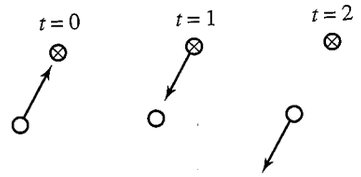

Figure 14.7 Example of a collision from a barrier. At $t = 1$ the particle moves to the barrier site and then reverses its velocity. The symbol ⊗ denotes a barrier site.

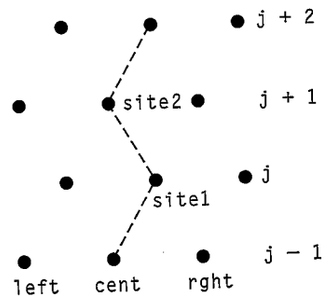

Figure 14.8 We update `site1` and `site2` at the same time. The rows are indexed by `j`. The dotted line connects sites in the same column.

The `step` method runs through the entire lattice and moves all the particles. The updated values of the sites are placed in the array `newLattice`. We then go through the `newLattice` array, implement the relevant collision rule at each site, and write the results into the array `Lattice`.

The movement of the particles is accomplished as follows. Because the even rows are horizontally displaced one half a lattice spacing from the odd rows, we need to treat odd and even rows separately. In the `step` method we loop through every other row and update `site1` and `site2` at the same time. An example will show how this update works. The statement

```
rght[j-1] |= site1 & RIGHT_DOWN;
```

means that if there is a particle moving to the right and down at `site1`, then the bit corresponding to `RIGHT_DOWN` is added to the site `rght` (see Figure 14.8). The statement

```
cent[j] |= site1 & (STATIONARY|BARRIER) | site2 & RIGHT_DOWN;
```

means that a stationary particle at `site1` remains there, and if `site1` is a barrier, it remains so. If `site2` has a particle moving in the direction `RD`, then `site1` will receive this particle.

To maintain a steady flow rate, we add the necessary horizonal momentum to the lattice uniformly after each time step. The procedure is to chose a site at random and determine if it is possible to change the sites's horizontal momentum. If so, we then remove the left bit and add the right bit or vice versa. This procedure is accomplished by the statements at the end of the `step` method.

Listing 14.13 Listing of the `LatticeGas` class.

```
package org.opensourcephysics.sip.ch14.latticegas;
import org.opensourcephysics.display.*;
```



```
import java.awt.*;
import java.awt.geom.*;

public class LatticeGas implements Drawable {
   // input parameters from user
   public double flowSpeed;           // controls pressure
   // size of velocity arrows displayed
   public double arrowSize;
   public int spatialAveragingLength; // spatial averaging of velocity
   public int Lx, Ly;                 // linear dimensions of lattice
   public int[][] lattice, newLattice;
   private double numParticles;
   static final double SQRT3_OVER2 = Math.sqrt(3)/2;
   static final double SQRT2 = Math.sqrt(2);
   static final int RIGHT = 1, RIGHT_DOWN = 2, LEFT_DOWN = 4;
   static final int LEFT = 8, LEFT_UP = 16, RIGHT_UP = 32;
   static final int STATIONARY = 64, BARRIER = 128;
   // maximum number of particles per site
   static final int NUM_CHANNELS = 7;
   // 7 channel bits plus 1 barrier bit per site
   static final int NUM_BITS = 8;
   // total number of possible site configurations = 2^8
   static final int NUM_RULES = 1<<8;
   // 1 << 8 means move the zeroth bit over 8 places to the left to the
   // eighth bit

   static final double ux[] = {1.0, 0.5, -0.5, -1.0, -0.5, 0.5, 0};
   static final double uy[] = {0.0, -SQRT3_OVER2, -SQRT3_OVER2, 0.0,
        SQRT3_OVER2, SQRT3_OVER2, 0};
   // averaged velocities for every site configuration
   static final double[] vx, vy;
   static final int[] rule;

   static { // set rule table
      // default rule is the identity rule
      rule = new int[NUM_RULES];
      for(int i = 0;i<BARRIER;i++) {
         rule[i] = i;
      }
      // abbreviations for channel bit indices
      int RI = RIGHT, RD = RIGHT_DOWN, LD = LEFT_DOWN;
      int LE = LEFT, LU = LEFT_UP, RU = RIGHT_UP;
      int S = STATIONARY;
      // three-particle zero momentum rules
      rule[LU|LD|RI] = RU|LE|RD;
      rule[RU|LE|RD] = LU|LD|RI;
      // three-particle rules with unperturbed particle
      rule[RU|LU|LD] = LU|LE|RI;
      rule[LU|LE|RI] = RU|LU|LD;
      rule[RU|LU|RD] = RU|LE|RI;
      rule[RU|LE|RI] = RU|LU|RD;
      rule[RU|LD|RD] = LE|RD|RI;
      rule[LE|RD|RI] = RU|LD|RD;
      rule[LU|LD|RD] = LE|LD|RI;
      rule[LE|LD|RI] = LU|LD|RD;
      rule[RU|LD|RI] = LU|RD|RI;
```