3 Unification of Box Shapes in Molecular Simulations

In molecular simulations with periodic boundary conditions the computational box used, may have five different shapes: triclinic, the hexagonal prism, two types of dodecahedrons, and the truncated octahedron. In this chapter we show that every molecular simulation, irrespective of the shape of the initial computational box, can be done as a simulation in one of the other ones, i.e. we show that in a preprocessing phase a simulation formulated in one particular box can be transformed into a simulation in another box such that the simulation in the new box is exactly *identical to the simulation in the original one. This means that every molecular simulation may be done in the same type of box. Because the triclinic box is the easiest one to implement, we pay special attention on how to transform the other four box types into triclinic boxes. As a consequence, simulations in the often used truncated octahedron are superfluous; they may be done in a much simpler way in a triclinic box.*

3.1 Introduction

To mitigate finite system effects most molecular simulations are done on systems with periodic boundary conditions (P.B.C.). This means that the computational box is surrounded in a space-filling way by replica boxes, with identical content. In terms of the crystallographic Bravais lattices we consider only triclinic systems, i.e systems without symmetry elements.

In [1] it is proven that in 3-D space there are five convex[1] box types (see Figure 3.1)

(1) This property is not strictly necessary, but image calculations would become very complex for a non-convex box.

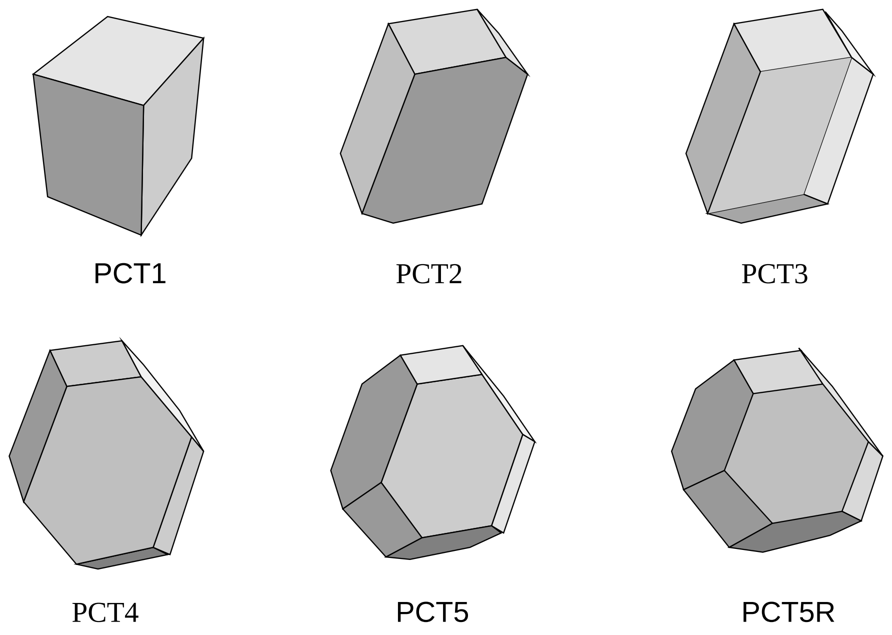

FIGURE 3.1 Instances of the triclinic box, the hexagonal prism, two types of dodecahedrons, the truncated octahedron, and the most regular instance of the truncated octahedron, in this chapter designated by PCT1, PCT2, PCT3, PCT4, PCT5, and PCT5R respectively.

that can be stacked in a space filling way, i.e. that there are five possible types of boxes which may serve as a computational box: the triclinic box, the hexagonal prism, two types of dodecahedrons, and the truncated octahedron. For short we will designate these box types by PCT1, PCT2, PCT3, PCT4, and PCT5, where PC stands for 'Primitive Cell', and T stands for 'Type'. The notion 'primitive cell' will be explained later. The rectangular instance of PCT1 will be designated by PCT1R and the most regular instance of PCT5 by PCT5R. In the M.D. world, PCT5R is often called 'the truncated octahedron', but as we will show later it is only the most regular instance of a broader class of boxes. In the early years of molecular simulation PCT1 was used. Later PCT3 was introduced [9], and then PCT5R [8].

In current implementations of molecular simulation algorithms, the shape of the computational box has to be taken into account at many places in the algorithm, notably in neighbour searching, in non-bonded force calculations, in bonded force calculations, and in the part in which particles are reset into the box. For the computational boxes PCT2, . . ., PCT5, which have a complex shape, calculating the position of image particles outside the box as a function of their position in the box, is complex. For this reason, in most molecular simulation packages only a limited set of box shapes has been implemented. E.g. in the molecular dynamics package GROMOS [2] only a limited instance of PCT1 and PCT5R have been implemented.

In this chapter we will show that every molecular simulation which is formulated in one of the boxes PCT1, ..., PCT5, can be transformed into a simulation in any one of the other boxes. So, a simulation, formulated in PCT2, ..., PCT5 can be transformed into a simulation in PCT1 or PCT1R. These transformations can be done in a preprocessing stage of a molecular simulation, so the actual simulation can take place in for example PCT1 or PCT1R, including neighbour searching, non-bonded force calculations, bonded force calculations, resetting particles into the box, pressure scaling, etc. The simulation in PCT1 and PCT1R is *exactly* identical to a simulation of the initial, untransformed system. So, for example, the number of particles and interactions to be evaluated is exactly the same in all cases.

The structure of this chapter is as follows. In Section 3.2 we define the shape of PCT1, ..., PCT5 in an algebraic way by a lattice and an alternative metric. Using this representation we derive the main theorem of this chapter. The lattice-and-metric way of defining PCT1, ..., PCT5 is not suitable for geometrical considerations. Therefore, in Section 3.3 we introduce a different, but equivalent representation of PCT1, ..., PCT5. Using this representation, we show that PCT1, ..., PCT4 are degenerate instances of PCT5, and we show show how a tiling of the space with PCT5 defines a lattice. In Section 3.4 we define a PCT1 and a PCT1R in terms of a given PCT5, such that PCT1 and PCT1R define the same lattice as PCT5. Because PCT1, ..., PCT4 are degenerate instances of PCT5, the same expressions may be used to define a PCT1 and a PCT1R in terms of PCT1, ..., PCT4.[2] In Section 3.5 we show how to map particles from one box into another one. As an example, in Section 3.6 we show how a simulation, formulated in PCT5R, is transformed into PCT1 and PCT1R.

The fact that every simulation, formulated in some box may be transformed into a simulation in an other box, clarifies a number of unresolved matters. Notably the pressure scaling of simulations in an PCT2, ..., PCT5 box, controlling the long range order of a molecular systems, and the maximum allowed cut-off radius. These

(2) Obviously, transforming PCT1 into PCT1 is an identity transformation. But because of the generic character of the algorithms we propose, we do not have to exclude PCT1 from our algorithms.

matters and more will be discussed in Section 3.7.

The methods presented in this chapter may be used to transform existing molecular simulations, formulated in PCT2, ... PCT5, into a simulation in for example PCT1 and PCT1R. That is, however, not the best way to set up a new simulation because then, complex boxshapes are still used to set up a simulation. In Subsection 3.7.4 it is shown how to set up a new simulation without using complex boxes.

We feel that the methods as presented in this chapter to do molecular simulations in a simple box, together with the efficient method presented in [3], will result in faster and simpler molecular simulation software with a wider range of features. All this, is brought about, not by improving existing implementations, but by revising the basic concepts of M.D. simulation.

3.2 Defining primitive cells by a lattice and a metric

In 3-D space, a lattice $\mathcal{L}$ is the set of points

$$\mathcal{L}(\boldsymbol{K},\boldsymbol{L},\boldsymbol{M}) \equiv n_1\boldsymbol{K} + n_2\boldsymbol{L} + n_3\boldsymbol{M}, \qquad \text{with } n_1, n_2, n_3 \in \mathbb{Z}\,, \tag{3.1}$$

where $\boldsymbol{K}$, $\boldsymbol{L}$, and $\boldsymbol{M}$ are three independent vectors, called the *basis* vectors. We define a *lattice vector* as a vector connecting two lattice points, so, because the origin is a lattice point, lattice vectors are also given by (3.1). Two points, 1 and 2, are called *corresponding points* when their positions are related by

$$\boldsymbol{r}_1 + \text{lattice vector} = \boldsymbol{r}_2\,. \tag{3.2}$$

In Euclidean space the squared distance $d^2(p_1,p_2)$ between two points p_1 and p_2 is given by

$$d^2(p_1,p_2) = (\boldsymbol{p}_1 - \boldsymbol{p}_2)^T\,\mathbf{I}\,(\boldsymbol{p}_1 - \boldsymbol{p}_2)\,. \tag{3.3}$$

where $\mathbf{I}$ is the unit matrix. The Euclidean distance function satisfies the general conditions of a distance function

$$d(p_1,p_2) > 0, \ \text{ if } p_1 \neq p_2; \qquad d(p_1,p_1) = 0; \tag{3.4}$$

$$d(p_1,p_2) = d(p_2,p_1); \qquad d(p_1,p_2) \leq d(p_1,p_3) + d(p_3,p_2), \ \forall p_3. \tag{3.5}$$

However, the Euclidean distance function is not the only one meeting these conditions. Every function defined as

$$d^2(p_1,p_2) = (\boldsymbol{p}_1 - \boldsymbol{p}_2)^T\mathbf{m}\,(\boldsymbol{p}_1 - \boldsymbol{p}_2)\,, \tag{3.6}$$

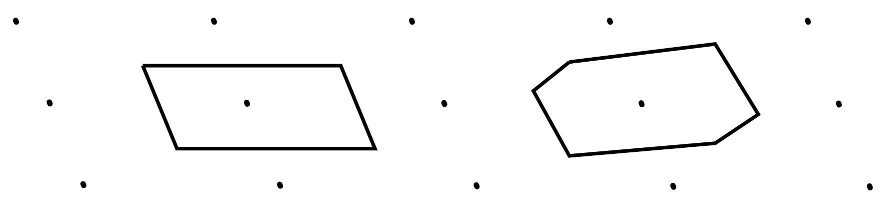

FIGURE 3.2 Two primitive cells defined by the same lattice but two different distance functions.

with **m** a positive definite matrix, satisfies (3.4), (3.5). (A matrix **m** is called positive definite when

$$\mathbf{m}^T = \mathbf{m} \text{ and } \boldsymbol{x}^T \mathbf{m}\, \boldsymbol{x} > 0, \qquad \text{for all } \boldsymbol{x} \neq 0\,. \tag{3.7}$$

Note that **m** is by definition symmetric.)

We can use a lattice and a distance function to partition the whole space in *primitive cells* in the following way. To every lattice point p belongs a primitive cell PC, defined so that every point in PC is closer to p than to any other lattice point. This is the well known Voronoi or Wigner-Seitz construction [10], using a more general metric. In Figure 3.2 two 2-D examples are given of a primitive cell defined by the same lattice but by different distance functions. It has been proven that by giving a lattice and a metric, the 3-D space is partitioned into five types of primitive cells [6]. These are the triclinic box, the hexagonal prism, two types of dodecahedrons, and the truncated octahedron, i.e. PCT1, . . ., PCT5. In this way every primitive cell is uniquely determined by a lattice and a distance function. I.e. every possible computational box can be described by giving a lattice and a metric, and the other way around.

In our definition of a lattice, the origin is a lattice point. We will designate a primitive cell in general by PC, and a primitive cell centred around the origin by PC0. The primitive cells PCT1, . . ., PCT5, PCT1R, and PCT5R, centred at the origin, are denoted as PC0T1, . . ., PC0T5, PC0T1R, and PC0T5R. In each primitive cell is a unique lattice point, which we will often call the *centre* of PC.

A primitive cell defined in this way is an open set of points, because, in our definition of a primitive cell, we do not consider points with the same distance to two lattice points. This would mean that a point with equal distance to two or more lattice points is not in any primitive cell at all. For molecular simulation this is undesirable;

every point in the infinite PBC system should belong to exactly one (image) box. Therefore it is necessary that we define a primitive cell as a half open, half closed set of points, so that tiling the space with primitive cells covers every point of space exactly once. How this is implemented is of no importance in later discussions. There we will simply take a primitive cell as a half open half closed set of points.

Using the lattice-and-metric way to define primitive cells, we will now derive some theorems about primitive cells, culminating in the main theorem of this chapter. We present these derivations in an informal style.

A molecular system with P.B.C. is in principle an infinite system because every (image) box is surrounded by replica boxes. With an infinite system we will mean the set of particles, not the boxes. This gives the following definition:

Definition 1 *Two infinite molecular systems IS and IS′ are called* identical *when for every particle in IS there is an identical particle in IS′ at the same position, and when for every particle in IS′ there is an identical particle in IS at the same position.*

Theorem 1 *A primitive cell does not contain two corresponding points.*

Proof: Assume that a primitive cell PC contains two corresponding points i and j. This means that i and j are closer to the centre of PC than to any other lattice point, while $\boldsymbol{r}_i - \boldsymbol{r}_j$ is a lattice vector. Primitive cells are centred around lattice points, so the relative position of primitive cells are lattice vectors. This means that shifting a point, belonging to some primitive cell, over a lattice vector will bring this point to another cell. However, shifting i over the lattice vector $\boldsymbol{r}_i - \boldsymbol{r}_j$ brings it to j, which is in the same cell. This contradiction implies that the first assumption is wrong.

Theorem 2 *A lattice $\mathcal{L}$ and a metric* **m** *define the cell* PC0*. Then, for every point in space, there is a corresponding point in* PC0*.*

Proof: A tiling of the space with PC covers every point of the space. So, every point p will fall in some PC. This PC is shifted over a lattice vector with respect to PC0. Shifting p over minus this lattice vector will bring this point into PC0. According to Theorem 1 this is the only translation over a lattice vector which brings p into PC0.

Consequently, when we have two primitive cells defined by the same lattice but different metrics, for every point in one cell there is a corresponding point in the other one, and the other way around. The central theorem of this chapter is:

Theorem 3 *A lattice $\mathcal{L}$ and a metric* $\mathbf{m}$ *define a primitive cell* PC0*.* PC0 *contains particles. Tiling the space with* PC0 *gives an infinite system IS. The same lattice $\mathcal{L}$ and another metric* $\mathbf{m}'$ *define* PC′0*. According to Theorem 2, the particles from* PC0 *are brought into* PC′0 *by shifts over lattice vectors. Tiling the space with* PC′0 *gives an infinite molecular system IS′. Then IS and IS′ are identical.*

Proof: The position of a particle in PC0 and its position in PC′0 only differ by a lattice vector (Theorem 2). IS is created by tiling the space with PC, that is, by locating a tile PC at every lattice point. IS′ is created by locating a tile PC′ at every lattice point. So, particles are only shifted over lattice vectors. Because tiling means shifting over all possible lattice vectors, every particle in IS coincides with a particle in IS′.

So far for theorems. We will now investigate how many parameters are required to specify the most general primitive cell, being PCT5. In 3-D space, a lattice is defined by giving three independent vectors, so, by giving nine numbers. A distance function is fully defined by giving a symmetric 3×3 matrix $\mathbf{m}$, so, by giving six numbers. However, we do not use the distance function to measure distances but only to compare distances. So, for our purpose it does not matter whether we use $\mathbf{m}$, or $\mathbf{m}$ multiplied with a uniform factor. In this way, the number of parameters required to define the distance function reduces from six to five. Therefore, we arrive at nine plus five is fourteen parameters to describe a primitive cell. Describing a primitive cell in this way means that its shape and its orientation in space is determined, but not its position.

A last remark, about the diameter of a PC. The diameter of a PC is the maximum of the distance between any pair of points belonging to PC. It can be shown that for any lattice $\mathcal{L}$, potentially the diameter of the primitive cell is unbounded.

Using a lattice and metric to define a primitive cell is conceptually elegant, but not very well suited for geometrical considerations. In the next section we will use another way to describe PCT1, . . ., PCT5, which makes it possible to think in a more geometrical way about these box types.

3.3 Defining boxes by their edges

It has been proved [4] that a primitive cell, as introduced in the previous sections, is centrally symmetric, and that it is bounded by pairs of parallel faces. A face is a centrally symmetric hexagon or parallelogram. The edges of a primitive cell consist

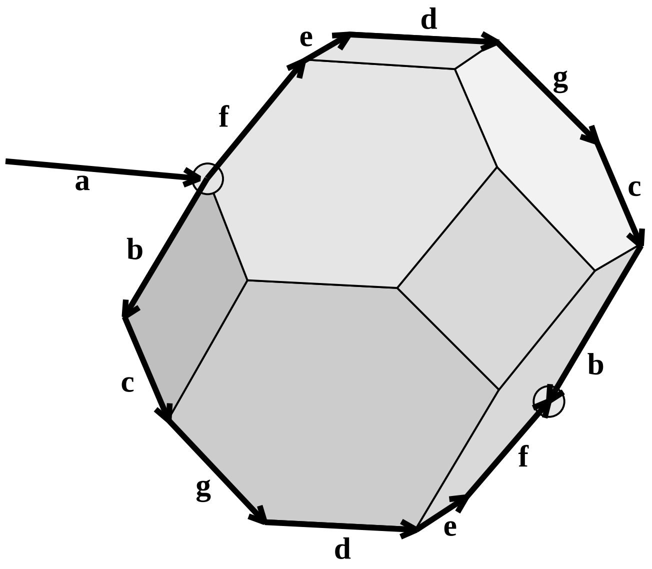

FIGURE 3.3 An instance of PCT5 defined by the six edges $\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{f}, \boldsymbol{g}$.

of groups of parallel lines. Using this last property, we come to our way of describing a primitive cell.

We describe PCT5 by giving its edge vectors $\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{f}, \boldsymbol{g}$ (see Figure 3.3). These six vectors completely define PCT5 because it consists of 36 edges, which can be grouped into six groups of six parallel edges each. When the vectors $\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{f}, \boldsymbol{g}$ were independent, PCT5 would have $6 \times 3 = 18$ degrees of freedom. However, the

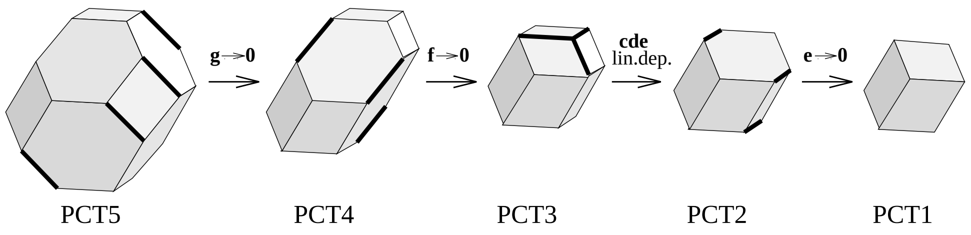

FIGURE 3.4 PCT5, PCT4 created by letting $\boldsymbol{g} \to \mathbf{0}$ of PCT5, PCT3 created by letting $\boldsymbol{f} \to \mathbf{0}$ of PCT4, PCT2 created by letting $|\boldsymbol{cde}| \to 0$ of PCT3, PCT1 created by letting $\boldsymbol{e} \to \mathbf{0}$ of PCT2. Fat lines go to zero.

vectors defining the hexagonal planes should be coplanar. This gives four constraint conditions: $|\boldsymbol{c}, \boldsymbol{e}, \boldsymbol{g}| = 0$, $|\boldsymbol{b}, \boldsymbol{d}, \boldsymbol{g}| = 0$, $|\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{f}| = 0$, $|\boldsymbol{b}, \boldsymbol{e}, \boldsymbol{f}| = 0$. So, PCT5 can be described by $18 - 4 = 14$ parameters, which corresponds with the number found in the previous section.

The number of degrees of freedom of PCT4, ..., PCT1 can be obtained by degenerating PCT5 as shown in Figure 3.4. To degenerate PCT5 into PCT4, only the *length* of the vector $\boldsymbol{g}$ should go to zero because the direction of $\boldsymbol{g}$ is not free. That is because $\boldsymbol{g}$ is the intersection of the planes defined by the vectors $\boldsymbol{c}, \boldsymbol{e}$ and $\boldsymbol{b}, \boldsymbol{d}$. So, PCT4 has one degree of freedom less than PCT5, i.e. $14 - 1 = 13$. In the same way PCT4 can be degenerated into PCT3 by letting $\boldsymbol{f} \to \mathbf{0}$. Again, because $\boldsymbol{f}$ is the intersection of two planes, defined by the vectors $\boldsymbol{c}, \boldsymbol{d}$ and $\boldsymbol{b}, \boldsymbol{e}$, only the *length* of $\boldsymbol{f}$ can be changed. So, PCT3 can be described by $13 - 1 = 12$ parameters. PCT2 can be obtained from PCT3 by choosing the vectors $\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}$ to be linearly dependent. This condition brings the number of degrees of freedom of PCT2 to $12 - 1 = 11$. PCT1 can be obtained from PCT2 by $\boldsymbol{e} \to \mathbf{0}$. The vector $\boldsymbol{e}$ is not completely free; it should be in the plane defined by the vectors $\boldsymbol{c}, \boldsymbol{d}$. So it has two degrees of freedom. This brings the number of degrees of freedom of PCT1 to $11 - 2 = 9$. This number of degrees of freedom is what may be expected expected from a triclinic box.

The whole process of going from PCT5 to PCT1, can be concisely written as

$$\mathsf{PCT5} \xrightarrow{\boldsymbol{g} \to \mathbf{0}} \mathsf{PCT4} \xrightarrow{\boldsymbol{f} \to \mathbf{0}} \mathsf{PCT3} \xrightarrow{|\boldsymbol{cde}| \to 0} \mathsf{PCT2} \xrightarrow{\boldsymbol{e} \to \mathbf{0}} \mathsf{PCT1}\,. \tag{3.8}$$

This shows that PCT1, ..., PCT4 are degenerate instances of PCT5, that PCT1, ..., PCT3 are degenerate instances of PCT4, etc. Put in an other way one can say that PCT5 is the generic space filler. Therefore, in the following paragraphs we only consider transformations of PCT5. Some properties of PCT5, ..., PCT1 are given in Table 3.1.

Again something about notation. Until now we only have been speaking about different box *types*. In this and the following sections different ways to *describe* boxes are introduced. We will denote a box described by the vectors $\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{f}, \boldsymbol{g}$ as PCDg. The g stands for ‘general’ because this is the most general way to describe a primitive cell. The box PCDg can be of any type because some vectors may be chosen zero or linearly dependent. Later, two other ways to describe boxes will be introduced. Analogous to our previous notation, the box PCDg centred at the origin is designated by PC0Dg.

A few words about the absolute position of boxes. The centre of symmetry of PCDg is half way the line connecting two opposite points of PCDg. The opposite

	PCT5	PCT4	PCT3	PCT2	PCT1
nr. faces	14	12	12	8	6
nr. rhombi	6	8	12	6	6
nr. hexagons	8	4	0	2	0
nr. edges	36	28	24	18	12
nr. vertices	24	18	14	12	8
degr. freedom	14	13	12	11	9

TABLE 3.1 Some properties of PCT5, . . ., PCT1.

vertices marked by a dot in Figure 3.3 are connected by the vector $-(\boldsymbol{b}+\boldsymbol{c}+\boldsymbol{d}+\boldsymbol{e}+\boldsymbol{f}+\boldsymbol{g})$. We will use the vector $\boldsymbol{a}$ to give the position of PCDg. Centring PCDg around the origin means that $\boldsymbol{a}$ should have the value

$$\boldsymbol{a} = -\frac{1}{2}(\boldsymbol{b}+\boldsymbol{c}+\boldsymbol{d}+\boldsymbol{e}+\boldsymbol{f}+\boldsymbol{g}), \tag{3.9}$$

so, by applying this expression, a PCDg becomes a PC0Dg.

3.4 Constructing simple boxes

Theorem 3 was about box shapes and particle positions. Let us first focus only on box shapes. In Theorem 3 it was shown that every box PC′0, which generates the same lattice as PC0, may be used to construct an infinite molecular system IS′, identical to IS. In this section we will propose two boxes, a triclinic and a rectangular one, generating the same lattice as an initial box PCDg.

We will first derive expressions for the lattice vectors of the lattice generated by a box PCDg. We start by considering an PC0Dg. As can be seen in Figure 3.5, the centres of replica boxes, fitted to this box are at the positions

$$\boldsymbol{K} = (\boldsymbol{g}+\boldsymbol{d}+\boldsymbol{e}+\boldsymbol{f}), \quad \boldsymbol{L} = (\boldsymbol{g}+\boldsymbol{b}+\boldsymbol{e}), \quad \boldsymbol{M} = (\boldsymbol{f}-\boldsymbol{c}+\boldsymbol{e}). \tag{3.10}$$

With some patience, it can be verified that every other replica box fitted to the original box, is shifted over an integer linear combination of $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$. *So, the whole space can be tiled with copies of the original box centred at the lattice points defined by* $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$. As long as the vectors $\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}$ are linearly independent the expressions (3.10) for $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$ are meaningful, i.e. they also hold for the boxes PCT1,. . .,PCT4.

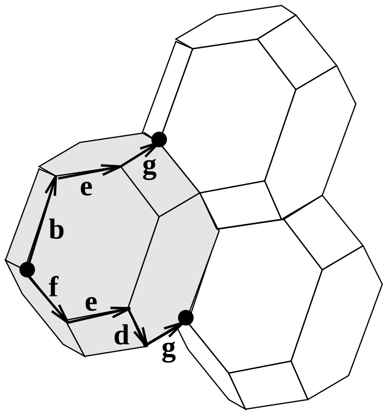

FIGURE 3.5 The vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$ defined by PCT5 with three replica boxes fitted along whole faces. It can be seen by inspection that $\boldsymbol{K} = (\boldsymbol{g} + \boldsymbol{d} + \boldsymbol{e} + \boldsymbol{f})$, $\boldsymbol{L} = (\boldsymbol{g} + \boldsymbol{b} + \boldsymbol{e})$, $\boldsymbol{M} = (\boldsymbol{f} - \boldsymbol{c} + \boldsymbol{e})$ (not shown).

With the lattice vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$ we can easily define a primitive cell which generates the lattice defined by $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$, namely the triclinic box spanned by the lattice vectors themselves (see Figure 3.6). We will call a box defined by the vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$, 'PCDKLM', and 'PC0DKLM' when it is centred at the origin. We will use these names only in relation with a given box PCDg or a given lattice $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$. The boxes PCDKLM and PC0DKLM can only be of the type PCT1.

Now we will introduce a rectangular box that generates the lattice defined by the

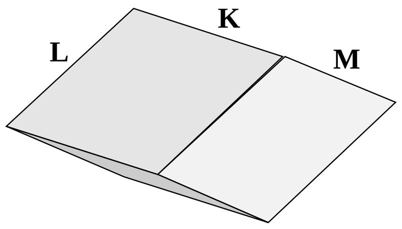

FIGURE 3.6 The most trivial primitive cell of a lattice is the triclinic box PCT1, spanned by the basis vectors of the lattice.

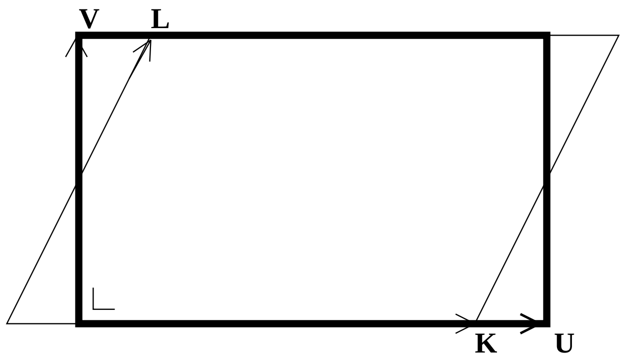

FIGURE 3.7 A rectangular primitive cell PCDUVW (fat lines), and the the PCDKLM from which it is derived (thin lines), both centred around the same point.

vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$. First the vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$ have to be reordered such that

$$|\boldsymbol{K}| \geq |\boldsymbol{L}| \geq |\boldsymbol{M}|. \tag{3.11}$$

As we will show in Appendix B this simplifies some calculations in a later stage. Using the reordered vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$, the vectors $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}$ spanning a rectangular primitive cell are given by a Gram-Schmidt orthogonalisation process (see Figure 3.7)

$$\boldsymbol{U} = \boldsymbol{K}, \quad \boldsymbol{V} = \boldsymbol{L} - (\boldsymbol{L} \cdot \hat{\boldsymbol{K}})\hat{\boldsymbol{K}}, \quad \boldsymbol{W} = \boldsymbol{M} - (\boldsymbol{M} \cdot \boldsymbol{K} \hat{\times} \boldsymbol{L})\boldsymbol{K} \hat{\times} \boldsymbol{L}\,, \tag{3.12}$$

with $\hat{\boldsymbol{a}} \equiv \frac{\boldsymbol{a}}{a}$ and with $\boldsymbol{b} \hat{\times} \boldsymbol{c} \equiv \frac{\boldsymbol{b} \times \boldsymbol{c}}{|\boldsymbol{b} \times \boldsymbol{c}|}$. The first expression needs no comment. The second expression means that $\boldsymbol{V}$ is perpendicular to $\boldsymbol{K}$, so to $\boldsymbol{U}$, and that it is in the plane defined by $\boldsymbol{K}$ and $\boldsymbol{L}$. The third expression means that $\boldsymbol{W}$ is perpendicular to the plane defined by $\boldsymbol{K}$ and $\boldsymbol{L}$, which implies that it is perpendicular to the plane defined by $\boldsymbol{U}$ and $\boldsymbol{V}$. Analogously with the nomenclature already introduced, we will call the primitive cell described by the vectors $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}$ PCDUVW or PC0DUVW. We will use these names only in relation with a given box PCDKLM or a given lattice $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$.

Boxes should be centred at lattice points, so, should be stacked *with relative shifts over the lattice vectors* $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$. This means that in a tiling with the boxes PCDUVW, the boxes are not fitted along whole faces (see Figure 3.8). This last fact looks a bit special because with the primitive cells PCT1, . . ., PCT5, the space could be tiled by fitting these boxes along whole faces. A way out of this seemingly strange property of PCT1R is by taking it as a PCT5, with some of its faces in the

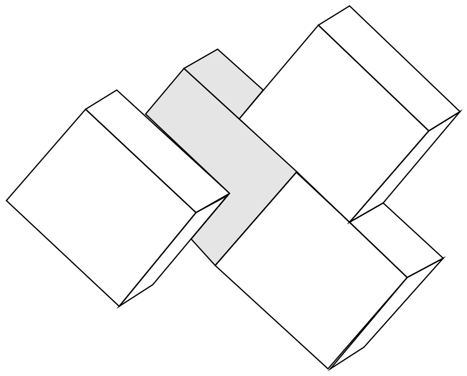

FIGURE 3.8 The box PCT1R has to be centred at lattice points, resulting in a tiling that is seemingly not a tiling along whole faces. However, by taking PCT1R as a special instance of PCT5, this tiling may be taken as a face to face tiling. This is indicated by the fact that every box PCT1R is directly surrounded by 14 boxes, just like a tiling with a general PCT5.

same plane. In general PCT5 has contact along whole faces with 14 adjacent boxes, just like PCDUVW. So, by taking PCT1R as a special instance of PCT5 the anomaly is explained.

Let us now briefly look at the volume of the various primitive cells we encountered. For a given lattice, defined by the vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$, the volume of the primitive cells PCDg, PCDKLM, and PCDUVW is the same, and is given by determinants $|\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}|$ and $|\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}|$. That is because to every lattice point belongs one primitive cell, no matter the shape of this primitive cell.

With this we have finished the discussion on how to transform one type of primitive cell into another type. In the following section we will see how the particles in one primitive cell should be mapped into another primitive cell.

3.5 Translating particles between primitive cells.

In Theorem 2 it was shown how to map particles from a primitive cell PC0 into a primitive cell PC′0, both defined by the same lattice but a different metric: particles should be shifted from PC0 to PC′0 over lattice vectors. In this way, the infinite molecular system generated by tiling the space with PC′0 is identical to the infinite system generated by tiling the space with PC0. In principle, that is all there is to mapping particles between primitive cells. The only remaining problem is to find for every particle the lattice vector bringing the particle from PC0 into PC′0. In general, it is difficult to give an explicit expression for the required shift. Therefore, we will not use a direct method to find the required lattice vector, but try lattice vectors. This can be done because it is possible to give an upper bound of the order of the required shift, i.e. if the required shift is $n_1\boldsymbol{K} + n_2\boldsymbol{L} + n_3\boldsymbol{M}$ it is possible to give an upper bound of n_1, n_2, n_3. For example, in Appendix A it is proved that particles in PC0Dg have to undergo at most first order shifts to be translated into PC0DKLM, i.e. $-1 \leq n_1, n_2, n_3 \leq 1$. This way of determining the required lattice vectors is not the most efficient, but it is general. Because the process of translating particles from PC0 into PC′0 is done in a preprocessing stage of the actual molecular simulation, the inefficiency is no problem.

The algorithm for translating particles

We will now discuss two algorithms to move particles from PC0Dg into the related PC0DKLM. We assume that we have a boolean function **`INPC0D2(r)`**, which determines whether **`r`** is in the box PC0DKLM. With this function, and using the boundedness of the required translations, the algorithm to move a particle from PC0Dg into PC0DKLM is as follows:

```
procedure PutIntoPC0D2(var r: vector);
          {r is shifted from PC0D1 into PC0D2}
   constant
      maxOrder = 1; {see Appendix A}
   var
      n1, n2, n3: integer;
      s: vector;
         {k,l,m are vectors, globally declared and initialised}
begin
   for n1 := -maxOrder to maxOrder do begin
      for n2 := -maxOrder to maxOrder do begin
         for n3 := -maxOrder to maxOrder do begin
            s := n1*k + n2*l + n3*m{vector operators}
            if InPC0D2(r + s) then begin
               r := r + s;              {vector operators}
               exit(PutIntoPC0D2);
            end; {if InPC0D2}
          end; {for n3}
     end; {for n2}
  end; {for n1}
end; {putIntoPC0D2}
```

This implementation of `PutIntoPC0D2` is not very efficient because translations over first order shifts are tried first, while the shift over zero is the most probable one. Later we will encounter a case where `maxOrder` is more than one, which results in even more inefficiency. Therefore we will now show a more efficient implementation of `PutIntoPC0D2`. The inefficiency is removed by first trying the most probable shift, which is the zero shift. Then, the second most probable shifts are tried, which are shifts over lattice vectors in the first layer around the origin. Then, if $max\_order > 1$, the shifts over lattice vectors in the third layer are tried, and so on.

```
procedure PutIntoPC0D2(var r: vector);
   var
      j, a, b, maxRadius: integer;

   procedure tryShiftingIntoBox(n1, n2, n3: integer);
      var
         shift: vector;
      begin           {note vector operations}
         shift := n1*k + n2*l + n3*m;
         if InPC0D2(r+shift) then begin
            r := r + shift;
            exit(PutIntoPC0D2);
         end; {if}
         if inbox1(r -- shift) then begin
            r := r - shift;
            exit(PutIntoPC0D2);
         end; {if}
      end; {tryShiftingIntoBox}

   begin{PutIntoPC0D2}
      maxRadius := 100;
      for j := 0 to maxRadius do begin
                  { try further and further away }
```

```
        for a := –j to j do begin
            for b := –j to j do begin
                tryShiftingIntoBox(a, b, j);
            end; {for b}
        end; {for a}
        for a := –j to j do begin
            for b := –j+1 to j–1 do begin
                tryShiftingIntoBox(a, j, b);
            end; {for b}
        end; {for a}
        for a := –j+1 to j–1 do begin
            for b := –j+1 to j–1 do begin
                tryShiftingIntoBox(j, b, a);
            end; {for b}
        end; {for a}
    end; {for j}
    FatalError('Max radius overflow.');
end; {PutIntoPC0D2}
```

Comments on this pseudo code: Note that we use vector operators in this code. The code consists of three similar blocks, each consisting of a nested loop over **`a`** and **`b`**. In the first block all lattice points in the top and bottom plane of a cube with 'radius' **`j`** are visited. In the second block the lattice points in the left and right plane are visited, and in the third block the lattice points in the front and back plane are visited.

In Appendix B it is shown that particles in PC0DKLM have to undergo at most second order shifts to be translated into PC0DUVW. This means that the procedure proposed in this subsection can also be used for that case, with of course the exception that in the algorithms **`maxOrder:=2`**. Later we will encounter a case where the maximum order of the translation is unbounded, but still zero shifts are the most probable ones with decreasing probability outwards. For that case, the second implementation of the procedure **`PutIntoPC0D2`** is the only one that can be used because in the first implementation infinite translations would be tried first.

3.6 An example transformation of a simulation

In the M.D. simulation package GROMOS, two box shapes are implemented: PCT1R, and PCT5R. In this section, as an example application of the theory, we will show how a simulation, formulated in PCT5R, can be transformed into a simulation in PCT1 and PCT1R. We will assume that PCT5R is centred at the origin, so, that it is actually a PC0T5R.

PC0T5R is obtained by cutting away pieces of a cube with edge lengths h. The cutting away of pieces is done with the Voronoi, or Wigner-Seitz construction, using the Euclidean metric. This results in a PC0T5R with edge vectors $\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{f}, \boldsymbol{g}$ given by

$$\boldsymbol{b} = \begin{pmatrix} 0 \\ -\frac{1}{4}h \\ -\frac{1}{4}h \end{pmatrix}, \quad \boldsymbol{c} = \begin{pmatrix} 0 \\ \frac{1}{4}h \\ -\frac{1}{4}h \end{pmatrix}, \quad \boldsymbol{d} = \begin{pmatrix} -\frac{1}{4}h \\ \frac{1}{4}h \\ 0 \end{pmatrix}, \tag{3.13}$$

$$\boldsymbol{e} = \begin{pmatrix} -\frac{1}{4}h \\ -\frac{1}{4}h \\ 0 \end{pmatrix}, \quad \boldsymbol{f} = \begin{pmatrix} -\frac{1}{4}h \\ 0 \\ \frac{1}{4}h \end{pmatrix}, \quad \boldsymbol{g} = \begin{pmatrix} -\frac{1}{4}h \\ 0 \\ -\frac{1}{4}h \end{pmatrix}. \tag{3.14}$$

Applying (3.10) gives the vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$

$$\boldsymbol{K} = \begin{pmatrix} -h \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{L} = \begin{pmatrix} -\frac{1}{2}h \\ -\frac{1}{2}h \\ -\frac{1}{2}h \end{pmatrix}, \quad \boldsymbol{M} = \begin{pmatrix} -\frac{1}{2}h \\ -\frac{1}{2}h \\ +\frac{1}{2}h \end{pmatrix}. \tag{3.15}$$

Applying (3.12) gives the vectors $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}$

$$\boldsymbol{U} = \begin{pmatrix} -h \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{V} = \begin{pmatrix} 0 \\ -\frac{1}{2}h \\ -\frac{1}{2}h \end{pmatrix}, \quad \boldsymbol{W} = \begin{pmatrix} -\frac{1}{2}h \\ -\frac{1}{2}h \\ +\frac{1}{2}h \end{pmatrix}. \tag{3.16}$$

It can be checked that the volume of each of these three figures (PC0T5R, PC0T1, PC0T1R) is $\frac{1}{2}h^3$.

In Figure 3.9a PC0T5R is shown with a (fancy) spherical molecule. The molecule is mapped into PC0T1 according to Theorem 2 (see Figure 3.9b). The fact that the molecule is ‘cut into pieces’ in PC0T1 indicates that the atoms of the molecule are shifted over different lattice vectors when translated from PC0T5R into PC0T1.

PCT5R is the most regular instance of PCT5. Consequently, as can be seen in (3.15), the lattice vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$ are also special, i.e., to create image particles

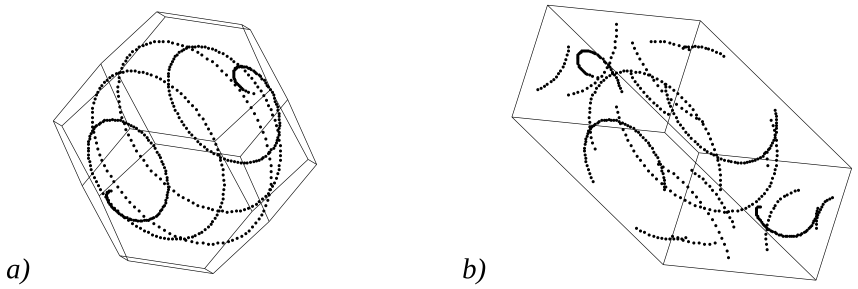

FIGURE 3.9 *a*: PC0T5R with a (fancy) spherical molecule. *b*: PC0T1 derived from PC0T5R, with the molecule mapped into it. It is instructive to copy *b*) on a transparent sheet, and to fit this copy at various faces to its original. It can then be seen that the molecule is reconstructed.

surrounding the original box PCT5R, the particles in the box have to undergo regular shifts. The regularity of these shifts is exploited in [5] to calculate in a simple way the required shifts. Quite appropriately, this shift pattern is called the 'checkerboard' periodic boundary condition. However, this shift method is only applicable to PCT5R, and the actual simulation is still done in PCT5R.

We have made some software available[3] as both Turbo Pascal and C code with executables. In **DEM1** the primitive cells PCT1, ..., PCT5 can be (randomly) generated, and visualised (in X). In **DEM2** the process of moving particles from PC0T5R into PC0DKLM and PC0DUVW is implemented. **DEM2** can thus be used by the M.D. community to transform existing simulations, formulated in PCT5R, into a simulation in PC0DKLM and PC0DUVW.

3.7 Related Topics

3.7.1 Pressure scaling

The most general pressure of an molecular system can be represented by a 3×3 tensor **P**. The pressure per dimension is defined as a vector $(\mathbf{P}_{xx}, \mathbf{P}_{yy}, \mathbf{P}_{zz})$, and the scalar pressure P is defined as

$$P \equiv \frac{1}{3}\text{trace}(\mathbf{P})\,. \tag{3.17}$$

(3) Can be obtained by anonymous ftp from **`ftp.cs.rug.nl`** in directory **`pub/mdbox`**.

In many M.D. simulations, every now and then the M.D. system, that is, the box and particle positions, is scaled depending on the most recently calculated pressure. In case the computational box is triclinic, it is well known how to scale the system [11]: in case only the scalar pressure is calculated, the box and particles are scaled in every dimension with the same factor. In case the pressure is calculated per dimension, the system is scaled per dimension, proportional to the components of the pressure vector. In case the full tensorial pressure is used, the system is scaled by multiplying all particle position and box vectors with the scaled pressure tensor. As a result of the the last two types of pressure scaling, the angles of the system may change.

With the notions developed in this article, it is clear how to scale the system when the computational box is one of PCT2, . . ., PCT5 and a pressure scaling per dimension or a full tensorial pressure is used. Then, just as in the case of a triclinic box, *the system may be scaled by scaling box vectors and particle positions per dimension, resp. by multiplying box vectors* (*$\boldsymbol{b}$*, . . ., *$\boldsymbol{g}$*) *and particle positions with a scaled tensor* **P**. This is because the infinite M.D. system may be taken as a tiling of the space with one of PCT2, . . ., PCT5 but just as well as a tiling with PCT1.

3.7.2 Lattice reduction

Until now our attention has been focussed on transforming simulations in a complex box into simulations in a simple box, i.e. on transformations between different box types. We will now discuss a transformation from one PCT1 into another PCT1, both defining the same lattice.

Let us suppose that a 2-D simulation of a long thin molecule is set up as shown in Figure 3.10a. In principle the simulation may be done in this box but for a number of practical reasons this may be unattractive. For example, then the cut off sphere may be located in many boxes at the same time. To improve this situation, a general technique, called lattice reduction [6], may be applied. According to Theorem 3 a simulation may be done in every box that defines the same lattice as the original box. When we assume that the original box defines the lattice basis vectors $\boldsymbol{K}, \boldsymbol{L}$, the same lattice is defined by the basis vectors $\boldsymbol{K}, \boldsymbol{L} - n\boldsymbol{K}$ with $n \in \mathbb{Z}$. So, *the simulation may just as well be done in a box defined by the vectors* $\boldsymbol{K}, \boldsymbol{L} - n\boldsymbol{K}$. When the particles are moved from the original box to the new one this results in a system as shown in Figure 3.10b. This method may be generalised to 3-D.

Let us now be a bit more precise. For a given box PCT1, spanned by the vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$, we look for three vectors $\boldsymbol{K}', \boldsymbol{L}', \boldsymbol{M}'$, such that the vectors $\boldsymbol{K}', \boldsymbol{L}', \boldsymbol{M}'$

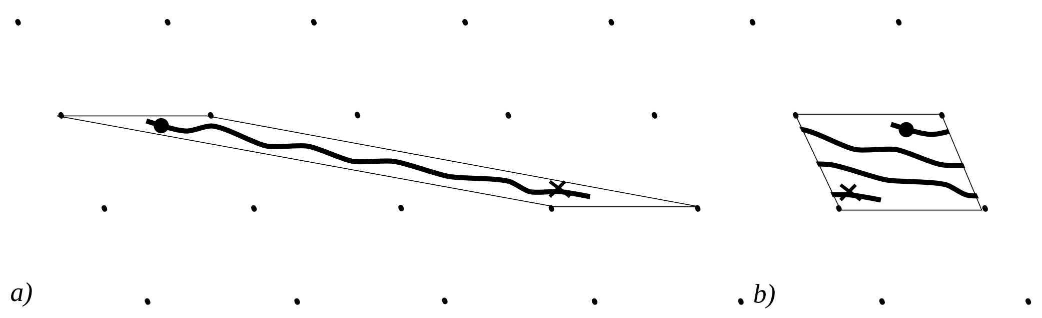

FIGURE 3.10 2-D example of two primitive cells of the same type (parallelogram), defining the same lattice. Applying lattice reduction to *a* gives *b*, resulting in a cell with shorter spanning vectors than the original primitive cell. The molecule in *a* is mapped into *b* according to Theorem 2.

define the same lattice as the vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$. Moreover, the vectors $\boldsymbol{K}', \boldsymbol{L}', \boldsymbol{M}'$ should span a 'nice' box, where nice means something like 'as cubic as possible'. The process of transforming the vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$ into the vectors $\boldsymbol{K}', \boldsymbol{L}', \boldsymbol{M}'$ is called lattice reduction. Many different notions of 'reduced' exist in the literature, but roughly speaking, they all mean that the cell $\boldsymbol{K}'$, $\boldsymbol{L}'$, $\boldsymbol{M}'$ is as cubic as possible. It has been shown [6] that in 3-D the three shortest, linearly independent lattice vectors are a basis of the lattice. We will define a reduced basis as follows: *a reduced basis consists of the three shortest, linearly independent lattice vectors.*

After the process of lattice reduction, particles from the box $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$ should be mapped into the box $\boldsymbol{K}', \boldsymbol{L}', \boldsymbol{M}'$. This should be done according to Theorem 2, i.e. particles should be shifted over lattice vectors. Which lattice basis is used, $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$ or $\boldsymbol{K}', \boldsymbol{L}', \boldsymbol{M}'$, does not matter because both are a basis of the same lattice. The algorithm from Section 3.5 may be used to shift particles over the required lattice vectors, although, unlike the situation in Section 3.6, now there is no upper limit on the required shift (called **`max_shift`** in the algorithm).

A useful application of lattice reduction has to do with the maximum allowed cut-off radius. More precise: *for a given triclinic box spanned by the (unreduced) vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$, how large may the cut-off radius be at most, such that a particle has no interactions with two corresponding particles?* This may be reformulated as: how large may the cut-off sphere be at most, such that it does not contain corresponding points? As can be seen in Figure 3.11a, it is not enough that $max\ \boldsymbol{R}_{co} = \frac{1}{2} min(|\boldsymbol{K}|, |\boldsymbol{L}|, |\boldsymbol{M}|)$. Using our foregoing definition of 'reduced

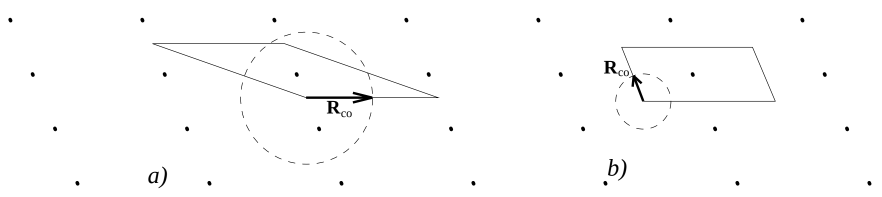

FIGURE 3.11 a: 2-D example of an unreduced primitive cell. When R_{co} is chosen half the length of the shortest vector spanning the primitive cell, the cut-off sphere still contains corresponding particles. b: When R_{co} is chosen half the length of the shortest vector spanning the *reduced* primitive cell, the cut-off sphere does not contain corresponding particles.

basis', the answer is

$$max\ \boldsymbol{R}_{co} = \frac{1}{2} min(|\boldsymbol{K}'|, |\boldsymbol{L}'|, |\boldsymbol{M}'|)\,, \tag{3.18}$$

i.e. the cut-off radius should be less than half the shortest *reduced* lattice basis vector (Figure 3.11b).

3.7.3 Long range order

Stacking boxes in a space filling way introduces a well defined long range order in the infinite system. This long range order may influence the results of a simulation. For example, when the box shape is chosen such that it defines a long range order close to the long range order of ice, it may happen that in a simulation of pure water, the water freezes above 0 ^{o}Celsius. By simulating water in a box with a long range order incompatible with the long range order of ice, the water may be liquid below 0 ^{o}Celsius. Probably, for every solvent and depending on the type of simulation, there is an optimal long range order, so that the solvent behaves normal. So, when setting up a simulation, the resulting lattice must be compatible with the desired long range order. This means that the shape of the computational box is not completely free anymore.

3.7.4 How to set up a simulation

From the foregoing it will be clear that a molecular simulation can be done without using complex boxes. We will now show that setting up a simulation can also be done without using complex boxes, i.e. we will show that it is not necessary to set up a simulation in a complex box which is subsequently transformed into a simulation in a simple box. Historically, M.D. simulations are done in complex boxes because it was believed that this was the only way to get a minimal volume simulation. An implicit condition was that the box should contain an *unfragmented* molecule. This superfluous implicit condition has led to the use of complex shaped boxes. As will be clear from this chapter, it is not forbidden that the molecule is stored in the box in pieces, provided that the molecule is reconstructed when the boxes are stacked.

Let us now assume that one single large molecule has to be simulated in a solvent. The molecule has been given, the solvent has to be added later. We will designate this molecule by 'mol'. See Figure 3.12. In general a molecule is not allowed to interact with its own image molecules, so, in the infinite system the smallest distance between two atoms of two different images of mol should be at least R_{co} apart. For this purpose we surround mol by an enlarged convex hull, such that no atom of mol is closer than $1/2R_{co}$ to this enlarged hull. We will designate this enlarged hull of mol by MOL. Three replica's of MOL, with the same orientation as MOL, are designated by MOL$'$, MOL$''$, and MOL$'''$.

To set up a PBC simulation with a minimal amount of solvent means that we have to find a densest lattice packing of translates of MOL. A practical approach to this minimisation problem is to fit MOL$'$, . . ., MOL$'''$ to MOL, such that the volume of the tetrahedron defined by these four molecules is minimal. More exactly, when we define the vector $\boldsymbol{K}$ as the vector connecting the centre of MOL with the centre of MOL$'$, the vector $\boldsymbol{L}$ as the vector connecting MOL with MOL$''$, and the vector $\boldsymbol{M}$ as the vector connecting MOL with MOL$'''$, the problem boils down to: minimise $|\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}|$ so that each of the molecules MOL, . . ., MOL$'''$ is touched[4] by the other three. This is a minimisation problem in three parameters. This can be seen as follows: because MOL$'$ has to touch MOL, the position of MOL$'$ is determined by two angles, say θ and ϕ, where the origin of these two angles is somewhere in MOL. MOL$''$ should touch MOL and MOL$'$, so there is only one degree of freedom in the placement of MOL$''$. Finally, MOL$'''$ has to touch the first three ones, so the placement of MOL$'''$ is completely determined by the positions of MOL, . . ., MOL$''$. Thus, we have a minimisation problem in three variables, (minimise $|\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}|$), subject to six contact conditions (contact between every pair of MOL, . . ., MOL$'''$).

(4) It is a well known property of the densest lattice packing of convex figures that every figure is touched by twelve other ones.

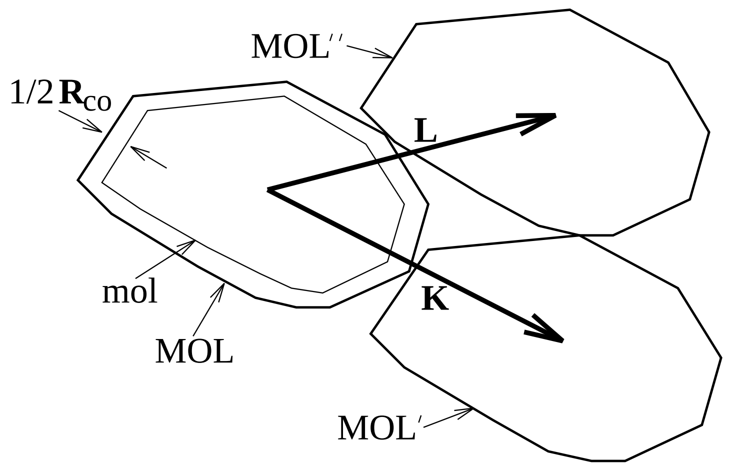

FIGURE 3.12 To find a computational box with a minimal volume, containing a single molecule MOL, three translates of MOL have to be fitted to MOL, defining three vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$, such that the volume of box defined by $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$ is minimal. After finding such a minimal box, the atoms of MOL can be translated into this box by shifts over lattice vectors, where the lattice is defined by $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$.

A near minimal solution can be found by a standard minimisation procedure as for example NAG routine **`E04UCF`**. When a minimal volume configuration of MOL, $\cdots$, MOL″ has been found, the vectors $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$ are the vectors defining the triclinic simulation box.[5] By shifts over lattice vectors, the atoms of mol can now be brought into this triclinic box, and the empty space can be filled with solvent. Of course, if desired this box can be transformed into a rectangular box as described earlier in this chapter.

3.7.5 Which box to use: the triclinic or the rectangular?

The main message of this chapter is that complex shaped boxes with particles, as for example PCT5 and its degenerates PCT4, $\cdots$, PCT2, can be transformed into simpler ones, i.e. into PCDKLM and PCDUVW. Which one of these last two is the best one as a simulation box is not very clear. The choice may be slightly influenced by some parts of the simulated system and the simulation methods used. We will

(5) Obviously, when the simulation has to be set up with a predefined long range order the optimisation process may be skipped.

briefly discuss some of these aspects. Still, this discussion will not lead to a strong preference.

Neighbour searching: As has been shown in [7], using a grid search technique significantly improves the efficiency of neighbour searching. The essence of the grid search technique is that a grid is constructed in the computational box, and that for every particle it is determined in which grid cell it is located. Neighbour searching for a given particle then boils down to inspecting its own and directly neighbouring grid cells for neighbouring particles. In [7] it was shown that a grid size of $L = \frac{1}{2}R_{co}$ gives an optimal neighbour searching speed, which is six times faster than neighbour searching without using a grid. However, as far as we can see now, the grid search technique can only work efficiently when the grid cells are rectangular, or even better, cubic. Obviously, a rectangular box can be partitioned in cells in a natural way. This does not hold for the non-rectangular box PCDKLM, so then many grid cells will be empty. Therefore we think that in case neighbour searching is implemented with the grid search technique, the rectangular box is to preferred over the triclinic box. Of course, although the box is rectangular, image particles are created by shifting particles over lattice vectors, so not over the orthogonal vectors $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}$.

The function `inbox(r)`: Every now and then during an M.D. simulation, particles that moved outside the computational box have to be reset into the box. To check whether a particle is inside or outside the computational box, the boolean function **`inbox(r)`** is used. When $\boldsymbol{r}$ is inside the box the function evaluates to true. Obviously, when PCT1R is used as a computational box, and the directions of $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}$ coincide with the x, y, z axes, **`inbox(r)`** can be implemented by checking independently in three directions in what range the components of $\boldsymbol{r}$ are. This does not work that simple in case of a non-rectangular triclinic box. Then a linear transformation on $\boldsymbol{r}$ has to be done, or some other more complex calculation. So, for the implementation of **`inbox(r)`** it is desirable to work with PCDUVW.

Full pressure scaling: As we explained before, three kinds of pressure scaling are possible in an M.D. simulation: uniform in every direction, scaling per dimension, and by using the full pressure tensor. In general, the last two ways of pressure scaling will change the directions of the vectors spanning the computational box. This means that when PCDUVW is used as a computational box, the angles between the vectors $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}$ will change, i.e. afterwards the box will not be rectangular anymore. Then, in principle, the function **`inbox(r)`** will not work properly, and the search grid will not be rectangular anymore. However, the computational effort of recalculating the vectors $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}$ and resetting particles is small, so possibly pressure scaling will not be an obstacle for using a rectangular box.

Summarising one may say that a rectangular box simplifies the implementation of some parts of molecular algorithms (grid search, **`inbox(r)`**), but causes small complications in the implementation of other parts.

3.8 Conclusion

In this chapter we studied the possible shapes of the computational box of molecular simulations with PBC. For this purpose, five types of boxes are suitable: triclinic, the hexagonal prism, two types of dodecahedrons, and the truncated octahedron, for short PCT1, $\cdots$, PCT5. We showed that PCT1, $\cdots$, PCT4 are degenerate instances of PCT5.

The main purpose of this chapter is to show that for every simulation in some type of box, simulations in the other four types can be devised which give exactly the same simulation result, i.e. it is shown that boxes with a complex shape are superfluous. Therefore we first showed how to transform the complex shaped box into a triclinic one, and how to transform the triclinic one into a rectangular one. Then we showed how to map particles from the complex shaped box into the simpler ones.

Important conceptual tools in this chapter are lattices and primitive cells. It was shown that a simulation box may be taken as a primitive cell. Tiling the space with a box with particles gives an infinite molecular system. A cornerstone of this chapter is a theorem in which it is stated which transformations are allowed on the original box with particles, subject to the condition that the infinite system generated by tiling the space with the transformed box with particles gives the same infinite system as the initial box.

Although most of this chapter is about transforming complex boxes with particles, this does not mean that a molecular simulation should be set up in a complex box which is subsequently transformed into a simpler box. On the contrary, because every simulation in a complex box can be transformed in a simpler one in a triclinic box, nothing is lost when a simulation is set up right away in a triclinic box. In Subsection 3.7.4 it is explained how this can be done.

With the concepts developed in this chapter, some questions are clarified. This includes amongst others: pressure scaling in a complex box, and the long range order introduced by the shape of the box.

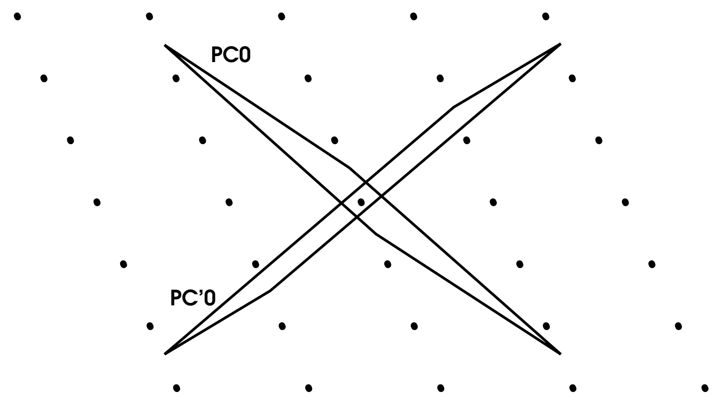

FIGURE A1 Particles in PC0 have to be shifted over more than first order shifts to be translated into PC′0.

Acknowledgements

This research was initiated by the insights of Berend Reitsma. Discussions with Wim Hesselink led to the use of an alternative metric. The discussions with Michael Renardus about the text and the figures improved this chapter significantly.

Appendix A

We will prove that particles in the box PC0Dg have to be shifted at most over first order shifts, i.e. over $n_1\boldsymbol{K} + n_2\boldsymbol{L} + n_3\boldsymbol{M}$, with $-1 \leq n_1, n_2, n_2 \leq 1$, to be located in the related PC0DKLM. It is instructive to see that this does not hold in general (see Figure A1), i.e. that particles in a primitive cell PC0 potentially require infinite shifts in order to be located in a related PC′0, where 'related' means that the cells define the same lattice.

The reason that between PC0Dg and PC0DKLM at most first order shifts are required, has to do with the special choice of $\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}$. Consequently, when PC0Dg is long and thin, PC0DKLM is also long and thin and is oriented in the same direction. In this way PC0Dg and PC0DKLM have a large overlap, so going from one to the other cell, only limited particle shifts are required.

Let us now give a more formal proof. Using the 3-D nomenclature, we give the proof for the 2-D case, but it is simple to extend it to 3-D. We start with PC0Dg

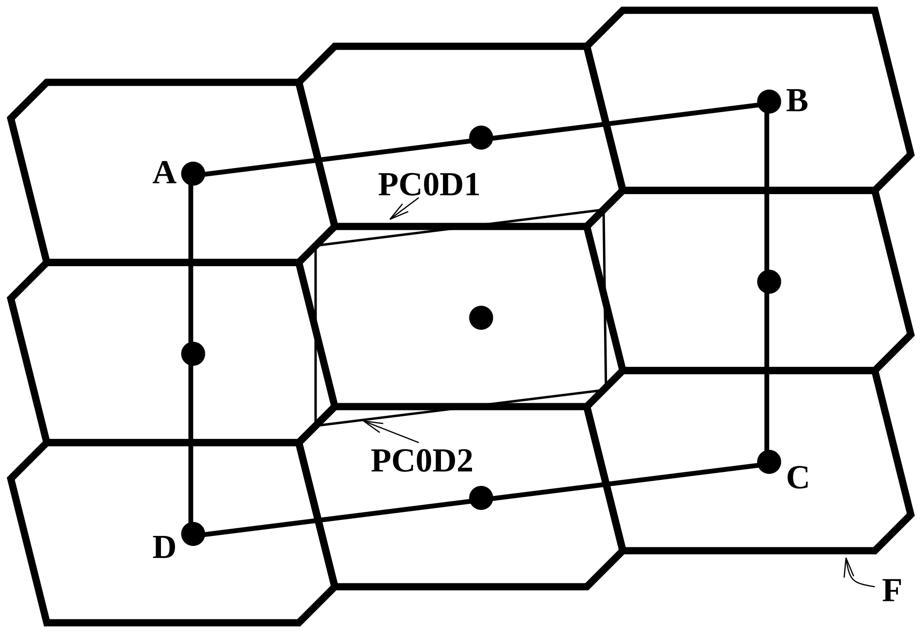

FIGURE A2 Every point belonging to PC0DKLM is also in F.

and its eight first order images. We will call this arrangement of 9 tiles ‘F’. See Figure A2. In this same figure the related PC0DKLM is drawn. It is constructed by scaling the rhombus A,B,C,D with a factor 1/2. The rhombus defined by A,B,C,D is in F because the boundary of the rhombus A,B,C,D is in F. This last fact is because, when two space fillers (either 2-D or 3-D) are fitted face to face, the line connecting their centres of symmetry lies in these two figures. Because PC0DKLM is in the rhombus A,B,C,D, it is also in F. The particles in F are shifted from PC0Dg over at most first order lattice vectors, so, the particles in PC0Dg have to be shifted over at most first order shifts to be located in PC0DKLM.

Appendix B

In this appendix we will show the necessity to order the vectors $\boldsymbol{K}$, $\boldsymbol{L}$, $\boldsymbol{M}$ before calculating $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}$. We first show what may go wrong when it is not done. Just as in Appendix A, using the 3-D nomenclature, we will do this for 2-D.

Suppose that we have a PC0DKLM as shown in Figure B1. We can construct a

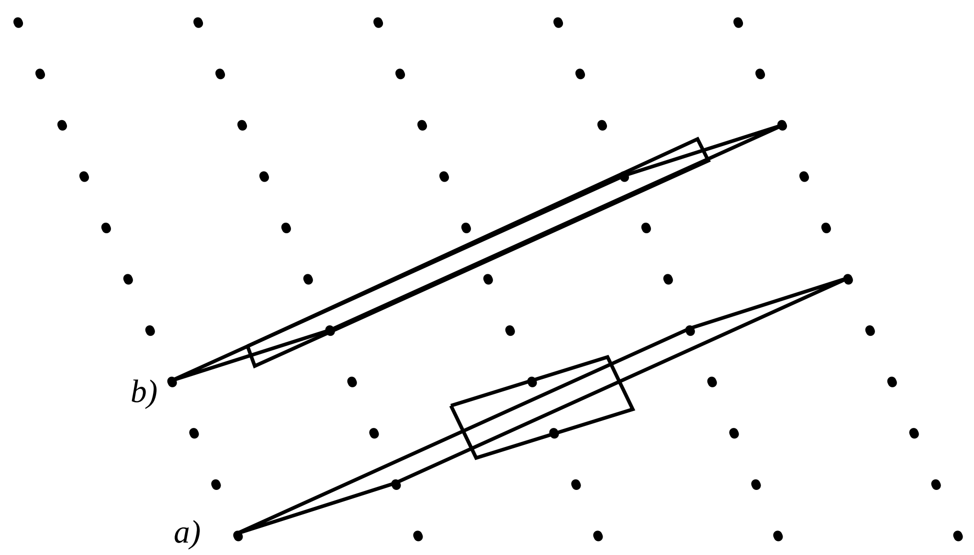

FIGURE B1 a: When the vectors K,L,M are not ordered such that $|\boldsymbol{K}| \geq \boldsymbol{L}| \geq |\boldsymbol{M}|$, possibly the boxes PCDKLM and PCDUVW have little overlap, so possibly, particles require shifts over high order lattice vectors to be moved from one box into another one. b: The boxes PCDKLM and PCDUVW have a large overlap because the longest box vector is called $\boldsymbol{K}$.

rectangular primitive cell from PC0DKLM in two ways: by $\boldsymbol{U} = \boldsymbol{K}$ and $\boldsymbol{V} \perp \boldsymbol{U}$ (Figure B1a), and by $\boldsymbol{U} = \boldsymbol{L}$ and $\boldsymbol{V} \perp \boldsymbol{U}$. (Figure B1b). From these figures it is clear that PC0DUVW has a large overlap with PC0DKLM when the longest one of the pair K,L is defined as U. So, ordering K,L,M prevents possibly infinite shifts of particles going from PC0DKLM into PC0DUVW.

Now, just like in Appendix A, we will determine the maximum shift required to bring a particle from PC0DKLM into PC0DUVW. We suppose that the vectors $\boldsymbol{K}$, $\boldsymbol{L}$ are ordered, i.e. that $|\boldsymbol{K}| \geq |\boldsymbol{L}|$. We define ‘F’ as the array of nine cells, created by all possible first order shifts of PC0DKLM (Figure B2). To show that every point of PC0DUVW is in F it is sufficient to show that C$'$ is in F. This last statement can be reformulated as: show that the distance of C$'$ to C is less than the distance C to D. This last statement is true because CC$' <$ CE $\leq$ DC. So, at most first order shifts are required to bring particles from PC0DKLM into PC0DUVW.

For the 3-D case the reasoning above can be applied twice. Thus, in 3-D at most second order shifts are required to bring particles from PC0DKLM into PC0DUVW.

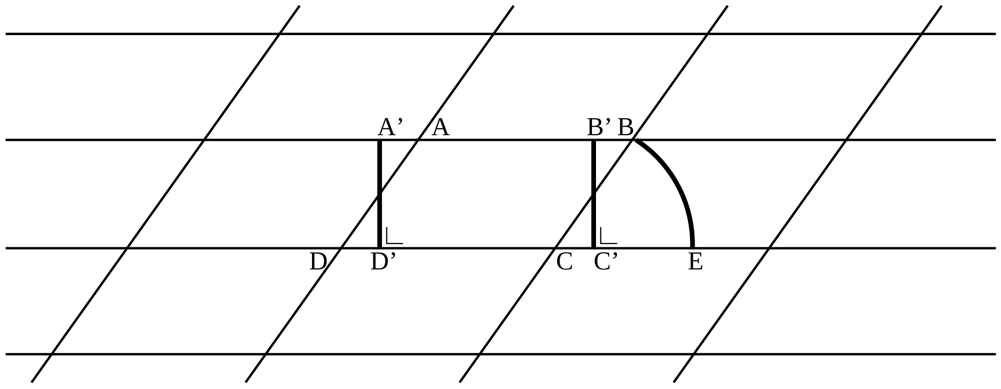

FIGURE B2 When the vectors K, L, M are ordered such that $|K| \geq |L| \geq |M|$, every point belonging to PC0DUVW is also in F.

Literature

[1] L. Fejes Tóth, *Regular Figures*, Pergamon Press, London, 1964, 114-119.
[2] W.F. van Gunsteren and H.J.C. Berendsen, *Groningen Molecular Simulation (GROMOS) Library Manual Biomos*, Groningen, The Netherlands(1987).
[3] H. Bekker, E.J. Dijkstra, H.J.C. Berendsen, M.K.R. Renardus, *An efficient, box shape independent non-bonded force and virial algorithm for Molecular Dynamics.* Molecular Simulation, 1995, Vol. 14, pp. 137-151.
[3] *See also Chapter 2.*
[4] H. Minkowski, *Allgemeine Lehrsätze über die konvexen Polyeder, Nachr. Ges. Wiss. Göttingen, Math.-Phys. Kl.*, or in the collected works *Gesammelte Abhandlungen*, Chelsea, New York, 1967.
[5] W. Dzwinel, J. Kitowski, J. Mościński, *Mol. Sim.*, Vol. 7 (1991) pp. 171-179.
[6] Handbook of Convex Geometry, Vol. B, page 917. Edited by P.M. Gruber and J.M. Wills, North-Holland, 1993.
[7] H. Bekker, H.J.C. Berendsen, E.J. Dijkstra, S. Achterop, R.v. Drunen, D. v.d. Spoel, A. Sijbers, H. Keegstra, B. Reitsma and M.K.R. Renardus, GROMACS: a parallel computer for Molecular Dynamics Simulations, in *Conf. Proc. Physics Computing '92*, 257-261, World Scientific Publishing Co. Singapore, New York, London.
[8] D.J. Adams, Computer simulation of ionic systems: the distorting effect of the boundary conditions. *Chem. Phys. Letters*, Vol. 62, nr. 2 (1979) 329.

[9] S.S. Wang, J.A. Krumhansl, Superposition assumption. II. High density fluid Argon. *J. Chem Phys.* 56(1972), 4287.
[10] C. Kittel, Introduction to solid state physics. John Wiley and Sons, New York, 1976.
[11] H.J.C. Berendsen, J.P.M. Postma, W.F. van Gunsteren, A. DiNola, J.R. Haak, J. Chem. Phys. 81 (1984), 3684.