2.1 Newtonian deterministic mechanics

In this section, we consider hard disks and spheres[1] colliding with each other and with walls. Instantaneous pair collisions conserve momentum, and wall collisions merely reverse one velocity component, that normal to the wall. Between collisions, disks move straight ahead, in the same manner as free particles. To numerically solve the equations of motion—that is, do a molecular dynamics simulation—we simply propagate all disks up to the next collision (the next event) in the whole system. We then compute the new velocities of the collision partners, and continue the propagation (see Fig. 2.1 and the schematic Alg. 2.1 (`event-disks`)).

```
procedure event-disks
input $\{\mathbf{x}_1,\ldots,\mathbf{x}_N\},\{\mathbf{v}_1,\ldots,\mathbf{v}_N\},t$
$\{t_{\text{pair}},k,l\} \leftarrow$ next pair collision
$\{t_{\text{wall}},j\} \leftarrow$ next wall collision
$t_{\text{next}} \leftarrow \texttt{min}[t_{\text{wall}},t_{\text{pair}}]$
for $m=1,\ldots,N$ do
    { $\mathbf{x}_m \leftarrow \mathbf{x}_m + (t_{\text{next}}-t)\mathbf{v}_m$
if ($t_{\text{wall}} < t_{\text{pair}}$) then
    { call wall-collision($j$)
else
    { call pair-collision($k,l$)
output $\{\mathbf{x}_1,\ldots,\mathbf{x}_N\},\{\mathbf{v}_1,\ldots,\mathbf{v}_N\},t_{\text{next}}$
——
```

Algorithm 2.1 `event-disks`. Event-driven molecular dynamics algorithm for hard disks in a box (see Alg. 2.4 (`event-disks(patch)`)).

Our aim in the present section is to implement this event-driven molecular dynamics algorithm and to set up our own simulation of hard disks and spheres. The program is both simple and exact, because the integration of the equations of motion needs no differential calculus, and the numerical treatment contains no time discretization.

2.1.1 Pair collisions and wall collisions

We determine the time of the next pair collision in the box by considering all pairs of particles $\{k,l\}$ and isolating them from the rest of the system (see Fig. 2.2). This leads to the evolution equations

$$\mathbf{x}_k(t) = \mathbf{x}_k(t_0) + \mathbf{v}_k \cdot (t-t_0),$$
$$\mathbf{x}_l(t) = \mathbf{x}_l(t_0) + \mathbf{v}_l \cdot (t-t_0).$$

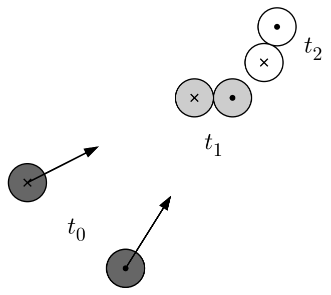

Fig. 2.2 Motion of two disks from time t_0 to their pair collision time t_1.

[1]In this chapter, the words "disk" and "sphere" are often used synonymously. For concreteness, most programs are presented in two dimensions, for disks.

A collision occurs when the norm of the spatial distance vector

$$\underbrace{\mathbf{x}_k(t) - \mathbf{x}_l(t)}_{\Delta\mathbf{x}(t)} = \underbrace{\Delta_{\mathbf{x}}}_{\mathbf{x}_k(t_0)-\mathbf{x}_l(t_0)} + \underbrace{\Delta_{\mathbf{v}}}_{\mathbf{v}_k-\mathbf{v}_l} \cdot (t - t_0) \tag{2.1}$$

equals twice the radius σ of the disks (see Fig. 2.2). This can happen at two times t_1 and t_2, obtained by squaring eqn (2.1), setting $|\Delta\mathbf{x}| = 2\sigma$, and solving the quadratic equation

$$t_{1,2} = t_0 + \frac{-(\Delta_{\mathbf{x}} \cdot \Delta_{\mathbf{v}}) \pm \sqrt{(\Delta_{\mathbf{x}} \cdot \Delta_{\mathbf{v}})^2 - (\Delta_{\mathbf{v}})^2((\Delta_{\mathbf{x}})^2 - 4\sigma^2)}}{(\Delta_{\mathbf{v}})^2}. \tag{2.2}$$

The two disks will collide in the future only if the argument of the square root is positive and if they are approaching each other ($(\Delta_{\mathbf{x}} \cdot \Delta_{\mathbf{v}}) < 0$, see Alg. 2.2 (`pair-time`)). The smallest of all the pair collision times obviously gives the next pair collision in the whole system (see Alg. 2.1 (`event-disks`)). Analogously, the parameters for the next wall collision are computed from a simple time-of-flight analysis (see Fig. 2.3, and Alg. 2.1 (`event-disks`) again).

```
procedure pair-time
input $\Delta_{\mathbf{x}}$ $(\equiv \mathbf{x}_k(t_0) - \mathbf{x}_l(t_0))$
input $\Delta_{\mathbf{v}}$ $(\equiv \mathbf{v}_k - \mathbf{v}_l \neq 0)$
$\Upsilon \leftarrow (\Delta_{\mathbf{x}} \cdot \Delta_{\mathbf{v}})^2 - |\Delta_{\mathbf{v}}|^2(|\Delta_{\mathbf{x}}|^2 - 4\sigma^2)$
if ($\Upsilon > 0$ and $(\Delta_{\mathbf{x}} \cdot \Delta_{\mathbf{v}}) < 0$) then
  { $t_{\text{pair}} \leftarrow t_0 - \left[(\Delta_{\mathbf{x}} \cdot \Delta_{\mathbf{v}}) + \sqrt{\Upsilon}\right] / \Delta_{\mathbf{v}}^2$
else
  { $t_{\text{pair}} \leftarrow \infty$
output $t_{\text{pair}}$
——
```

Algorithm 2.2 `pair-time`. Pair collision time for two particles starting at time t_0 from positions $\mathbf{x}_k$ and $\mathbf{x}_l$, and with velocities $\mathbf{v}_k$ and $\mathbf{v}_l$.

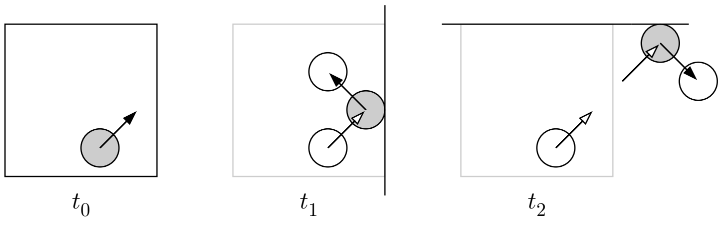

Fig. 2.3 Computing the wall collision time $t_{\text{wall}} = \min(t_1, t_2)$.

Continuing our implementation of Alg. 2.1 (`event-disks`), we now compute the velocities after the collision: collisions with a wall of the box lead to a change of sign of the velocity component normal to the

```
procedure pair-collision
input $\{\mathbf{x}_k, \mathbf{x}_l\}$ (particles in contact: $|\mathbf{x}_k - \mathbf{x}_l| = 2\sigma$)
input $\{\mathbf{v}_k, \mathbf{v}_l\}$
$\Delta_\mathbf{x} \leftarrow \mathbf{x}_k - \mathbf{x}_l$
$\hat{\mathbf{e}}_\perp \leftarrow \Delta_\mathbf{x}/|\Delta_\mathbf{x}|$
$\Delta_\mathbf{v} \leftarrow \mathbf{v}_k - \mathbf{v}_l$
$\mathbf{v}'_k \leftarrow \mathbf{v}_k - \hat{\mathbf{e}}_\perp(\Delta_\mathbf{v} \cdot \hat{\mathbf{e}}_\perp)$
$\mathbf{v}'_l \leftarrow \mathbf{v}_l + \hat{\mathbf{e}}_\perp(\Delta_\mathbf{v} \cdot \hat{\mathbf{e}}_\perp)$
output $\{\mathbf{v}'_k, \mathbf{v}'_l\}$
——
```

Algorithm 2.3 `pair-collision`. Computing the velocities of disks (spheres) k and l after an elastic collision (for equal masses).

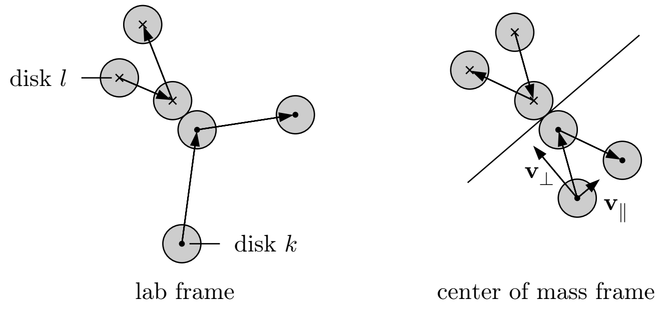

Fig. 2.4 Elastic collision between equal disks k and l, as seen in two different reference frames.

wall involved in the collision. Pair collisions are best analyzed in the center-of-mass frame of the two disks, where $\mathbf{v}_k + \mathbf{v}_l = 0$ (see Fig. 2.4). Let us write the velocities in terms of the perpendicular and parallel components $\mathbf{v}_\perp$ and $\mathbf{v}_\parallel$ with respect to the tangential line between the two particles when they are exactly in contact. This tangential line can be thought of as a virtual wall from which the particles rebound:

$$\underbrace{\begin{aligned} \mathbf{v}_k &= \mathbf{v}_\parallel + \mathbf{v}_\perp \\ \mathbf{v}_l &= -\mathbf{v}_\parallel - \mathbf{v}_\perp \end{aligned}}_{\text{before collision}}, \quad \underbrace{\begin{aligned} \mathbf{v}'_k &= \mathbf{v}_\parallel - \mathbf{v}_\perp \\ \mathbf{v}'_l &= -\mathbf{v}_\parallel + \mathbf{v}_\perp \end{aligned}}_{\text{after collision}}.$$

The changes in the velocities of particles k and l are $\mp 2\mathbf{v}_\perp$. Introducing the perpendicular unit vector $\hat{\mathbf{e}}_\perp = (\mathbf{x}_k - \mathbf{x}_l)/|\mathbf{x}_k - \mathbf{x}_l|$ allows us to write $\mathbf{v}_\perp = (\mathbf{v}_k \cdot \hat{\mathbf{e}}_\perp)\hat{\mathbf{e}}_\perp$ and $2\mathbf{v}_\perp = (\Delta_\mathbf{v} \cdot \hat{\mathbf{e}}_\perp)\hat{\mathbf{e}}_\perp$, where $2\mathbf{v}_\perp = \mathbf{v}'_k - \mathbf{v}_k$ gives the change in the velocity of particle k. The formulas coded into Alg. 2.3 (`pair-collision`) follow immediately. We note that $\hat{\mathbf{e}}_\perp$ and the changes in velocities $\mathbf{v}'_k - \mathbf{v}_k$ and $\mathbf{v}'_l - \mathbf{v}_l$ are relative vectors and are thus the same in all inertial reference frames. The program can hence be used directly with the lab frame velocities.

Algorithm 2.1 (`event-disks`) is rigorous, but naive in that it computes collision times even for pairs that are far apart and also recomputes all pair collision times from scratch after each event, although they change only for pairs involving a collision partner. Improvements could easily be worked into the program, but we shall not need them. However, even our basic code can be accepted only if it runs through many billions of collisions (several hours of CPU time) without crashing. We must care about code stability. In addition, we should invest in simplifying input–output: it is a good strategy to let the program search for a file containing the initial configuration (velocities, positions, and time), possibly the final configuration of a previous calculation. If no such file is found, a legal initial configuration should be automatically generated by the program. Finally, we should plot the results in the natural units of time for the simulation, such that during a unit time interval each particle undergoes about one pair collision (and the whole system has $N/2$ pair collisions). In this way, the results become independent of the scale of the initial velocities.

Our molecular dynamics algorithm moves from one collision to the next with an essentially fixed number of arithmetic operations, requiring on the order of a microsecond for a few disks on a year 2005 laptop computer. Each collision drives the system further in physical time. Solving Newton's dynamics thus appears to be a mere question of computer budget.

2.1.2 Chaotic dynamics

Algorithm 2.1 (`event-disks`) solves the hard-sphere equations of motion on the assumption that the calculation of collision times, positions, velocity changes, etc. is done with infinite precision. This cannot really be achieved on a computer, but the question arises of whether it matters that numbers are truncated after 8 or 16 decimal digits. It is easiest to answer this question by pitting different versions of the same event-driven algorithm against each other, starting from identical initial conditions, but with all calculations performed at different precision levels: in one case in short format (single precision), and in the other case in long format (double precision). The standard encoding of 32-bit floating-point numbers $\pm a \times 10^b$ uses one bit for the sign, eight bits for the exponent b, and 23 bits for the fraction a, so that the precision—the ratio of neighboring numbers that can be represented on the computer—is approximately $\epsilon = 2^{-23} \simeq 1.2\times10^{-7}$. For a 64-bit floating point number, the precision is about $\epsilon = 1\times10^{-16}$. Most computer languages allow one to switch precision levels without any significant rewriting of code.

The two versions of the four-disk molecular dynamics calculation, started off from identical initial conditions (as in Fig. 2.1), get out of step after as few as 25 pair collisions (see Fig. 2.5), an extremely small number compared with the million or so collisions which we can handle per second on a year 2005 laptop computer.

This situation is quite uncomfortable: our computational results, for

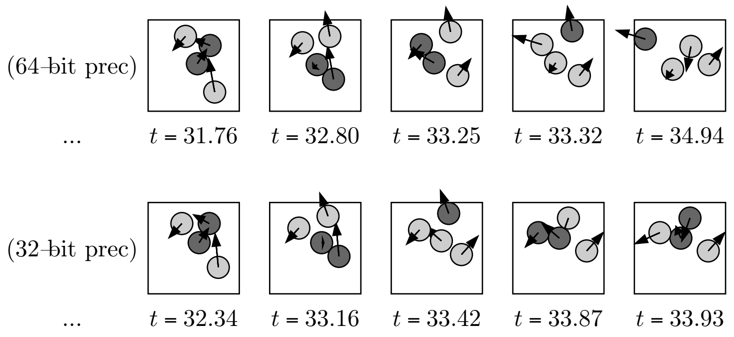

Fig. 2.5 Calculations starting from the initial configuration of Fig. 2.1, performed with 64-bit precision (*top*) and with 32-bit precision (*bottom*).

computing times beyond a few microseconds, are clearly uncontrolled. We may drive up the precision of our calculation with special number formats that are available in many computer languages. However, this strategy cannot defeat the onset of chaos, that is, cure the extreme sensitivity to the details of the calculation. It will be impossible to control a hard-sphere molecular dynamics simulation for a few billion events.

The chaos in the hard-sphere model has its origin in the negative curvature of the spheres' surfaces, which magnifies tiny differences in the trajectory at each pair collision and causes serious rounding errors in computations and humbling experiences at the billiard table (see Fig. 2.6). On the other hand, this sensitivity to initial conditions ensures that even finite systems of disks and spheres can be described by statistical mechanics, as will be discussed later in this chapter.

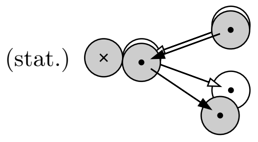

Fig. 2.6 Magnification of a difference in trajectories through a pair collision, in the reference frame of a stationary disk.

2.1.3 Observables

In the previous subsections, we have integrated the equations of motion for hard spheres in a box, but have only looked at the configurations, without evaluating any observables. We shall do the latter now. For simplicity, we consider a projected density $\eta_y(a)$, essentially the fraction of time for which the y-coordinate of any particle is equal to a. More precisely, $\{\eta_y(a)\,\mathrm{d}a\}$ is the fraction of time that the y-coordinate of a disk center spends between a and $a + \mathrm{d}a$ (see Fig. 2.7). It can be computed exactly for given particle trajectories between times $t = 0$ and $t = T$:

$$\begin{Bmatrix} y\text{-density} \\ \text{at } y = a \end{Bmatrix} = \eta_y(a) = \frac{1}{T} \sum_{\substack{\text{intersections } i \\ \text{with gray strip} \\ \text{in Fig. 2.7}}} \frac{1}{|v_y(i)|}. \tag{2.3}$$

In Fig. 2.7, there are five intersections (the other particles must also be considered). At each intersection, $1/|v_y|$ must be added, to take into account the fact that faster particles spend less time in the interval $[a, a + \mathrm{d}a]$, and thus contribute less to the density at a.

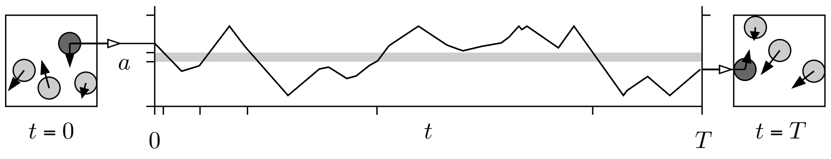

Fig. 2.7 y-coordinate vs. time for one disk. The y-density $\eta_y(a)$ is computed from the time spent between a and $a + \mathrm{d}a$ (see eqn (2.3)).

The y-density at a corresponds to the observable $\mathcal{O}(t) = \delta\left[y(t) - a\right]$ (where we have used the Dirac δ-function), and can also be obtained from the time average

$$\langle \mathcal{O}(t) \rangle_T = \frac{1}{T} \int_0^T \mathrm{d}t\ \mathcal{O}(t),$$

so that

$$\eta_y^T(a) = \frac{1}{T} \int_0^T \mathrm{d}t\ \delta\left[y(t) - a\right]. \tag{2.4}$$

Seasoned theorists can derive the formula for the y-density in eqn (2.3) in a split second from eqn (2.4) via the transformation rules for the δ-function. Algorithm 2.1 (`event-disks`) numerically solves the equations of motion for hard disks without discretization errors or other imperfections. In addition to this best possible method of data acquisition, we may analyze the data using eqn (2.3), without losing information. In principle, we are limited only by the finite total simulation time.

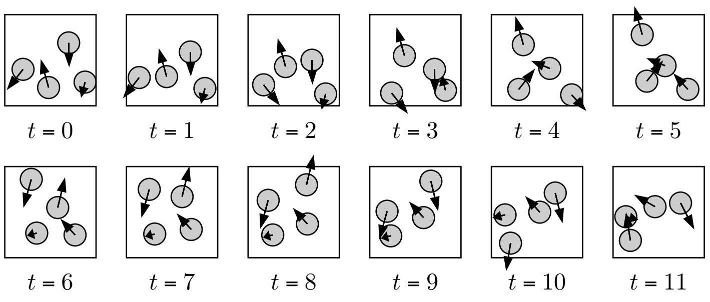

Fig. 2.8 Stroboscopic snapshots of the simulation shown in Fig. 2.1.

It is possible to implement data analysis as in eqn (2.3), but this approach is somewhat overambitious: we can certainly obtain exact run averages, but because the number of runs is finite and the runs are not infinitely long, statistical errors would still creep into the calculation. It is thus justified to take a more leisurely approach to data analysis and simply discretize the time average (2.4):

$$\langle \mathcal{O}(t) \rangle_T \simeq \frac{1}{M} \sum_{i=1}^{M} \mathcal{O}(t_i).$$

We thus interrupt the calculation in Alg. 2.1 (`event-disks`) at fixed, regular time intervals and take stroboscopic snapshots (see Fig. 2.8 and Alg. 2.4 (`event-disks(patch)`)). The snapshots can then be incorpo-

```
procedure event-disks(patch)
...
$i_{\min} \leftarrow \texttt{Int}(t/\Delta_t) + 1$
$i_{\max} \leftarrow \texttt{Int}(t_{\text{next}}/\Delta_t)$
for $i = i_{\min}, \ldots, i_{\max}$ do
    $t' \leftarrow i\Delta_t - t$
    output $\{\mathbf{x}_1 + t'\mathbf{v}_1, \ldots, \mathbf{x}_N + t'\mathbf{v}_N\}, \{\mathbf{v}_1, \ldots, \mathbf{v}_N\}$
...
——
```

Algorithm 2.4 `event-disks(patch)`. Alg. 2.1 (`event-disks`), modified to output stroboscopic snapshots (with a time interval Δ_t).

rated into histograms of observables, and give results identical to an analysis of eqn (2.3) in the limit of vanishing width of the time slices or for large run times. The density at point y then converges to a value independent of the initial configuration, a true time average:

$$\eta_y(a) = \lim_{T \to \infty} \frac{1}{T} \sum_k \int_0^T \mathrm{d}t \; \delta[y_k(t) - a],$$

where $k = 1, \ldots, N$ represent the disks in the box.

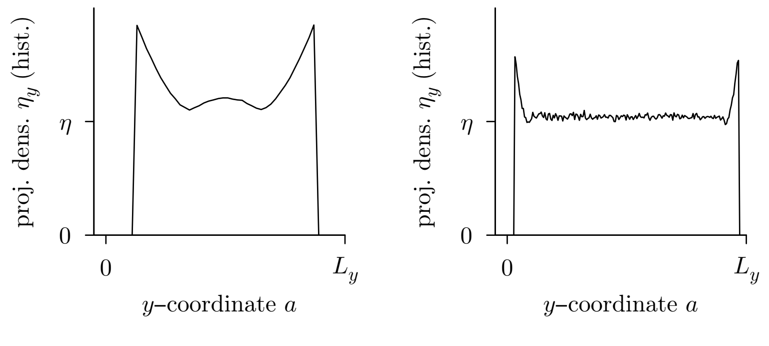

Fig. 2.9 Projected density in a square box at density $\eta = 0.18$. *Left*: $N = 4$. *Right*: $N = 64$ (from Alg. 2.4 (`event-disks(patch)`)).

Some data obtained from Alg. 2.4 (`event-disks(patch)`), for hard-disk systems in a square box, are shown in Fig. 2.9. In the two graphs, the covering density is identical, but systems are of different size. It often surprises people that the density in these systems is far from uniform, even though the disks do not interact, other than through their hard-core. In particular, the boundaries, especially the corners, seem to attract disks. Systems with larger $\{N, L_x, L_y\}$ clearly separate into a bulk region (the inner part) and a boundary.

2.1.4 Periodic boundary conditions

Bulk properties (density, correlations, etc.) differ from properties near the boundaries (see Fig. 2.9). In the thermodynamic limit, almost the whole system is bulk, and the boundary reduces to nearly nothing. To compute bulk properties for an infinite system, it is often advantageous to eliminate boundaries even in simulations of small systems. To do so, we could place the disks on the surface of a sphere, which is an isotropic, homogeneous environment without boundaries. This choice is, however, problematic because at high density, we cannot place the disks in a nice hexagonal pattern, as will be discussed in detail in Chapter 7. Periodic boundary conditions are used in the overwhelming majority of cases (see Fig. 2.10): in a finite box without walls, particles that move out through the bottom are fed back at the top, and other particles come in from one side when they have just left through the opposite side. We thus identify the lower and upper boundaries with each other, and similarly the left and right boundaries, letting the simulation box have the topology of an abstract torus (see Fig. 2.11). Alternatively, we may represent a box with periodic boundary conditions as an infinite, periodically repeated box without walls, obtained by gluing together an infinite number of identical copies, as also indicated in Fig. 2.10.

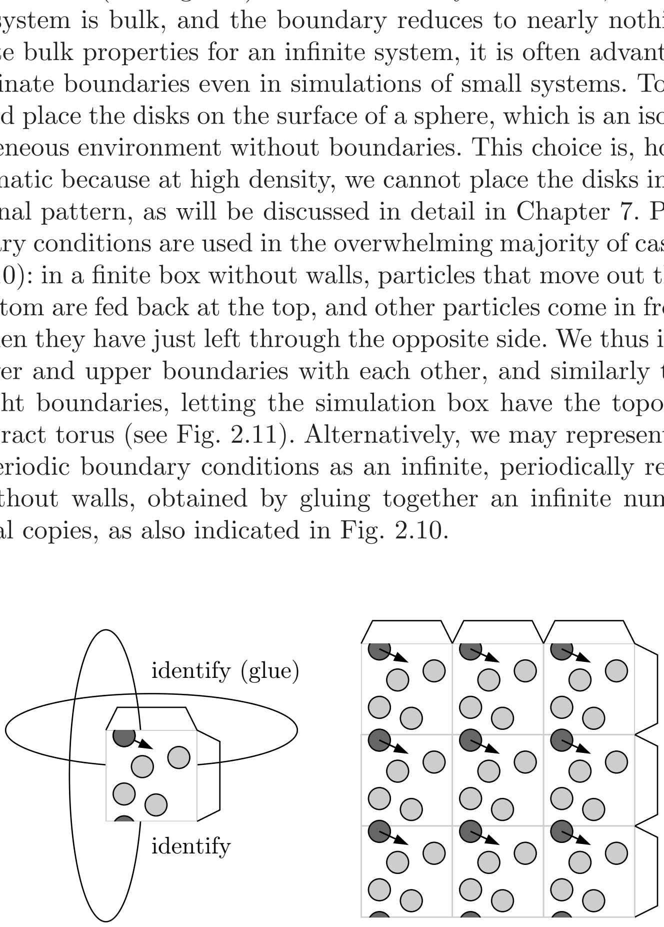

Fig. 2.10 Sample with periodic boundary conditions, interpreted as a finite torus (*left*) or an infinite periodic system (*right*).

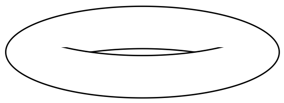

Fig. 2.11 Gluing together the sides of the square in Fig. 2.10 generates an object with the topology of a torus.

For hard spheres, the concept of periodic boundary conditions is relatively straightforward. In more complicated systems, however (particles with long-range interaction potentials, quantum systems, etc.), sloppy implementation of periodic boundary conditions is frequently a source of severe inconsistencies which are difficult to detect, because they enter the program at the conception stage, not during execution. To avoid trouble, we must think of a periodic system in one of the two frameworks of Fig. 2.10: either a finite abstract torus or a replicated infinite system. We should stay away from vaguely periodic systems that have been set up with makeshift routines lacking a precise interpretation.

Algorithm 2.1 (`event-disks`) can be extended to the case of periodic boundary conditions with the help of just two subroutines. As one particle can be described by many different vectors (see the dark particle in the right part of Fig. 2.10), we need Alg. 2.5 (`box-it`) to find the representative vector inside the central simulation box, with an x-coordinate between 0 and L_x and a y-coordinate between 0 and L_y.[2] Likewise, since many different pairs of vectors correspond to the same two particles, the difference vector between these particles takes many values. Algorithm 2.6 (`diff-vec`) computes the shortest difference vector between all representatives of the two particles and allows one to decide whether two hard disks overlap.

```
procedure box-it
input $\mathbf{x}$
$x \leftarrow \texttt{mod}\,(x, L_x)$
if $(x < 0)$ $x \leftarrow x + L_x$
$y \leftarrow \texttt{mod}\,(y, L_y)$
if $(y < 0)$ $y \leftarrow y + L_y$
output $\mathbf{x}$
——
```

Algorithm 2.5 `box-it`. Reducing a vector $\mathbf{x} = \{x, y\}$ into a periodic box of size $L_x \times L_y$.

```
procedure diff-vec
input $\{\mathbf{x}, \mathbf{x}'\}$
$\Delta_{\mathbf{x}} \leftarrow \mathbf{x}' - \mathbf{x}$ $(\Delta_{\mathbf{x}} \equiv \{x_\Delta, y_\Delta\})$
call box-it $(\Delta_{\mathbf{x}})$
if $(x_\Delta > L_x/2)$ $x_\Delta \leftarrow x_\Delta - L_x$
if $(y_\Delta > L_y/2)$ $y_\Delta \leftarrow y_\Delta - L_y$
output $\Delta_{\mathbf{x}}$
——
```

Algorithm 2.6 `diff-vec`. The difference $\Delta_{\mathbf{x}} = \{x_\Delta, y_\Delta\}$ between vectors $\mathbf{x}$ and $\mathbf{x}'$ in a box of size $L_x \times L_y$ with periodic boundary conditions.

Periodic systems have no confining walls, and thus no wall collisions.

[2]Some computer languages allow the output of the `mod`() function to be negative (e.g. `mod`$(-1, 3) = -1$). Others implement `mod`() as a nonnegative function (so that, for example, `mod`$(-1, 3) = 2$). The **if ( )** statements in Alg. 2.5 (`box-it`) are then unnecessary.