conditions are a reasonable approximation. However, fixed boundary conditions usually lead to nonphysical reflections off the edges, and a variety of approaches have been used including boundary conditions equivalent to a conducting medium that gradually absorbs the fields. In some cases physically motivated boundary conditions can be employed. For example, in simulations of microwave cavity resonators (see Problem 10.24), the appropriate boundary conditions are that the tangential component of **E** and the normal component of **B** vanish at the boundary.

As we have noted, **E** and ρ are defined at different times than **B** and **j**. This half-step approach leads to well-behaved equations that are stable over a range of parameters. An analysis of the stability requirement for the Yee–Visscher algorithm shows that the time step Δt must be smaller than the spatial grid Δl by

$$c\Delta t \le \frac{\Delta l}{\sqrt{3}} \quad \text{(stability requirement).} \tag{10.58}$$

The `Maxwell` class implements the Visscher–Yee finite difference algorithm for solving Maxwell's equations. The field and current data are stored in multidimensional arrays `E`, `B`, and `J`. The first index determines the vector component. The last three indices represent the three spatial coordinates. The `current` method models a positive current flowing for one time unit. This current flow produces both electric and magnetic fields. Because charge is conserved, the current flow produces an electrostatic dipole. Negative charge remains at the source and a positive charge is deposited at the destination. Note that the `doStep` method invokes a `damping` method that reduces the fields at points near the boundaries, thereby absorbing the emitted radiation and reducing the reflected electromagnetic waves. Your understanding of the Yee–Visscher algorithm for finding solutions to Maxwell's equations will be enhanced by carefully reading the `MaxwellApp` program and the `Maxell` class.

Listing 10.8 The `Maxwell` class implements the Yee–Visscher finite difference approximation to Maxwell's equations.

```java
package org.opensourcephysics.sip.ch10;

public class Maxwell {
   // static variables determine units and time scale
   static final double pi4 = 4*Math.PI;
   static final double dt = 0.03;
   static final double dl = 0.1;
   static final double escale = dl/(4*Math.PI*dt);
   static final double bscale = escale*dl/dt;
   static final double jscale = 1;
   double dampingCoef = 0.1; // damping coefficient near boundaries
   int size;
   double t;                 // time
   double[][][][] E, B, J;

   public Maxwell(int size) {
      this.size = size;
      // 3D arrays for electric field, magnetic field, and current
      // last three indices indicate location, first index indicates x,
      // y, or z component
      E = new double[3][size][size][size];
```

```java
      B = new double[3][size][size][size];
      J = new double[3][size][size][size];
   }

   public void doStep() {
      current(t); // update the current
      computeE(); // step electric field
      computeB(); // step magnetic field
      damping();  // damp transients
      t += dt;
   }

   void current(double t) {
      final int mid = size/2;
      double delta = 1.0;
      for(int i = -3;i<5;i++) {
         J[mid+i][mid][mid][0] = (t<delta) ? +1 : 0;
      }
   }

   void computeE() {
      for(int x = 1;x<size-1;x++) {
         for(int y = 1;y<size-1;y++) {
            for(int z = 1;z<size-1;z++) {
               double curlBx = (B[1][x][y][z]-B[1][x][y][z+1]
                               +B[2][x][y+1][z]-B[2][x][y][z])/dl;
               E[0][x][y][z] += dt*(curlBx-pi4*J[0][x][y][z]);
               double curlBy = (B[2][x][y][z]-B[2][x+1][y][z]
                               +B[0][x][y][z+1]-B[0][x][y][z])/dl;
               E[1][x][y][z] += dt*(curlBy-pi4*J[1][x][y][z]);
               double curlBz = (B[0][x][y][z]-B[0][x][y+1][z]
                               +B[1][x+1][y][z]-B[1][x][y][z])/dl;
               E[2][x][y][z] += dt*(curlBz-pi4*J[2][x][y][z]);
            }
         }
      }
   }

   void computeB() {
      for(int x = 1;x<size-1;x++) {
         for(int y = 1;y<size-1;y++) {
            for(int z = 1;z<size-1;z++) {
               double curlEx = (E[2][x][y][z]-E[2][x][y-1][z]
                               +E[1][x][y][z-1]-E[1][x][y][z])/dl;
               B[0][x][y][z] -= dt*curlEx;
               double curlEy = (E[0][x][y][z]-E[0][x][y][z-1]
                               +E[2][x-1][y][z]-E[2][x][y][z])/dl;
               B[1][x][y][z] -= dt*curlEy;
               double curlEz = (E[1][x][y][z]-E[1][x-1][y][z]
                               +E[0][x][y-1][z]-E[0][x][y][z])/dl;
               B[2][x][y][z] -= dt*curlEz;
            }
         }
      }
   }
```