherwise, when a is located outside the difference body of K, then K and K_a will be disjunct. This is formalized in the following theorem. Later, we will see that K and K_a *touch* each other when a is located on the boundary of $\mathcal{D}(K)$ (see fig. 2.5).

Theorem 2.1.8 $a \in \mathcal{D}(K)_b \Leftrightarrow K_a \cap K_b \neq \emptyset$, *for any* $a, b \in \mathbb{R}^d, K \subset \mathbb{R}^d$.

PROOF. If $a \in \mathcal{D}(K)_b$, then $a - b \in \mathcal{D}(K)$. So there exist $k_1, k_2 \in K$ such that $k_1 - k_2 = a - b$. We can write this as $k_1 + b = k_2 + a$. Since $k_1, k_2 \in K$, it follows that $k_1 + b \in K_b$ and $k_2 + a \in K_a$, so $K_a \cap K_b \neq \emptyset$. ⊠

To construct the difference body of an object, we only need its boundary:

Theorem 2.1.9 $\mathcal{D}(K) = \mathcal{D}(\partial K)$, *for any* $K \in \mathcal{K}^d$.

PROOF. If we take a set $S_1 = \{a \in \mathbb{R}^d \mid K_a \cap K \neq \emptyset\}$, then it follows from theorem 2.1.8 that $S_1 = \mathcal{D}(K)$. From lemma 2.1.2 we can deduce that $K_a \cap K \neq \emptyset \Leftrightarrow \partial K_a \cap \partial K \neq \emptyset$, so the set $S_2 = \{a \in \mathbb{R}^d \mid \partial K_a \cap \partial K \neq \emptyset\} = S_1$. From theorem 2.1.8, it follows that $S_2 = \mathcal{D}(\partial K)$, hence $\mathcal{D}(K) = \mathcal{D}(\partial K)$. ⊠

We explicitly use this fact for the actual construction of the difference body, which we discuss in chapter 4.

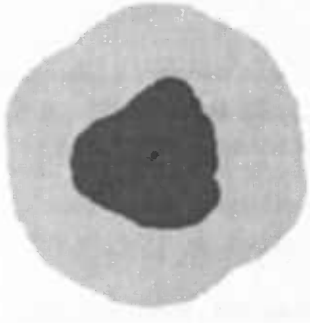

Figure 2.4: An object (dark) and its difference body (light).

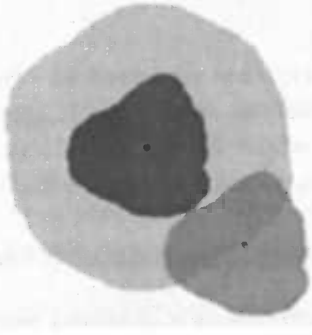

Figure 2.5: Objects K and K_a touch each other when $a \in \partial\mathcal{D}(K)$.

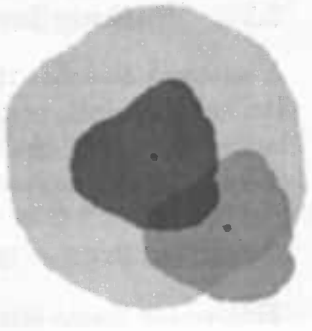

Figure 2.6: Objects K and K_a overlap when $a \in \Im\mathcal{D}(K)$.

2.1.4 Touching and Overlapping

Now we have proven some key properties of difference body $\mathcal{D}(K)$ of K, we can precisely define what 'touching' and 'overlapping' means in terms of $\mathcal{D}(K)$. When $a \in \mathcal{D}(K)_b$, we distinguish between two cases:

- Objects K_a and K_b *touch* each other if and only if $a \in \partial\mathcal{D}(K)_b$, for any $a, b \in \mathbb{R}^d$ (see fig. 2.5).
- Objects K_a and K_b *overlap* if and only if $a \in \Im\mathcal{D}(K)_b$, for any $a, b \in \mathbb{R}^d$ (see fig. 2.6).

2.2 Lattices and Packings

2.2.1 Lattices

Definition 2.2.1 A *lattice* $\Lambda \subset \mathbb{R}^d$ with basis $L = \{l_1, \ldots, l_d\}$, where $l_1, \ldots, l_d \in \mathbb{R}^d$ are linearly independent, is defined as the set of points $\Lambda = \{z_1 l_1 + \cdots + z_d l_d \mid z_1, \ldots, z_d \in \mathbb{Z}\}$.

$l_1, \ldots, l_d$ are called the lattice vectors of a lattice Λ (see fig. 2.7). The determinant $\det(\Lambda)$ of Λ is the volume of the parallelepiped spanned by these lattice vectors, i.e.: $\det(\Lambda) = |\det(L)|$.

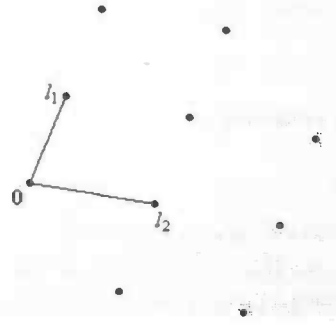

Figure 2.7: A lattice Λ with lattice vectors l_1 and l_2.

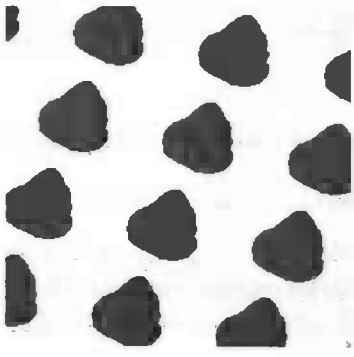

Figure 2.8: A packing lattice Λ for object K.

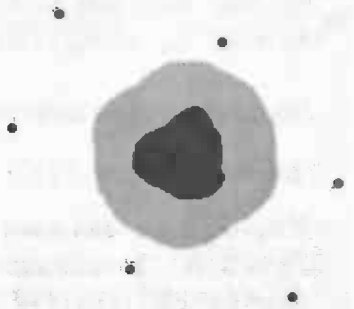

Figure 2.9: An admissible lattice with respect to $\mathcal{D}(K)$.

2.2.2 Packing lattices

A lattice Λ that has the property that copies of an object K do not overlap when placed at the lattice points, is called a *packing lattice* for K (see fig. 2.8). In other words, a packing lattice for K is a lattice Λ for which K_a and K_b do not overlap for any $a, b \in \Lambda, a \neq b$. For the exact definition, we need the notion of *admissibility*:

Definition 2.2.2 A lattice Λ is called *admissible* for object K if $\Lambda \cap \Im K = \{\mathbf{0}\}$.

This means that a lattice is admissible for K iff all points of the lattice except the origin lie outside $\Im K$. From section 2.1.4 we know that an object K_a does not overlap K when a lies outside $\Im\mathcal{D}(K)$. So when a lattice is admissible for $\mathcal{D}(K)$, no copy of K placed at a lattice point other than the origin, will overlap K (see fig. 2.9). This leads to the precise definition of packing lattices:

Definition 2.2.3 A lattice Λ is called a *packing lattice* for K, iff Λ is admissible for $\mathcal{D}(K)$.

We can prove that in a packing lattice for K, no two copies of K placed at arbitrary distinct lattice points overlap. This links up with the intuitive meaning of packing lattices:

Corollary 2.2.4 *If Λ is a packing lattice for K, then K_a and K_b do not overlap for any $a, b \in \Lambda, a \neq b$.*

PROOF. If Λ is a packing lattice for K, then $\Lambda \cap \Im\mathcal{D}(K) = \{\mathbf{0}\}$. This implies that $\forall(\lambda \in \Lambda, \lambda \neq \mathbf{0} :: \lambda \notin \Im\mathcal{D}(K))$. Now choose some $a, b \in \Lambda, a \neq b$, then $a - b \in \Lambda$ and $a - b \neq \mathbf{0}$.

Hence, $a - b \notin \Im\mathcal{D}(K)$. This is equivalent to $a \notin \Im\mathcal{D}(K)_b$. So, for any $a, b \in \Lambda, a \neq b$, K_a and K_b do not overlap. ⊠

There always exists a packing lattice Λ for K with minimal determinant. Such a packing lattice for K is called the *densest packing lattice* $\Lambda^*(K)$ for K (see fig. 2.10):

Definition 2.2.5 The *densest packing lattice* $\Lambda^*(K)$ for K is defined as the packing lattice for K with minimal determinant.

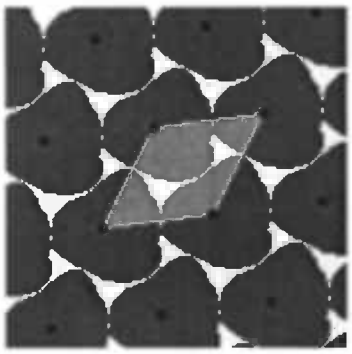

Figure 2.10: The densest packing lattice Λ^* for K. The area of the parallelogram, i.e. $\det(\Lambda^*)$, is minimal.

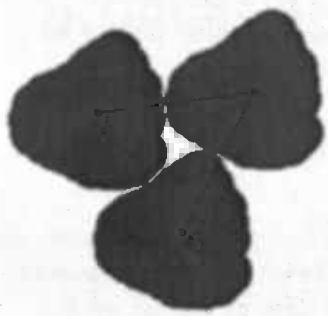

Figure 2.11: A 2D impression of a 'type 1'-touching packing lattice; objects K, K_{l_1}, K_{l_2} touch each other.

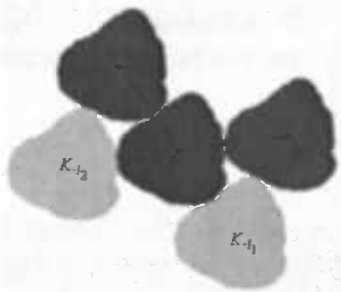

Figure 2.12: A 2D impression of a 'type 2'-lattice; objects K_{l_1} and K_{l_2} touch K and the negative counterparts of each other.

2.2.3 Touching packing lattices

So far we discussed general objects and packing lattices. Now we will focus on 3D objects and their packing lattices. It has been shown that for a special class of 3D objects, notably convex objects, a densest packing lattice has the following property:

Theorem 2.2.6 *For any convex 3D object K, there exists a densest packing lattice $\Lambda^*(K)$ with basis $L = \{l_1, l_2, l_3\}$, for which one of the two following cases holds:*

- $\{l_1, l_2, l_3, l_2 - l_3, l_1 - l_3, l_1 - l_2\} \subset \partial\mathcal{D}(K)$. *This means that the objects K, K_{l_1}, K_{l_2} and K_{l_3} touch each other.*
- $\{l_1, l_2, l_3, l_2 + l_3, l_1 + l_3, l_1 + l_2\} \subset \partial\mathcal{D}(K)$. *This means that the objects K_{l_1}, K_{l_2} and K_{l_3} touch K and the negative counterparts of each other.*

This theorem was already proved by Minkowski in 1904 [9].

We call a packing lattice for which one of the cases above holds a *touching packing lattice.* When the first case holds, it is called a 'type 1'-touching packing lattice (see fig. 2.11) and similarly when the second case holds, it is called a 'type 2'-lattice (see fig. 2.12).

Unfortunately, any properties like those above were never derived for densest packing lattices of non-convex objects, and, as we stated in the introduction, it is unlikely that this will happen in the near future. We therefore make one clear assumption: the touching packing

lattice with minimal determinant is assumed to be a *near-densest packing lattice* for a general 3D object K.

In the next chapter, we show how we can construct such near-densest packing lattices.

2.2.4 Lattice packings

In this chapter we discussed *packing lattices*, while we talk of *lattice packings* in the title and the introduction of this paper. There is a close relation between lattice packings and packing lattices. A lattice packing of an object K can be generated from a packing lattice Λ for the object, and can be defined as $\Lambda \oplus K$. Obviously, constructing a lattice packing of an object K is equivalent to constructing a packing lattice for K. In the remainder of this paper, we use the notion of packing lattices.

Chapter 3

Finding near-densest packing lattices

In this chapter, we show how we can construct a near-densest packing lattice for an arbitrary object K. First, we show how we can construct a valid packing lattice for object K. Later, we see how we can find a near-densest packing lattice by searching through the set of all packing lattices.

3.1 Difference body representation

Until now, we dealt with objects (and difference bodies of objects) as mathematically defined sets. This is, however, not a good representation for computational methods. A natural way to describe an object in a computer, is by means of a polyhedral approximation of the object (see fig. 3.1). A polyhedron is defined as a set of vertices, straight edges and planar facets. The facets divide the space into interior and exterior components. The set of facets itself can be seen as the boundary ∂K of object K. We shall not try to give a precise, formal definition of a polyhedron. Giving such a definition is tricky and not necessary in this context.

The facets of the polyhedron must all be triangular. We use this property for efficiently computing intersections between polyhedra, as we will see later. It is always possible to fulfil this requirement by triangulating polygonal facets.

Figure 3.1: An example of a polyhedral object K.

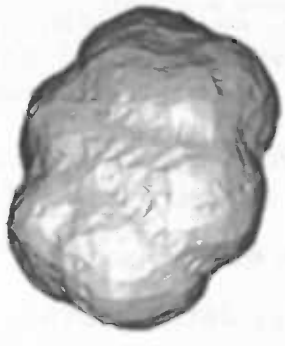

Figure 3.2: A polyhedral approximation of the difference body $\mathcal{D}(K)$ of K. Note that the scaling is different with respect to fig. 3.1.

For now, we assume that we have a polyhedral approximation of the difference body $\mathcal{D}(K)$ of object K (see fig. 3.2). In the next chapter, we discuss how this difference body is actually constructed.

3.2 Constructing a packing lattice

In section 2.2.3 we introduced the notion of a touching packing lattice for an object K. In a 'type 1'-touching packing lattice, the object on the origin and the ones defining the lattice vectors touch each other. Using a polyhedral difference body of K, we can construct such a packing lattice incrementally. The same can be done for 'type 2'-packing lattices.

3.2.1 Placing difference bodies

For 'type 1'-packing lattices, we start by placing an object K at the origin (see fig. 3.3). We now have to find three possible positions for translated copies K_{l_1}, K_{l_2} and K_{l_3} of this object such that these copies touch object K and each other. The positions of these three translated copies define the lattice vectors l_1, l_2 and l_3 of a possible packing lattice.

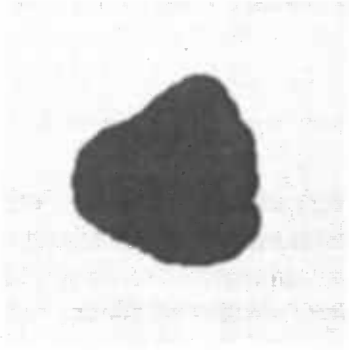

Figure 3.3: A 2D object K.

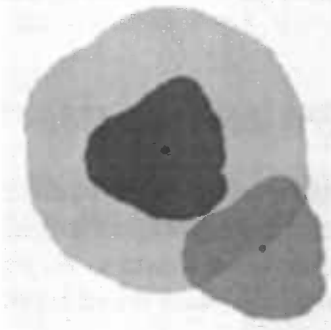

Figure 3.4: l_1 must be chosen anywhere on $\partial\mathcal{D}(K)$.

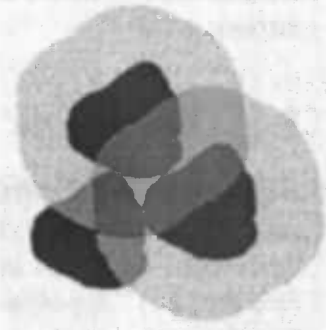

Figure 3.5: l_2 must be chosen on $\partial\mathcal{D}(K) \cap \partial\mathcal{D}(K)_{l_1}$.

The first translated copy K_{l_1} must touch K, so l_1 can be chosen anywhere on the boundary of difference body $\mathcal{D}(K)$ of K (see fig. 3.4). The second translated copy K_{l_2} has to be placed such that it touches K and K_{l_1}. Hence, l_2 must lie on the boundaries of both $\mathcal{D}(K)$ and $\mathcal{D}(K)_{l_1}$. In other words: it must be chosen on the intersection of the boundaries of these difference bodies (see fig. 3.5). Similarly, for the third translated copy K_{l_3}, l_3 must be chosen on the intersection of the boundaries of $\mathcal{D}(K)$, $\mathcal{D}(K)_{l_1}$ and $\mathcal{D}(K)_{l_2}$, hence:

Observation 3.2.1 A possible 'type 1'-packing lattice with basis $L = \{l_1, l_2, l_3\}$ for object K can be found by finding l_1, l_2, l_3 such that

- $l_1 \in \partial\mathcal{D}(K)$ and
- $l_2 \in \partial\mathcal{D}(K) \cap \partial\mathcal{D}(K)_{l_1}$ and
- $l_3 \in \partial\mathcal{D}(K) \cap \partial\mathcal{D}(K)_{l_1} \cap \partial\mathcal{D}(K)_{l_2}$.

In more or less the same way, we can incrementally construct a 'type 2'-packing lattice:

Observation 3.2.2 A possible 'type 2'-packing lattice with basis $L = \{l_1, l_2, l_3\}$ for object K can be found by finding l_1, l_2, l_3 such that

- $l_1 \in \partial\mathcal{D}(K)$ and
- $l_1 + l_2 \in \partial\mathcal{D}(K) \cap \partial\mathcal{D}(K)_{l_1}$ and
- $l_1 + l_2 + l_3 \in \partial\mathcal{D}(K)_{l_1} \cap \partial\mathcal{D}(K)_{l_2} \cap \partial\mathcal{D}(K)_{l_1+l_2}$.

For both 'type 1'- and 'type 2'-packing lattices, a basis $L = \{l_1, l_2, l_3\}$ found in the way described above does not always span an admissible packing lattice for K. So, we have to check explicitly whether the lattice is admissible for $\mathcal{D}(K)$ (cf. definition 2.2.3.).

3.2.2 Checking admissibility

A lattice Λ with basis $L = \{l_1, l_2, l_3\}$ is admissible for $\mathcal{D}(K)$ if $\mathbf{0}$ is the only point of the lattice that is element of $\Im\mathcal{D}(K)$ (see definition 2.2.2). Now let $(\alpha, \beta, \gamma)L$ denote the lattice point $\alpha l_1 + \beta l_2 + \gamma l_3$ for $\alpha, \beta, \gamma \in \mathbb{Z}$ and let $\{(\alpha, \beta, \gamma)L \mid |\alpha|, |\beta|, |\gamma| \leq k\}, k \in \mathbb{Z}$ be the set of all k-th order lattice points.

Of the first order lattice points, we know that for a 'type 1'-packing lattice $(1,0,0)L$, $(0,1,0)L$, $(0,0,1)L$, $(0,1,-1)L$, $(1,0,-1)L$, $(1,-1,0)L \in \partial\mathcal{D}(K)$, since the objects placed at $0, l_1, l_2$ and l_3 touch each other (see also theorem 2.2.6). Since $\mathcal{D}(K)$ is centrally symmetric in $\mathbf{0}$, we know that the negative counterparts of these lattice points are also in $\partial\mathcal{D}(K)$, so the only first order lattice points for which we have to check explicitly whether or not they are in $\Im\mathcal{D}(K)$ are $(-1,1,1)L$, $(1,-1,1)L$, $(1,1,-1)L$, $(1,1,0)L$, $(0,1,1)L$, $(1,0,1)L$ and $(1,1,1)L$.

For a 'type 2'-packing lattice, we must check $(-1,1,1)L$, $(1,-1,1)L$, $(1,1,-1)L$, $(1,-1,0)L$, $(0,1,-1)L$, $(1,0,-1)L$ and $(1,1,1)L$.

If these lattice points are indeed not in $\Im\mathcal{D}(K)$, we would be sure for convex $\mathcal{D}(K)$ that the lattice Λ is admissible for $\mathcal{D}(K)$. For non-convex objects, however, there could exist weird shaped difference bodies which do not contain any first order lattice point in its interior, but do contain higher order points. This is hard to see, for there is no 2D analogy; in the 2D case it is topologically impossible that objects placed at higher order lattice points overlap the object at the origin, because their path to the central object is blocked by touching first order objects. In the 3D case, we have to check explicitly if there is any higher order lattice point that is contained in $\Im\mathcal{D}(K)$. Therefore, every lattice point falling in a bounding box around $\mathcal{D}(K)$ is checked for being in $\Im\mathcal{D}(K)$. As we will see later, we use a triangulation hierarchy data-structure for representing $\mathcal{D}(K)$, which makes it possible to classify a point as being interior or exterior very quickly.

If there is no first or higher order lattice point in $\Im\mathcal{D}(K)$, then the lattice Λ is admissible for $\mathcal{D}(K)$, which means that we have found a (touching) packing lattice for K.

3.3 Calculating intersections

In section 3.2.1, we have seen that we have to compute the intersection of the boundaries of difference bodies to construct a packing lattice. For this purpose, we use the RAPID collision detection method of Gottschalk [14]. This method uses an OBB-tree (object bounding box-tree) data-structure to rapidly detect collisions between two sets of triangles.

We have to compute the intersection between $\partial\mathcal{D}(K)$ and $\partial\mathcal{D}(K)_{l_1}$ in the second step of constructing a touching packing lattice (both 'type 1' and 'type 2', see observations 3.2.1 and 3.2.2). Since the boundary of the polyhedral representation of $\mathcal{D}(K)$ consists of triangles, we can use $\partial\mathcal{D}(K)$ and $\partial\mathcal{D}(K)_{l_1}$ as the two sets for collision detection. The method reports every pair of triangles from both sets that do collide. For each pair of intersecting triangles, we can compute the intersection line segment. All these line segments together form the intersection of $\partial\mathcal{D}(K)$ and $\partial\mathcal{D}(K)_{l_1}$.

In the next step, we have to compute the intersection of the boundaries of three difference bodies. For 'type 1'-packing lattices these are $\partial\mathcal{D}(K)$, $\partial\mathcal{D}(K)_{l_1}$ and $\partial\mathcal{D}(K)_{l_2}$. Since we already computed the intersection of the boundaries of the first two difference bodies, we only have compute the intersection of the line segments and the boundary of the third difference body. Because the RAPID-method does not take line segments as input, we construct small triangles on each line segment, before detecting the collisions. For each pair of colliding line segments and difference body triangles, we compute the intersection point. These points together form the intersection of $\partial\mathcal{D}(K)$, $\partial\mathcal{D}(K)_{l_1}$ and $\partial\mathcal{D}(K)_{l_2}$.

For 'type 2'-packing lattices, this step is a little more complicated. We also have to compute the intersection of the boundaries of three difference bodies; $\partial\mathcal{D}(K)_{l_1} \cap \partial\mathcal{D}(K)_{l_2} \cap \partial\mathcal{D}(K)_{l_1+l_2}$, but none of these intersections was calculated in a previous step. So we have to perform two intersections to find the points in $\partial\mathcal{D}(K)_{l_1} \cap \partial\mathcal{D}(K)_{l_2} \cap \partial\mathcal{D}(K)_{l_1+l_2}$.

3.4 Finding a near-densest packing lattice

Now that we have seen how to construct a single touching packing lattice, we can compute a near-densest touching packing lattice. We simply do this by brute force.

3.4.1 Degrees of freedom

From observations 3.2.1 and 3.2.2 we see that in the first step of constructing a touching packing lattice, we may choose l_1 somewhere on the boundary of $\mathcal{D}(K)$. Since $\mathcal{D}(K)$ is a three-dimensional volume, its boundary $\partial\mathcal{D}(K)$ is a surface that can be parametrized in two dimensions (see fig. 3.2), so there are two degrees of freedom for choosing l_1.

For 'type 1'-packing lattices we may choose l_2 in the second step somewhere on the intersection of $\partial\mathcal{D}(K)$ and $\partial\mathcal{D}(K)_{l_1}$. As we have seen in section 3.3, this intersection is in general, when discarding coplanarities, a curve in 3D space (see fig. 3.6). So for choosing l_2, there is one degree of freedom. For 'type 2'-packing lattices we must choose $l_1 + l_2$ on the same curve, which also yields one degree of freedom.

In the last step we choose, for 'type 1'-packing lattices, l_3 on the intersection of $\partial\mathcal{D}(K)$, $\partial\mathcal{D}(K)_{l_1}$ and $\partial\mathcal{D}(K)_{l_2}$. When discarding colinearities, this intersection is a small set of points in 3D space (see fig. 3.7), so for choosing l_3 we have no degree of freedom. For 'type 2'-packing lattices, we also have no degrees of freedom in the third step.

So finding a near-densest packing lattice is for both types a search problem in three variables.

3.4.2 Searching the parameter space

We cannot, of course, inspect all points on the boundary of $\mathcal{D}(K)$ for choosing l_1, since there are infinitely many of those. Ideally, one inspects a fixed number of points distributed

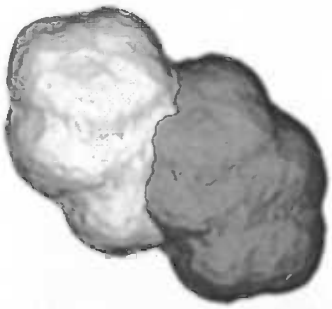

Figure 3.6: The intersection of $\partial\mathcal{D}(K)$ and $\partial\mathcal{D}(K)_{l_1}$ is a curve in 3D.

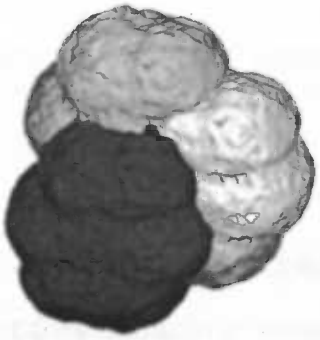

Figure 3.7: The intersection of $\partial\mathcal{D}(K)$, $\partial\mathcal{D}(K)_{l_1}$ and $\partial\mathcal{D}(K)_{l_2}$ is a small set of points (dark dot).

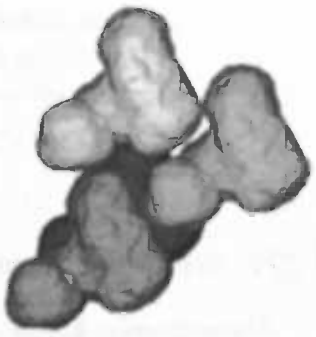

Figure 3.8: The objects K, K_{l_1}, K_{l_2} and K_{l_3} in a near-densest packing lattice for K.

homogeneously over the boundary of $\mathcal{D}(K)$.

We chose to take the vertices of the polyhedron representing $\mathcal{D}(K)$ as the set of points to inspect. In the next chapter, we show that these vertices are fairly homogeneously distributed over the surface of $\mathcal{D}(K)$ and that the number of vertices is more or less controllable. So the granularity of our search is dictated by the granularity of the triangulation on the polyhedral surface. The symmetry of $\mathcal{D}(K)$ is exploited by only inspecting half of the vertices, for example the vertices with x-coordinate greater than zero.

On the intersection of $\partial\mathcal{D}(K)$ and $\partial\mathcal{D}(K)_{l_1}$, a curve in 3D, we have the same problem. Here we choose the end points of each segment contained in the intersection as the set of possible positions for l_2 ('type 1') or $l_1 + l_2$ ('type 2').

The intersection of $\partial\mathcal{D}(K)$, $\partial\mathcal{D}(K)_{l_1}$ and $\partial\mathcal{D}(K)_{l_2}$ is a finite set of points, so we can inspect all possible positions for l_3 in a 'type 1'-packing lattice. The same holds for 'type 2'-packing lattices.

For every combination of l_1, l_2 and l_3 we inspect, the volume of the parallelepiped spanned by these lattice vectors is computed and the lattice Λ spanned by basis $L = \{l_1, l_2, l_3\}$ is checked for admissibility with respect to $\mathcal{D}(K)$. If Λ is admissible for $\mathcal{D}(K)$ and the volume of this lattice is smaller than any other calculated volume, we have found a near-densest packing lattice for K (see fig. 3.8).

3.4.3 Algorithm outline

The algorithm outline for finding a 'type 1'-near-densest packing lattice Λ^* for an object K is as follows. The input of the algorithm must be a polyhedral representation of $\mathcal{D}(K)$:

Algorithm 3.4.1 NEARDENSESTPACKINGLATTICETYPE1($\mathcal{D}(K)$)

$v^* \leftarrow \infty$
for all vertices $l_1 \in \mathcal{D}(K)$ **do**
  $I_1 \leftarrow \partial\mathcal{D}(K) \cap \partial\mathcal{D}(K)_{l_1}$
  for all end points l_2 of line segments $\in I_1$ **do**
    $I_2 \leftarrow I_1 \cap \partial\mathcal{D}(K)_{l_2}$
    for all points $l_3 \in I_2$ **do**
      $\Lambda \leftarrow$ the lattice spanned by $\{l_1, l_2, l_3\}$

$v \leftarrow \det(\Lambda)$
if Λ is admissible for $\mathcal{D}(K)$ **and** $v < v^*$ **then**
$v^* \leftarrow v$
$\Lambda^* \leftarrow \Lambda$
return Λ^*

For 'type 2'-near-densest packing lattices, we get:

Algorithm 3.4.2 NEARDENSESTPACKINGLATTICETYPE2($\mathcal{D}(K)$)
$v^* \leftarrow \infty$
for all vertices $l_1 \in \mathcal{D}(K)$ **do**
$I_1 \leftarrow \partial\mathcal{D}(K) \cap \partial\mathcal{D}(K)_{l_1}$
for all end points $l_1 + l_2$ of line segments $\in I_1$ **do**
$I_2 \leftarrow \partial\mathcal{D}(K)_{l_1} \cap \partial\mathcal{D}(K)_{l_2}$
$I_3 \leftarrow I_2 \cap \mathcal{D}(K)_{l_1+l_2}$
for all points $l_1 + l_2 + l_3 \in I_3$ **do**
$\Lambda \leftarrow$ the lattice spanned by $\{l_1, l_2, l_3\}$
$v \leftarrow \det(\Lambda)$
if Λ is admissible for $\mathcal{D}(K)$ **and** $v < v^*$ **then**
$v^* \leftarrow v$
$\Lambda^* \leftarrow \Lambda$
return Λ^*

These algorithms can be integrated of course, but for simplicity both algorithms are shown independently.

In practice, the 'type 2'-packing lattice algorithm proved not to be very useful. We tested the 'type 2'-algorithm on some objects, but we never found a 'type 2'-lattice that is admissible for even the first order lattice points. Furthermore, an additional intersection of two difference bodies is performed in the most inner loop of the algorithm. Hence, it takes much more time than the 'type 1'-algorithm.

For the 'type 1'-algorithm, it turns out in practice, that it is not necessary to check the admissibility for second or higher order lattice points. Without this check, the method is a lot faster, but theoretically incorrect. This seems not to harm; our tests show that even for weirdly shaped objects, the calculated packing lattices are admissible and correct.

Chapter 4

Constructing difference bodies

In the previous chapter, we saw that we can compute a near-densest packing lattice for object K if we have a polyhedral representation of the difference body $\mathcal{D}(K)$ of K. In this chapter, we discuss how to construct such an approximation for $\mathcal{D}(K)$. There is much literature about computing Minkowski sums (and consequently difference bodies), but most literature only concerns convex [7] or two-dimensional [2, 13] objects. In mathematical morphology, there exists a Minkowksi sum algorithm for 3D binary voxel objects [17], but it is not very practical for our application. We will therefore present our own method for approximating difference bodies in this chapter.

4.1 Object representation

4.1.1 Polyhedra

As we said in the previous chapter, the most natural way to describe an object in a computer, is by means of a polyhedral approximation of the object (see fig. 4.1). So, suppose that we have approximated an object by a polyhedron, the second step is to compute its difference body. Let n be the number of vertices of the polyhedron, then, if the polyhedron is convex, it takes $\mathcal{O}(n)$ time to compute the difference body [7]. For non-convex polyhedra, we don't know what the upper bound running time is. In case of a 2D polygon, it takes $\mathcal{O}(n^4)$ time to compute the difference body [8]. For the 3D case, it is probably even worse. In practice, it is not feasible to compute the difference body of any polyhedron but the smallest ones. We therefore have to look for another object representation.

4.1.2 Point sets

A less intuitive way to represent an object is by means of a cloud of points or a point set (see fig. 4.2). This cloud of points can approximate an object with a certain shape. We do not have to consider the points that are 'inside' the object, since theorem 2.1.9 tells us that we only need the boundary of an object to compute its difference body. So suppose that we approximate the boundary of an object using n points, then we can compute a point-set approximation of the difference body in $\mathcal{O}(n^2)$ time. This follows immediately from the definition of difference bodies (see definition 2.1.6).

So, in contrast to a polyhedral representation, a point set representation allows us to compute the difference body approximation in reasonable running time. If one still wants to

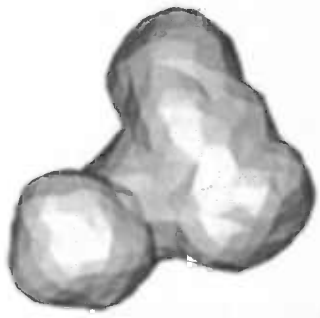

Figure 4.1: A polyhedral representation of object K.

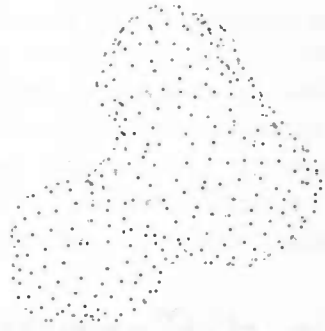

Figure 4.2: A point set representation of the boundary of K (638 points).

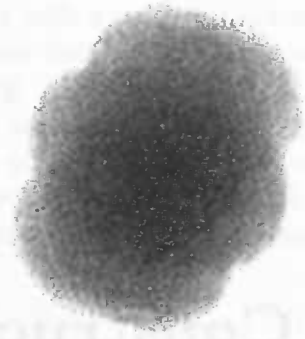

Figure 4.3: A dense cloud of points approximating $\mathcal{D}(K)$ (407044 points).

approximate the difference body of an object represented by a polyhedron, then the vertices of this polyhedron might serve as the point set approximating the boundary of the object.

4.2 Calculating the difference body

When we represent an object using a point set, the difference body is approximated by a point set as well. Yet, from the previous chapter it is clear that we need a polyhedral difference body, since we want to know exactly what is inside, what is outside and what is on the difference body. We therefore have to reconstruct a polyhedral approximation of the difference body from the point set we have calculated. This can be done using a surface reconstruction technique.

4.2.1 Grid filtering

A common problem for surface reconstruction techniques is that they cannot cope with very large point sets. The cloud of points that approximates the difference body is in general very dense, since the number of points in this cloud is the square of the number of points approximating the object (see fig. 4.3). So somehow, we have to reduce the number of points in the difference body point cloud.

A naive solution for this problem is selecting just a fixed number of points randomly from the point cloud. The new point set would still approximate the difference body quite nicely in general, but the shape of the body has changed a bit, since many points on the boundary of the difference body have been left out. Another drawback is that we can never be sure that we have a nice approximation, since it is possible, although the chances are small, that we only selected points from a specific part of the difference body. We can make this a bit better by taking the symmetry of the difference body into account, but still we could get an inhomogeneously distributed selection.

We therefore use another method to filter points from the dense cloud of points, namely grid filtering. This works as follows:

First, we construct an axis aligned bounding cube covering all points of the cloud (see fig. 4.4). This cube is centred at the origin, since the difference body is centrally symmetric in $\mathbf{0}$. We can find the size of the cube by iterating over all points.

We then divide this cube into a number of equally sized cubic cells. The idea of grid filtering is that each cell provides one point to the filtered point set, if possible. We can then guarantee that we get a fairly homogeneously distributed selection from the cloud of points. To guarantee that the shape of the difference body remains unchanged, each cell provides a boundary point of the difference body when that is possible.

We can detect whether a cell contains boundary points by investigating whether one of its 26 neighbour cells is empty (see fig. 4.4). If this is the case, then the cell contains a boundary point. The point in this cell that lies farthest along the normal vector on the surface of the difference body, is taken as the point this cell provides to the filtered point set.

We can, of course, only approximate the normal vector on the surface of the difference body. The normal vector is approximated per cell by averaging the vectors pointing from the cell towards each empty neighbour (see fig. 4.5).

An interior cell has no empty neighbours, i.e. the cell contains no boundary points, so the normal vector is a null vector. It depends on the used surface reconstruction technique whether a point should be selected from such cells; some reconstruction techniques only require boundary points. Others, like the one we use, require a point cloud, so for our application a point is randomly chosen from interior cells.

A neighbour cell that falls outside the bounding cube is by definition empty. We can detect the empty cells as a pre-processing step; each cell is initially defined empty, then we iterate over all points in the point cloud and the cells each of these points fall in are set non-empty.

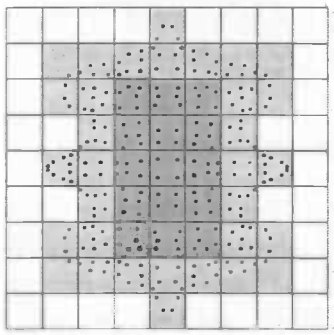

Figure 4.4: A grid covering all points of a symmetrical point cloud. The grid has empty grid cells (white), grid cells that contain a boundary point (grey) and cells not containing boundary points (dark).

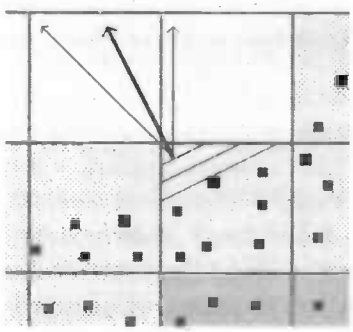

Figure 4.5: A normal (thick arrow) is approximated for each cell by averaging vectors pointing towards empty neighbours (thin arrows). The black points are taken as the point its cell provides to the filtered point set.

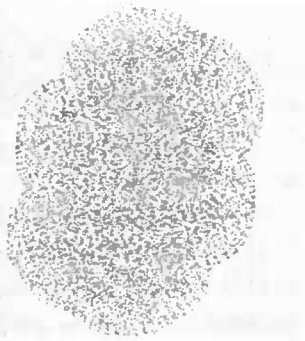

Figure 4.6: The dense cloud of points approximating $\mathcal{D}(K)$ (see fig. 4.3) filtered using a grid of $32 \times 32 \times 32$ cells, yielding a filtered point set of 10520 points.

The larger the number of grid cells is chosen, the better the reduced point set approximates $\mathcal{D}(K)$, but when the number of cells is chosen too large (and consequently the size of each cell too small), one can imagine that some cells 'inside' the point cloud will not contain any point. Then, our neighbour information has become useless. Hence, the number of cells may not be chosen too large.

Summarizing: for each cell we distinguish between three cases. When the cell is a boundary cell, we choose the point that lies maximal along the normal vector. If the cell is empty, it

does not provide a point to the filtered point set, and from interior cells a point is randomly chosen. This yields a filtered point set that is equally distributed over the original point cloud and contains enough boundary points to maintain its shape (see fig. 4.6).

The number of points in the filtered point set depends on the number of cells. So, when the number of grid cells is chosen appropriately, the number of points in the point cloud approximating $\mathcal{D}(K)$ is reduced such that surface reconstruction techniques can cope with it. Using such a technique, we can construct a polyhedral approximation of the difference body.

4.2.2 Surface reconstruction

As we said, we want to have a polyhedral approximation of the difference body. To this end, we reconstruct such an approximation from the filtered point set using a surface reconstruction technique. The reconstruction method must at least fulfil the following properties:

- The reconstructed surface must be a closed 'watertight' polyhedron, so that we can distinguish between the interior, exterior and boundary of the polyhedron.
- The reconstructed polyhedron must at least contain the points from the input set and the quality of the reconstruction must be good.
- The reconstructed surface must consist of triangular facets. We need this property to compute the intersections between difference bodies efficiently. This requirement can always be fulfilled by triangulating polygonal facets as a post-processing step.
- The algorithm for reconstructing a surface from a point set must be quick, and efficient in memory usage.
- The data-structure containing the reconstructed surface must be easily accessible.

We have experimented with a number of surface reconstruction methods, including the power crust method of Amenta [4], a Delaunay based reconstruction algorithm of Cohen-Steiner [10] and the alpha shape concept of Edelsbrunner and Mücke [11]. The first two methods do not guarantee to yield a closed polyhedron, which is the most important requirement. Only the alpha shape concept fulfils our requirements, although it is not very fast; the complexity of the alpha shape method is $\mathcal{O}(n^2)$, where n is the number of points in the input set. The alpha shape consists solely of triangles and its data-structure is easily accessible, for it is implemented in the computational geometry library CGAL [1]. In fact, we use a triangulation hierarchy data-structure for the alpha shape, so that it is possible to very quickly determine whether a point is inside or outside the object. We use this property among other things for checking admissibility (see section 3.2.2).

Intuitively, an alpha shape is constructed as follows. A ball with radius α rolls over the surface of our point set, and tries to penetrate it (see fig. 4.7). Every three points in the point set that can be touched simultaneously by the ball in this way, define a triangle that is part of the boundary of the alpha shape. All triangles thus constructed form together the boundary of the alpha shape of the input point set [11, 1].

The alpha shape method yields an entire family of shapes, each shape with a different value for α. The possible values for α range from zero to infinity with discrete steps, an α-value of zero yielding the input point set and an α-value of infinity yielding the convex hull of the input point set.

For our point sets, we choose the α-value equal to the length of the diagonal of the cells from the grid filtering method. One can easily see that a ball with this radius can not penetrate the point set; the largest possible distance between two neighbouring points is twice the cell diagonal, which equals the diameter of the α-ball. In this manner, a nice closed surface is reconstructed from the filtered point set (see fig. 4.8).

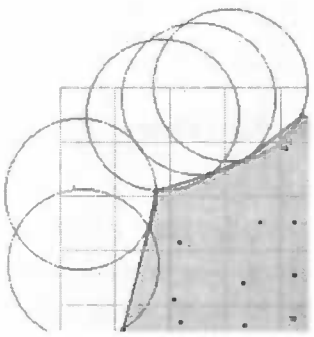

Figure 4.7: A ball rolling over the surface of a point cloud, thus forming an alpha shape (grey).

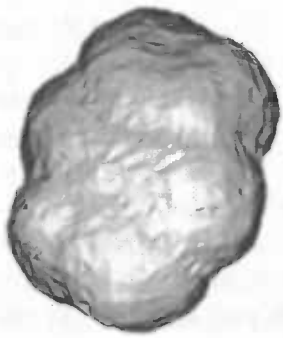

Figure 4.8: An alpha shape surface reconstruction of the filtered point cloud approximating $\mathcal{D}(K)$ (see fig. 4.6). It has 2400 vertices.

4.2.3 Triangulation granularity

In the previous chapter we made some assumptions about the granularity of the triangulation on the boundary of $\mathcal{D}(K)$ (see subsection 3.4.2). We assumed that the set of vertices of the polyhedron representing $\mathcal{D}(K)$ is fairly homogeneously distributed over the surface of $\mathcal{D}(K)$, and that the number of vertices is controllable in some way. We can now see that this indeed is the case. Since we calculated the alpha shape of a quite homogeneously distributed point set, one may expect that the set of vertices on the surface of the alpha shape is fairly evenly distributed as well. Also, the number of vertices of the alpha shape is controllable by varying the number of grid cells; a larger number of grid cells results in a larger number of vertices.

We may conclude that we have a polyhedral approximation of the difference body that fulfils all our demands. Using this difference body, a near-densest packing lattice can be computed.

4.2.4 Algorithm outline

The algorithm outline for constructing a polyhedral approximation $\mathcal{D}(K)$ of the difference body of an object K is as follows. The input of the algorithm must be a point set approximating ∂K:

Algorithm 4.2.1 DIFFERENCEBODY(∂K)
for all points $k_1 \in \partial K$ **do**
for all points $k_2 \in \partial K$ **do**
Add $k_1 - k_2$ to DK
Construct grid G covering all points in DK
Initialize all cells in G as being empty

```
for all points p ∈ DK do
    point(cell(p)) ← p
for all cells c ∈ G do
    normal(c) ← 0
    for all neighbour cells n of c do
        if n is empty then
            v ← vector pointing from c to n
            normal(c) ← normal(c) + v
for all points p ∈ DK do
    if p lies further along normal(cell(p)) than point(cell(p)) then
        point(cell(p)) ← p
for all cells c ∈ G do
    Add point(c) to DK*
α ← length of diagonal of grid cells
D(K) ← AlphaShape(DK*, α)
return D(K)
```

Chapter 5

An application: M.D. simulations

One of the major applications of our near-densest packing lattice algorithm lies in computational chemistry, to be more precise, in molecular dynamics (M.D.) simulations. In this chapter we give a short introduction in M.D. simulations and show how our algorithm can be used to speed up M.D. simulations dramatically.

5.1 Introduction

M.D. simulations are used to complement and to replace experiments in physics and chemistry. As such, M.D. has been used to study simple gases, liquids, polymers, crystals, liquid crystals, proteins, proteins in liquids, membranes, DNA-protein interactions, etc. The principle of M.D. simulations is very simple: the time development of a many atom system is evaluated by numerically integrating Newton's equations of motion [5]. The main concepts of the M.D. simulation technique are introduced in this section.

5.1.1 Main principle

The main principle of M.D. simulation is as follows: given the system state $S(t_0)$, that is, the position $\vec{r}$ and velocity $\vec{v}$ of every atom in the system at t_0, subsequent states $S(t_0 + \Delta t), S(t_0 + 2\Delta t), \ldots$ are calculated by using Newton's law $\vec{F} = m\vec{a}$. For accurate results small time-steps Δt have to be used. To calculate $S(t_0 + (n+1)\Delta t)$ from $S(t_0 + n\Delta t)$, first for every atom i in the system, $\vec{F}_i(t_0 + n\Delta t)$ is calculated. $\vec{F}_i(t_0 + n\Delta t)$ is the sum of the forces on i as exerted by the other atoms in the system at time $t_0 + n\Delta t$. For every atom i the force $\vec{F}_i(t_0 + n\Delta t)$ is then integrated to get the new velocity $\vec{v}_i(t_0 + n\Delta t)$. Using this velocity, for every atom i the new position $\vec{r}_i(t_0 + (n+1)\Delta t)$ can be calculated.

Generally speaking, in an M.D. simulation the forces between atoms only depend on atom positions, not on velocities. Usually, interactions are specified by giving an expression for the potential energy of the interaction. The force individual atoms exert on each other can then be calculated as the negative gradient of the potential.

5.1.2 Interaction potentials

Two classes of interactions may be distinguished: non-bonded interactions and bonded interactions. Non-bonded interactions model flexible interactions between atom pairs. Two non-bonded interaction potentials that are typically considered in M.D. simulations are the

Coulomb potential, which models electrostatic interactions between charged atoms, and the Lennard-Jones potential, which models basic interactions between atoms.

Bonded interactions model rather strong chemical bonds, and are not created or broken during a simulation. The three most widely used bonded interaction potentials are the covalent interaction potential, the bond-angle interaction potential and the dihedral interaction potential.

5.1.3 Cut-off radius

In principle, a non-bonded interaction exists between every atom pair. To reduce the CPU time an M.D. simulation takes, the use of a cut-off radius is a widely applied optimization.

Earlier we proposed to evaluate all pair interactions, no matter how far the atoms are separated. However, the main contribution to the total force exerted on a atom is from neighbouring atoms. Therefore, only a small error is introduced when only interactions are evaluated between atoms with a distance less than a fixed cut-off radius r_{co}. Choosing r_{co} so that for each atom 100 to 300 other atoms are within the cut-off radius gives a good balance between correct physics and efficiency. For a system of 10^4 atoms, this makes the non-bonded force computation a factor $\frac{10^4}{100}$ to $\frac{10^4}{300}$ faster.

Still, when using a cut-off radius, all pairs have to be inspected to see if their separation is less than r_{co}. However, the time-steps made are so small that the set of atoms within r_{co} of a given atom hardly changes during one time-step. Therefore, much CPU time can be saved when only every 10 or 20 time-steps all pairs are inspected to see if their distance is less than r_{co}, at the cost of some memory space to store a neighbour list of every atom.

5.1.4 Computational boxes

Many M.D. simulations involve the simulation of one large molecule, denoted by M, in a solvent. Since it is not possible to simulate a molecule in an infinitely large ocean, the molecule is placed in a simulation box B, for example a cube, filled with solvent. To prevent physically incorrect finite system effects, periodic boundary conditions are used. This means that an atom that leaves the box on one side, will enter the box again on the opposite side. This is equivalent to surrounding the computational box by an infinite number of identical replica boxes, stacked in a space filling manner (see fig. 5.1). Only the behaviour of one box has to be simulated; other boxes behave in the same way.

In a simulation, one is not interested in the behaviour of the solvent. Hence, we can use a computational box with a more complex shape than a cube, to reduce the amount of solvent to be simulated (see fig. 5.2). The shape of the computational box should be such that it can be stacked in a space filling way. In 3D space, there exist five convex box types with this property: the triclinic box, the hexagonal prism, the dodecahedron, the elongated dodecahedron and the truncated octahedron.

In an M.D. system with periodic boundary conditions, atoms are influenced by atoms in their own box and atoms in surrounding boxes. It is however, not desirable that atoms of molecule M interact with atoms of periodic copies of M. This would yield physically unsound behaviour that can best be described as a molecule 'biting itself in its tail'.

To this end, the computational box B must be constructed at a certain minimal distance from the molecule M. Obviously, this distance should at least be equal to $\frac{1}{2}r_{co}$. Now, let $\mathcal{P}(M, \frac{1}{2}r_{co})$ be the *parallel body* of M at distance $\frac{1}{2}r_{co}$, i.e. the molecule M dilated by a layer

of width $\frac{1}{2}r_{co}$. Then, B can be constructed by tightly enclosing $\mathcal{P}(M, \frac{1}{2}r_{co})$ by one of the space filling boxes. In this way, the distance between M and B is at least $\frac{1}{2}r_{co}$ (see fig. 5.3).

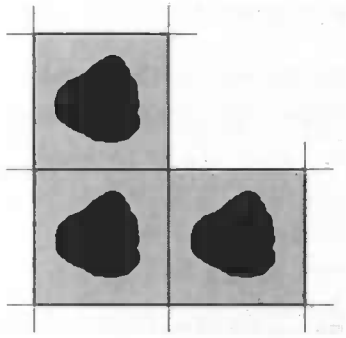

Figure 5.1: A molecule M in a cubic box and two of its periodic replicas.

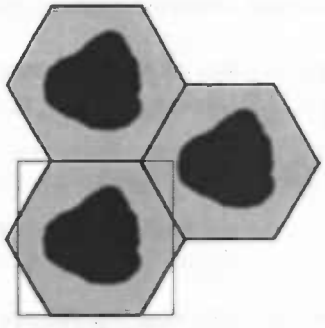

Figure 5.2: A computational box with a more complex shape. The cubic box is drawn for comparison.

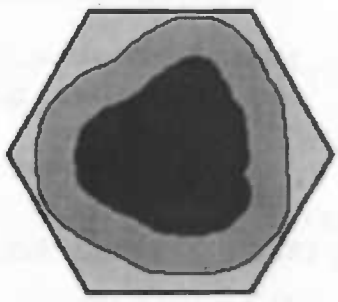

Figure 5.3: A box constructed tightly around $\mathcal{P}(M, \frac{1}{2}r_{co})$.

Let the space outside M and inside B be called C, so $C = B - M$, and let the space outside $\mathcal{P}(M, \frac{1}{2}r_{co})$ and inside B be called D, so $D = B - \mathcal{P}(M, \frac{1}{2}r_{co})$. After B has been constructed, M is placed in B and C is filled with solvent molecules. From the foregoing it will be clear that the solvent in D does not contribute to the simulation. Yet, per unit of volume, simulating it takes approximately the same CPU effort as simulating the atoms in $\mathcal{P}(M, \frac{1}{2}r_{co})$, so, denoting the volume of D by $\mathrm{vol}(D)$, we spend approximately $\frac{\mathrm{vol}(D)}{\mathrm{vol}(B)}$ of our CPU time on the simulation of irrelevant solvent. For oblong molecules, this ratio can be as high as 80%, so a huge amount of CPU time can be saved by minimizing the computational box volume.

5.2 Minimizing the computational box volume

So far we discussed current M.D. practice. In this section we show a new method for minimizing the computational box using our near-densest packing lattice algorithm. The CPU time required for an M.D. simulation can be reduced dramatically by using this minimal box.

5.2.1 Rotational restraining

During an M.D. simulation the molecule M rotates in an erratic way, resulting in a number of full rotations during a typical simulation. Besides rotational motion M may also show conformational changes. Often these changes are of major interest and should not be constrained in any way. In current M.D. practice, one of the five space fillers is used for the computational box B, so that M may rotate freely in B and may also show some conformational changes without leaving B. Often the additional space in B allowing for conformational changes is created by taking the width of the layer around M a bit larger than $\frac{1}{2}r_{co}$.

One can imagine that in a near-minimal-volume computational box there is not enough space for rotational motion of M, so for a near-minimal volume M.D. simulation two ingredients are essential:

1. A method to restrain the rotational motion of M during the simulation without affecting the conformational changes of M.

2. A method to construct a computational box of which the volume is nearly minimal.

Fortunately we do not have to take care of the first requirement, because such a method has been developed by Amadei et al. [3], among other things with the goal to enable minimal volume simulations.

The second requirement is taken care of in this section.

5.2.2 Triclinic boxes

As we said in the previous section, the computational box may have five different shapes. Bekker has shown that simulations using one of these box types can always be transformed to a simulation using a triclinic box, i.e. a parallelepiped [6].

It works as follows: when space is tesselated with a box of one of the types discussed above, a lattice Λ, with lattice vectors l_1, l_2 and l_3, is defined. These lattice vectors span a triclinic box that can be used just as well as the original box for the simulation (see fig. 5.4). In fact, a simulation in the triclinic box is exactly equivalent to a simulation in the original box.

This observation is essential for using the near-densest packing lattice for minimizing the computational box volume.

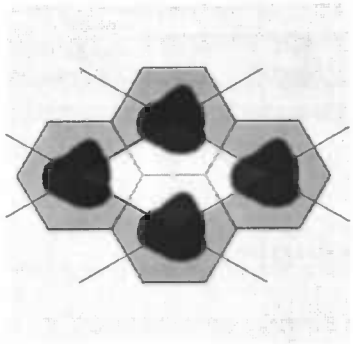

Figure 5.4: A triclinic box spanned by the lattice vectors (thick) of the lattice defined by tesselating space with the original box.

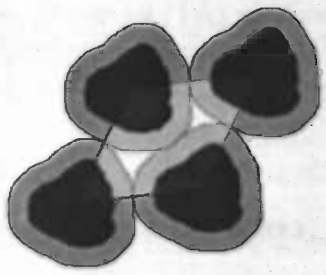

Figure 5.5: A near-minimal computational box spanned by the lattice vectors (thick) of the near-densest packing lattice for $\mathcal{P}(M,r)$.

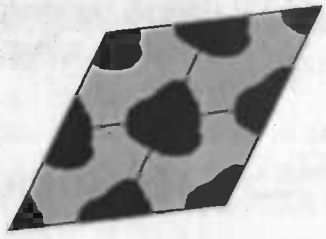

Figure 5.6: The fragmented molecule M is reconstructed by tesselating space with the near-minimal computational box.

5.2.3 Constructing a near-minimal-volume computational box

As introduced in the previous section, the molecule M dilated by a layer of a certain width r is called the parallel body $\mathcal{P}(M,r)$ of M. The distance r of the parallel body must at least be equal to $\frac{1}{2}r_{co}$, but may be chosen a bit larger to allow for conformational changes.

Now, a near-minimal-volume computational box B^* can be constructed by calculating a near-densest packing lattice Λ^* for $\mathcal{P}(M,r)$. The lattice vectors l_1, l_2, l_3 of Λ^* span a parallelepiped that can be used as the near-minimal-volume triclinic box B^* (see fig. 5.5).

To set up a near-minimal M.D. simulation, molecule M has to be put in the minimal box B^*. The location of M in B^* is completely free, but the obvious choice is to locate M in the middle of B^*. Sometimes M will not fit entirely in B^*; it sticks out no matter where it is located in B^*. That does not matter, we simply locate M somewhere in the middle of B^*. Now, for every atom of M protruding B^*, it holds that it can be shifted over some lattice vector of Λ^* such that it falls in B^*. For every protruding atom such a vector is calculated and the atom is translated over this vector. Now all atoms of M are in B^*, but M is possibly fragmented. That is no problem, since we use periodic boundary conditions; when space is tesselated with B^*, complete molecules are formed (see fig. 5.6).

Molecules M in the lattice packing of $\mathcal{P}(M,r)$ have a minimal distance of $2r$ to each other. If $r > \frac{1}{2}r_{co}$, this means that atoms of different periodic copies of M do not interact with each other during a simulation in B^*.

Summarizing: to construct a near-minimal-volume computational box B^* for a molecule M, we must compute the near-densest packing lattice for the parallel body $\mathcal{P}(M,r)$ of M, where r is at least equal to $\frac{1}{2}r_{co}$. This means that we need a representation for $\mathcal{P}(M,r)$. In the next section, we discuss how this parallel body is constructed.

5.3 Parallel bodies

A parallel body $\mathcal{P}(M,r)$ of a set M at distance r is defined precisely as the (infinite) set containing every point whose distance to the object is less than or equal to r:

Definition 5.3.1 The *parallel body* $\mathcal{P}(M,r)$ of a set $M \subset \mathbb{R}^d$ at distance $r \in \mathbb{R}$ is defined as the set $\mathcal{P}(M,r) = \{p \in \mathbb{R}^d \mid \text{dist}(p,M) \leq r\}$, where $\text{dist}(x,y)$ is the shortest distance between x and y.

The parallel body $\mathcal{P}(M,r)$ can equivalently be defined as the Minkowski sum of M and a sphere with radius r. A molecule M can be regarded as a finite set of points, where each point corresponds with the centre of an atom of the molecule. So, the parallel body of M at distance r is the union of spheres with radius r placed at the points in M (see fig. 5.7).

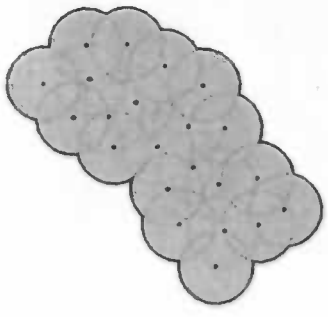

Figure 5.7: The parallel body of a point set is defined as the union of spheres placed at the points.

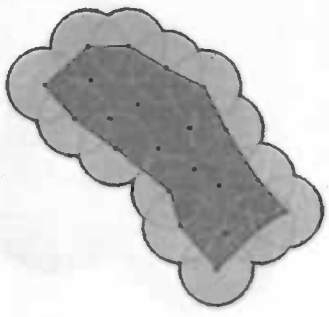

Figure 5.8: Only the spheres of boundary vertices of the alpha shape of the point set contribute to the parallel body.

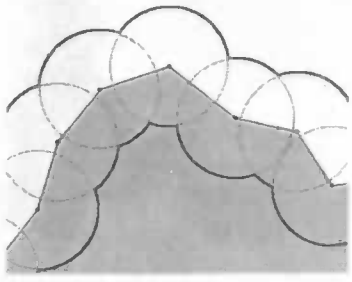

Figure 5.9: Two surfaces on either side of the boundary of the alpha shape (grey).

5.3.1 Constructing a parallel body point set

For the application discussed above, we have to compute the near-densest packing lattice for the parallel body $\mathcal{P}(M,r)$. Before this can be done, the difference body of $\mathcal{P}(M,r)$ has to be constructed. The algorithm for constructing difference bodies, discussed in chapter 4 (algorithm 4.2.1), requires a point set approximating the boundary of the object as input. This means that we have to construct a point set approximation of $\partial\mathcal{P}(M,r)$.

A natural solution to construct such an approximation is to replace every point in M by a sphere of radius r formed by points, evenly distributed over the surface of the sphere. However, this yields in general many points in the interior of the parallel body. Therefore, we use the following observation:

Observation 5.3.2 The sphere with radius r of a point in M contributes to the boundary of $\mathcal{P}(M,r)$ *if and only if* the point is a boundary vertex of the alpha shape with $\alpha = r$ of M.

This observation is proved in [11].

So our method starts with constructing an alpha shape of M with $\alpha = r$ (see fig. 5.8). Then, we construct an initial point set approximating the boundary of the parallel body by replacing each boundary vertex of the alpha shape with an evenly distributed set of points on a sphere with radius r. Many points of this initial point set fall inside the sphere of a neighbouring vertex, i.e. the distance between the point and the neighbour vertex is smaller than r. These points are discarded from the point set. Two vertices are considered neighbours when there is an edge between the vertices in boundary of the alpha shape.

The remaining points form two surfaces, one inside the alpha shape, the other outside the alpha shape (see figure 5.9). We are of course only interested in the points that form the outer boundary of the parallel body, so from the point set, we discard the points that lie in the interior of the alpha shape. For this purpose, we use the triangulation hierarchy data-structure that allows for efficient point location with respect to the alpha shape.

The remaining point set forms a very nice point approximation of the boundary of the parallel body. In figures 5.10-5.12 the result is shown for molecule 1A32 from the Protein Data Bank [18]. A similar method for constructing a parallel body point set is described in [12].

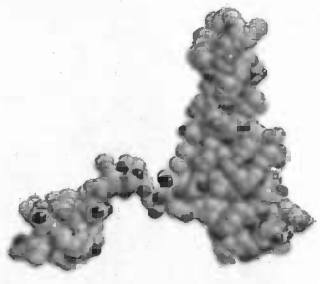

Figure 5.10: The molecule 1A32 from the Protein Data Bank (717 atoms).

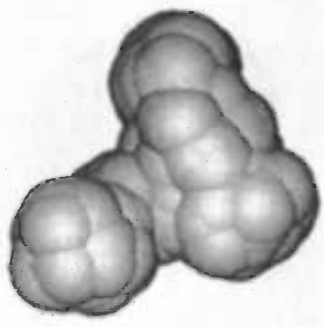

Figure 5.11: The parallel body of 1A32 at a distance of 10 Ångströms.

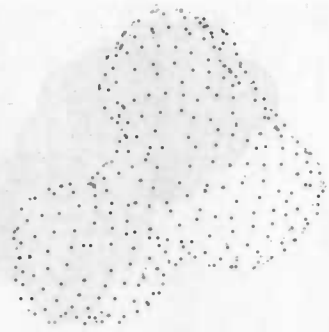

Figure 5.12: A point set approximating the boundary of the parallel body (638 points).

The quality of the distribution of points over the boundary of the parallel body, in terms of homogeneity and density, depends solely on the distribution of the points on the sphere. So, a good point distribution on the sphere is essential for constructing a nice parallel body point set.

5.3.2 Point distribution on the sphere

The spherical point distribution to be used for constructing a parallel body point set has to be as even as possible. The problem can be reduced to finding an even distribution on a spherical triangle (one octant of the sphere). By symmetry, the other parts of the sphere can then be filled with points. A natural candidate for distributing points on a spherical triangle is a projection onto the sphere of a triangular grid lying on a planar triangular face (see figure 5.13). The total number of points on the sphere would then be $4n^2 - 8n + 6$, where n is the number of points on an edge of the planar triangle. n could also be referred to as the point density on the sphere. The quality of this distribution is poor, however.

Therefore, we will construct a triangular grid directly on the sphere. In [12] such a grid is constructed by connecting the equiangular points on the edges of the spherical triangle by arcs of great circles. These arcs form a grid similar to the triangular grid previously mentioned. However, these arcs, and their subsequent subdivisions into spherical triangular grid points, do not coincide generally. This problem is solved by taking the average of the three resulting equivalent grid points.

In our implementation, we use the same method to construct a point distribution as described above, but instead of using arcs of great circles, we use arcs of circles perpendicular to the coordinate axes (see figure 5.14). This yields a nice point distribution on the sphere, at least good enough for our application (see figure 5.15). A point density of 5, i.e. $n = 5$, already proves to give good results for constructing a parallel body point set. The total number of points on the sphere is in this case equal to 66.

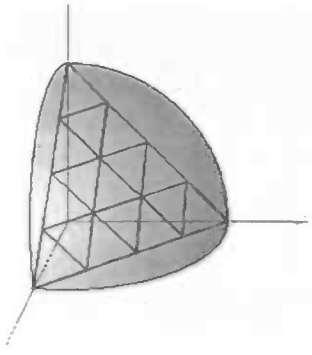

Figure 5.13: A planar triangular grid and a spherical triangle.

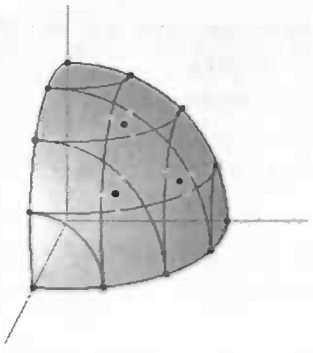

Figure 5.14: A spherical triangular grid. The equivalent grid points of each arc (white dots) do not coincide. Therefore, the average is taken.

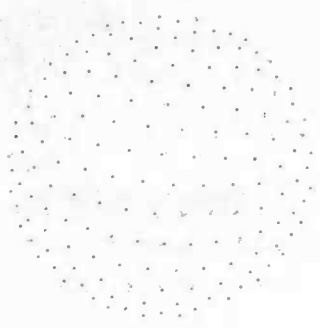

Figure 5.15: A point distribution on the sphere ($n = 10$).

5.3.3 Van der Waals radii

In our introduction to M.D. simulations, we assumed that whether or not two atoms interact with each other does not depend on the Van der Waals radii of the individual atoms, but solely on the distance between their centres. For some applications, it might be useful to take the Van der Waals radii into account as well.

The method to construct a parallel body point set must then be adapted. This can simply be done using weighted alpha shapes [1, 11] instead of basic alpha shapes. In weighted alpha shapes, every point in the input point set has an additional attribute, called its weight. For our application this might be the Van der Waals radius of the corresponding atom.

The weights might also be used to artificially vary the local distance of the 'parallel' body. We could choose for example a larger weight at parts of the molecule where relatively many conformational changes are expected. This allows for more local conformational changes during an M.D. simulation, without dramatically affecting the volume of the computational box.

In our current implementation, neither of these features is included, but the method we use allows for easy adaptation.

5.3.4 Algorithm outline

The algorithm outline for constructing a point set approximation $\partial\mathcal{P}(M, r)$ of the boundary of the parallel body of a molecule M at distance r is as follows. The input of the algorithm consists of a finite set of points, where each point corresponds with the centre of an atom of the molecule M, and the desired distance r of the parallel body.

Algorithm 5.3.3 PARALLELBODY(M, r)
$M^* \leftarrow$ ALPHASHAPE$(M, \alpha \leftarrow r)$
$S \leftarrow$ SPHEREPOINTS(density $\leftarrow 5$, radius $\leftarrow r$)
for all vertices $v \in M^*$ **do**
  for all points $p \in S_v$ **do**
    if p lies outside M^* **then**
      Initialize p as being a boundary point
      for all neighbouring vertices $n \in M^*$ of v **do**
        if dist$(p, n) < r$ **then**
          p is not a boundary point
      if p is a boundary point **then**
        Add p to $\partial\mathcal{P}(M, r)$
return $\partial\mathcal{P}(M, r)$

Chapter 6

The complete algorithm and Results

Until now, we have been considering three isolated algorithms: an algorithm for calculating near-densest packing lattices, an algorithm for constructing polyhedral difference bodies and an algorithm for computing parallel bodies. Obviously, the goal of these algorithms is that they are integrated into one program that computes near-minimal computational boxes for given molecules. In this chapter, we devise such a program.

The results we achieved with this program were beyond our own expectations. On average, we can reduce the CPU time of an M.D. simulation with more than 50%. For oblong molecules, this ratio can even be 80%.

6.1 Putting it all together

In this section we integrate the algorithms discussed in previous sections into one program that takes a list of atom coordinates as input, and returns the box vectors of a near-minimal computational box for the corresponding molecule as output. Such a program consists of the algorithms discussed in this paper, carried out in the reverse order of appearance:

1. In chapter 5, we devised an algorithm for computing the parallel body at a given distance of a molecule. This distance should at least be $\frac{1}{2}r_{co}$. The list of coordinates describing a molecule is provided as input to the algorithm. The output consists of a point set approximating the boundary of the parallel body.

2. To compute a packing lattice for this parallel body, we first have to construct its difference body. This can be done by algorithm 4.2.1. It takes the point set approximating the boundary of the parallel body as input and returns a polyhedral difference body.

3. Using the polyhedral difference body constructed in the previous step, we can compute a near-densest packing lattice for the parallel body. For this purpose, we devised two algorithms: an algorithm to compute 'type 1'-near-densest packing lattices, and an algorithm to compute 'type 2'-packing lattices. In principle, we should search for near-densest packing lattices of both types, but as we argued in chapter 3, we only use the algorithm for computing 'type 1'-near-densest packing lattices. This algorithm takes the polyhedral difference body as input and returns three lattice vectors that span the

near-densest packing lattice for the parallel body. In the corresponding lattice packing, the molecules have a minimal distance to each other of twice the layer width of the parallel body.

4. Finally, the desired triclinic computational box of near-minimal volume is constructed from the lattice vectors of the calculated near-densest packing lattice. This box can be used for simulating the molecule.

In pseudo-code, the overall outline of the program for constructing a near-minimal volume-computational box looks as follows. The input of the program is a set of points describing a molecule M, where each point corresponds with the centre of an atom of the molecule. Of course, molecule data is in general not available as a simple list of points, but is stored in various formats containing auxiliary information as well. Mostly, the Protein Data Bank (PDB) format is used for storing molecule data [18]. Because the coordinates of each atom of the molecule are listed in a PDB-file, a conversion to a list of points is straightforward.

Also, a number of parameters has to be set, namely the number of grid cells, the desired parallel body distance r and the density of points approximating the boundary of the parallel body. The output of the program is a near-minimal triclinic box B^* for molecule M.

Algorithm 6.1.1 NEARMINIMALCOMPUTATIONALBOX(M)
$\partial\mathcal{P}(M,r) \leftarrow$ PARALLELBODY(M,r) (see algorithm 5.3.3)
$\mathcal{D}(K) \leftarrow$ DIFFERENCEBODY($\partial\mathcal{P}(M,r)$) (see algorithm 4.2.1)
$\Lambda^* \leftarrow$ NEARDENSESTPACKINGLATTICETYPE1($\mathcal{D}(K)$) (see algorithm 3.4.1)
Construct a triclinic box B^* from the lattice vectors of Λ^*
return B^*

As a first test, we applied the program on molecule 1A32 from the Protein Data Bank (see fig. 6.1), which has a rather complicated shape, and the results were promising. In figure 6.2, we see four copies of a parallel body of 1A32 packed using our near-densest packing lattice method. The computational box that can be deduced from the packing lattice vectors proves to contain 4.86 times less volume than boxes that are constructed in the conventional way. *This means that the near-minimal box contains 79.4% less volume than the conventional box for 1A32.*

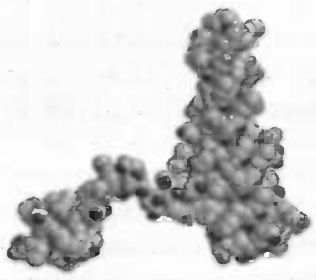

Figure 6.1: The molecule 1A32 from the Protein Data Bank.

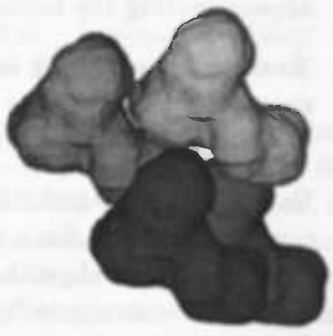

Figure 6.2: The near-densest lattice packing of the parallel body of 1A32.

The question is whether the gain in computational box volume results in proportional speedups of the M.D. simulations in terms of required CPU time. The answer is yes! A simulation carried out in both the near-minimal box and the conventional box, shows that *79.3%* of the simulation time is saved.

Of course, the example mentioned above is only a coarse test for one molecule. In the next section we test our program for many more molecules, and the results are discussed in more detail. Also, we describe how a test simulation is actually set up.

6.2 Results in M.D. simulations

For a significant number of macro-molecules of various sizes and shapes, we calculated a near-minimal triclinic box using the program presented above. Subsequently, the molecule was simulated in this near-minimal triclinic box by the molecular dynamics simulation package GROMACS [15].

For comparison, we also simulated the molecules in a conventional box. In GROMACS, usually a rhombic dodecahedron surrounding the molecule to be simulated is used. Such a box is more complex than a triclinic box, and therefore it suffers more computational overhead during a simulation. In order to obtain a fair comparison, each rhombic dodecahedron was transformed into an equivalent triclinic box (see section 5.2.2 of the previous chapter), before the simulations were carried out.

Macro-molecules			Rh. dodecahedron		Near-minimal triclinic box		
PDB code	number of atoms	pre. time [hr:min]	volume [nm^3]	sim. time [hr:min]	volume [nm^3]	sim. time [hr:min]	speedup
1A32	1102	0:11	577.82	06:51	118.93	01:25	**79.3%**
1A6S	805	0:15	142.43	01:45	80.08	00:58	**44.8%**
1ADR	763	0:11	167.25	01:59	80.73	00:55	**53.7%**
1AKI	1321	0:14	233.54	02:48	93.99	01:07	**60.0%**
1BW6	595	0:15	130.27	01:29	66.32	00:45	**49.2%**
1HNR	485	0:15	124.31	01:29	59.30	00:41	**53.9%**
1HP8	686	0:12	177.10	02:09	77.57	00:53	**58.8%**
1HQI	982	0:12	218.77	02:38	103.71	01:12	**54.3%**
1NER	768	0:16	147.91	01:45	85.35	00:58	**44.8%**
1OLG	1808	0:17	468.93	05:37	203.44	02:25	**56.9%**
1PRH	11676	0:18	1337.80	16:27	611.67	07:47	**52.6%**
1STU	668	0:08	190.32	02:16	73.41	00:50	**63.2%**
1VCC	833	0:16	152.69	01:48	69.77	00:49	**54.5%**
1VII	389	0:14	99.74	01:08	46.96	00:32	**52.8%**
2BBY	767	0:15	159.26	01:53	80.78	00:56	**50.2%**
1D0G	1052	0:05	645.92	07:42	112.78	01:19	**82.9%**
1D4V	3192	0:34	1319.21	15:51	451.23	05:29	**65.4%**

Table 6.1: Seventeen macro-molecules from PDB simulated with GROMACS in both a rhombic dodecahedron and a near-minimal triclinic box. For every molecule the box volume and simulation time is given for both boxes. In the last column, the gain in simulation time is given. The third column gives the running time used to calculate a near-minimal box.

Every molecule was simulated for 25000 time-steps of 2 fs in both boxes, using rotational restraining. The simulations were done on a single AMD Athlon 600 MHz. For the rhombic dodecahedron, the 'wall distance' (i.e. the minimal distance between the molecule and the box) was chosen at 10 Å. Similarly, in the near-densest packing lattice method the width of the layer around each molecule was 10 Å, i.e. the parallel body is constructed at a distance of 10 Å. For all simulations in both boxes, the cut-off radius was 14 Å.

In table 6.1 the simulation times in both boxes are given for seventeen macro-molecules. From this table, it is clear that on average *boxes calculated with the near-densest packing lattice method have a volume that is approximately 60% smaller than the volume of the corresponding GROMACS rhombic dodecahedron, and that the speedup of the simulation is about 55%.*

The simulation results for both boxes, in terms of conservation of energy, temperature, pressure, free energy and other statistical parameters, proved to be the same within noise limits. Because chaotic systems are simulated, there are of course, local differences in particle trajectories between the simulations carried out in both boxes. Globally however, we see in both boxes the same conformational changes during the simulation, which means that using a near-minimal computational box in combination with rotational restraining does not affect the overall results of a simulation.

The table above shows that the speedup gained by using a near-minimal triclinic box ranges from 44.8% (1A6S, 1NER) to even 82.9% (1D0G). The large difference between these numbers is explained by the shape of these molecules. Molecules 1A6S and 1NER are fairly spherical (see fig. 6.3), so that a rhombic dodecahedron can be constructed fairly tightly around the molecules. However, a rhombic dodecahedron can not be constructed efficiently around oblong molecules, such as 1D0G (see fig. 6.4), so for these kind of molecules the near-densest packing lattice method yields a huge speedup of M.D. simulations.

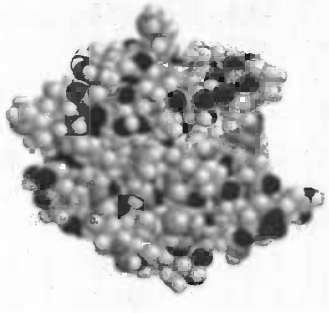

Figure 6.3: The fairly spherical molecule 1A6S from the Protein Data Bank.

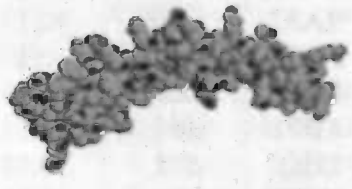

Figure 6.4: The rather oblong molecule 1D0G from the Protein Data Bank.

The simulations done for the purpose of testing our near-minimal volume boxes only took a few hours on average, and already the gain in simulation time outweighs the costs of calculating a near-minimal box (see table 6.1). For simulations that take longer, it might be possible that at some point in the simulation the computational box will no longer be suitable anymore, due to conformational changes of the molecule. This can be solved by simply calculating a new computational box for the changed molecule. The simulation can then be continued in the newly calculated box. Since it takes only a short period of time to compute a near-minimal box, the costs of such an operation are negligible. In the next section, we analyze the complexity and running time of our program in more detail.

6.3 Complexity analysis and running time

We can not simply give a formula for the running time complexity of our method; the running time complexity differs per step of the program. Therefore, we evaluate the running time for each step of the program. The experiments were carried out on a PIII-500 with 256 MB of memory.

6.3.1 Parallel body calculation

The first step of the program concerns computing a point set approximation of the molecule's parallel body. To this end, the alpha shape is computed of the input set of of points (i.e. the coordinates of the atoms of the molecule). If the molecule has n atoms, it takes $O(n^2)$ time in the worst case to compute the alpha shape.

The number of output points on the approximated parallel body depends on the density of the distribution of points on the unit sphere. If we choose this density equal to 5 (66 points on the sphere), the number of points in the output is in general smaller than n, depending on the shape of the molecule.

In table 6.2, we show for some molecules of various sizes the running time of the first step of the overall algorithm. It can be seen that for small molecules this step does not take a significant amount of running time. For large ones, on the contrary, it can be a major bottleneck.

PDB code	input points (nr. of atoms)	running time [hr:min:sec]	output points
1VII	596	0:00:03	346
1A32	717	0:00:05	638
1D4V	2531	0:00:30	2080
1PRH	9006	0:04:24	1939
1N2C	24190	0:30:37	3602
1AON	58688	3:00:18	8497

Table 6.2: Table showing the running time for calculating a parallel body point set of six molecules of various sizes. The number of atoms may differ from table 6.1, due to preprocessing of PDB-files by GROMACS. The density of points on the unit sphere is 5 (66 points).

6.3.2 Difference body calculation

In the second step of the program, the difference body of the molecule's parallel body is calculated. This calculation falls apart in two different stages. First, we have to calculate the dense cloud of points approximating the difference body, which is filtered using the grid method. If the number of points in the input (i.e. the number of points on the parallel body) is denoted by n, the dense cloud of points contains exactly n^2 points. So this stage requires $O(n^2)$ running time in all cases. Because only addition operations are used in this stage, the running time is, despite the quadratic complexity, not significant in practice.

The second stage concerns computing an alpha shape of the filtered point cloud. As said before, this takes quadratic running time in the number of points in the input point set, in this case the filtered point cloud. The number of points in this filtered point set depends on

the number of grid cells chosen and to a smaller degree on the shape of the difference body. In our experiments, the number of grid cells is chosen at $32 \times 32 \times 32$.

In table 6.3, the running times of the second step of the program are shown.

PDB code	input points	running time first stage [hr:min:sec]	filtered points	running time second stage [hr:min:sec]	surface vertices	total running time [hr:min:sec]
1VII	346	0:00:00	14230	0:10:05	2088	0:10:05
1A32	638	0:00:00	10520	0:05:30	2400	0:05:30
1D4V	2080	0:00:04	18906	0:17:29	3990	0:17:33
1PRH	1939	0:00:04	11056	0:06:07	2912	0:06:11
1N2C	3602	0:00:14	5972	0:01:58	2070	0:02:12
1AON	8497	0:01:16	12598	0:08:19	3310	0:09:35

Table 6.3: Table showing the running time for calculating a polyhedral difference body. The third column shows the running time for constructing a filtered point cloud approximating the difference body. The fifth column shows the running time of the alpha shape calculation of the difference body. The number of grid cells used is $32 \times 32 \times 32$.

6.3.3 Near-densest packing lattice calculation

In the last step of the program, we search for a near-densest packing lattice. Each of the vertices of the polyhedral difference body is inspected as a possible position for the first lattice vector. For each possible position of the first lattice vector, we have to search for a possible position for the second lattice vector. Since the number of possible positions for the third lattice vector, given a position for the first and second, is not significant in general (in most cases 2), the complexity of this step is roughly quadratic in the number of vertices.

The number of vertices depends mostly on the number of points in the filtered point set and to a less degree on the shape of the difference body. In table 6.4, the running times for finding a near-densest packing lattice are shown.

PDB code	surface vertices	running time [hr:min:sec]
1VII	2088	0:04:06
1A32	2400	0:05:41
1D4V	3990	0:16:25
1PRH	2912	0:07:16
1N2C	2070	0:05:12
1AON	3310	0:08:43

Table 6.4: Table showing the running time for searching for a near-densest packing lattice. From the running times for 1VII and 1D4V, it can be seen that the complexity is roughly quadratic in the number of vertices of the polyhedral difference body.

6.3.4 Summary

The total running times for each of these molecules is simply the sum of the running times of the individual steps. In table 6.5, an overview is given of the running times of each step of the program. It can be seen that the running time of the last two steps is more or less constant, on average taking a total time of approximately 15 minutes. The first step of the program really depends on the size of the input, so we may state that the overall complexity of our program is $O(n^2)$, where n is the number of atoms of the input molecule. For small n this step is not significant in running time, for large n, on the contrary, this step can be a major bottleneck.

The table shows that our method is perfectly capable of calculating a near-minimal box for molecules of approximately 60000 atoms. In current practice, very rarely molecules of this size are simulated in a solvent, so we may conclude that our method works for any realistic macro-molecule.

The running time of our program is on average approximately 15 to 30 minutes for molecules used nowadays, as can be seen from the table. Since M.D. simulations can take months in practice, the costs of computing a near-minimal computational box are really negligible when compared to the expected gain in simulation time.

PDB code	number of atoms	running time parallel body [hr:min:sec]	running time differ. body [hr:min:sec]	running time near-DLP [hr:min:sec]	total running time [hr:min:sec]
1VII	596	0:00:03	0:10:05	0:04:06	0:14:14
1A32	717	0:00:05	0:05:30	0:05:41	0:11:16
1D4V	2531	0:00:30	0:17:33	0:16:25	0:34:28
1PRH	9006	0:04:24	0:06:11	0:07:16	0:17:51
1N2C	24190	0:30:37	0:02:12	0:05:12	0:38:01
1AON	58688	3:00:18	0:09:35	0:08:43	3:18:36

Table 6.5: An overview of the running times of each step of the algorithm.

Chapter 7

The final chapter

7.1 Discussion

So far, we showed that our near-densest packing lattice method can be very useful, especially in the field of M.D. simulations. However, we would like to compare our method with other packing algorithms, to see how good it performs. We did not encounter other algorithms approximating densest packing lattices of non-convex objects in literature, so to do some comparison, the only candidate is the algorithm of Betke and Henk [9], which computes densest packing lattices exactly. This algorithm, however, only takes convex polyhedra as input, whereas our method has no such constraint. Also, the running time of this algorithm is $O(n^7)$, where n is the number of vertices, which means that is not feasible to compute the densest packing lattice for any but the simplest polyhedra. For these reasons, a comparison between the two methods seemed not very meaningful to us.

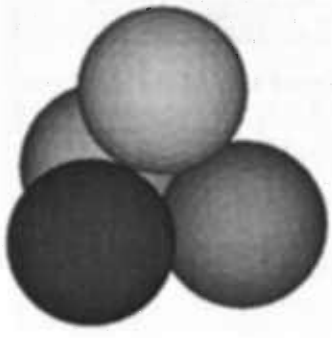

Figure 7.1: A close approximation of the densest lattice packing of a sphere.

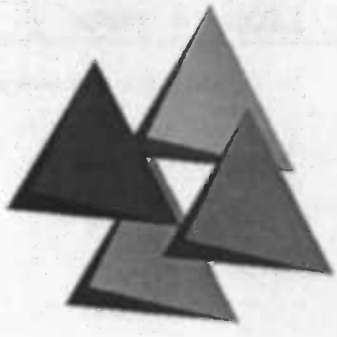

Figure 7.2: A close approximation of the densest lattice packing of a regular tetrahedron.

We can on the other hand, test our method on shapes of which the densest packing lattice is known, for instance the sphere or the regular tetrahedron. It turns out that our method very well approximates the actual densest packing lattice of a sphere (see fig 7.1). It is well known that the lattice packing density of the sphere is $\frac{\pi}{3\sqrt{2}} \approx 0.7405$ [19]. The packing density of a sphere calculated with our method is 0.7447. This number is a bit larger than the actual density, because we approximated the sphere by an inscribed polygon of 326 vertices. It shows nevertheless, that our algorithm works correctly in this case.

Also for the regular tetrahedron, our near-densest packing lattice method performs very well (see fig. 7.2). The actual lattice packing density of the regular tetrahedron is $\frac{18}{49} \approx 0.3673$ [9]. The density of the packing calculated with our method is 0.3672, which is a very close approximation of the actual density.

7.2 Conclusion

In this paper, we presented a conceptually clear method with a sound basis to compute near-minimal computational boxes that can be used for simulations where a molecule is simulated in a solvent. The presented method proved to be stable and robust; we showed that the method works for any realistic macro-molecule, even for those consisting of more than 60000 atoms. Furthermore, we proved that the running time of the method is negligible relative to the average duration of an M.D. simulation.

The results we achieved with this method in M.D. simulations were beyond our own expectations. On average, a speedup of about 50% can be realized when compared to conventional methods used nowadays, and for oblong molecules even 80% or more simulation time can be gained. We must note that a near-minimal computational box is only useful when it is used in combination with rotational restraining.

We only compared our method to compute a near-minimal computational box with the conventional methods of GROMACS, not with other M.D. simulation packages. However, because other packages use the same methods as GROMACS to calculate the computational box we expect that for other packages a similar gain can be achieved.

On average, it takes about half an hour to compute a near-minimal computational box. Since M.D. simulations can take months in practice, the costs of computing a near-minimal computational box are negligible.

The method to compute a near-densest packing lattice for an arbitrary object can at itself be very useful as well. Although we did not solve the densest packing lattice problem for non-convex objects exactly, we are convinced that the presented method is applicable at many places.

Numerous refinements and improvements can be made to our algorithm, for instance one could question the way we search through the configuration space and how a polyhedral difference body is constructed. Yet, we have the feeling that no significant improvements in lattice packing density can be achieved. We showed anyhow, that the concept of our method works very well.

The main conclusion of this paper remains that *using our near-densest packing lattice method, a speedup of 50% to even 80% can be achieved in Molecular Dynamics simulations.*

7.3 Acknowledgements

The author is very grateful to the following persons:

- Henk Bekker, for supervising me during this project.
- Tsjerk Wassenaar, for carrying out the simulations done on macromolecules using the minimal computational boxes introduced in this paper.
- Nico Kruithof, for the introduction he gave me in CGAL, the computational geometry software library that has been used to implement the presented algorithm.

- Arend Smit, for his help concerning OpenGL to create visualizations of packings of molecules shown in this paper and other implementation issues.
- Axel Brink, for his helpful comments on this paper and his moral support during the project.
- Meinte Boersma and Harm Bakker, for their help using LaTeX, in which this paper is set.

Bibliography

[1] *The CGAL user manual.* http://www.cgal.org, Version 2.4 (2002).

[2] P.K. Agarwal, E. Flato and D. Halperin, *Polygon decomposition for efficient construction of Minkowski sums.* Computational Geometry: Theory and Applications, Special Issue, selected papers from the European Workshop on Computational Geometry, Eilat, 2000, vol. 21, pp. 39-61 (2002).

[3] A. Amadei, G. Chillemi, M.A. Ceruso, A. Grottesi and A. Di Nola, *Molecular dynamics simulations with constrained roto-translational motions: Theoretical basis and statistical mechanical consistency.* Journal of Chemical Physics Vol. 112, No. 1, pp. 9-23 (2000).

[4] N. Amenta, S. Choi and R.K. Kolluri, *The power crust.* Computational Geometry Vol. 19, No. 2-3, pp. 127-153 (2001).

[5] H. Bekker, *Molecular dynamics simulation methods revised.* PhD thesis, University of Groningen (1996).

[6] H. Bekker, *Unification of box shapes in molecular simulations.* Journal of Computational Chemistry Vol. 18, No. 15, pp. 1930-1942 (1997).

[7] H. Bekker and J.B.T.M. Roerdink, *An efficient algorithm to calculate the Minkowski sum of convex 3D polyhedra.* Proc. Computational Science - ICCS 2001, Lecture Notes in Computer Science 2074, V. N. Alexandrov, J. J. Dongarra, B. A. Juliano, R. S. Renner and C. J. Kenneth Tan (eds.), pp. 619-628 (Part I), Springer (2001).

[8] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf, *Computational Geometry, Algorithms and Applications*, Second edition. Springer Verlag Berlin, Heidelberg (2000).

[9] U. Betke and M. Henk, *Densest lattice packings of 3-polytopes.* Comput. Geom. 16, 3, pp. 157-186 (2000).

[10] D. Cohen-Steiner and F. Da, *A greedy Delaunay based surface reconstruction algorithm.* ECG-TR-124202-01.

[11] H. Edelsbrunner and E.P. Mücke, *Three-dimensional alpha shapes.* ACM Transactions on Graphics Vol. 13, No. 1, pp. 43-72 (1994).

[12] F. Eisenhaber, *The double cubic lattice method: efficient approaches to numerical integration of surface area and volume and dot surface contouring of molecular assemblies.* Journal of Computational Chemistry Vol. 16, No. 3, 273-284 (1995).

[13] P.K. Ghosh, *A unified computational framework for Minkowski operations.* Comput.&Graphics Vol. 17, No. 4, pp. 357-378 (1993).

[14] S. Gottschalk, M.C. Lin and D. Manocha, *OBBTree: A hierarchical structure for rapid interference detection.* Computer Graphics Vol. 30, Annual Conference Series, pp. 171-180 (1996).

[15] *GROMACS: The GROningen MAchine for Chemical Simulations.* http://www.gromacs.org.

[16] S.R. Lay, *Convex sets and their applications.* John Wiley & Sons, New York (1982).

[17] N. Nikopoulos and I. Pitas, *An efficient algorithm for 3d binary morphological tranformations with 3D structuring elements of arbitrary size and shape.* Proceedings of IEEE Workshop on Nonlinear Signal and Image Processing (NSIP'97), Michigan, (1997).

[18] *PDB: The Protein Data Bank.* http://www.rcsb.org/pdb/.

[19] E. Weisstein, *Eric Weisstein's world of mathematics.* http://mathworld.wolfram.com.