CHAPTER

10 Electrodynamics

We compute the electric fields due to static and moving charges, describe methods for computing the electric potential in boundary value problems, and solve Maxwell's equations numerically.

10.1 ■ STATIC CHARGES

Suppose we want to know the electric field $\mathbf{E}(\mathbf{r})$ at the point $\mathbf{r}$ due to N point charges $q_1, q_2, \ldots, q_N$ at fixed positions $\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N$. We know that $\mathbf{E}(\mathbf{r})$ satisfies a superposition principle and is given by

$$\mathbf{E}(\mathbf{r}) = K \sum_i^N \frac{q_i}{|\mathbf{r}-\mathbf{r}_i|^3}(\mathbf{r}-\mathbf{r}_i), \tag{10.1}$$

where $\mathbf{r}_i$ is the fixed location of the ith charge, and K is a constant that depends on our choice of units. One of the difficulties associated with electrodynamics is the competing systems of units. In the SI (or rationalized MKS) system of units, the charge is measured in coulombs (C) and the constant K is given by

$$K = \frac{1}{4\pi\epsilon_0} \approx 9.0 \times 10^9\ \mathrm{N \cdot m^2/C^2} \quad \text{(SI units)}. \tag{10.2}$$

The constant ϵ_0 is the electrical permittivity of free space. This choice of units is not convenient for computer programs because $K \gg 1$. Another popular system of units is the Gaussian (cgs) system for which the constant K is absorbed into the unit of charge so that $K = 1$. Charge is in electrostatic units or esu. One feature of Gaussian units is that the electric and magnetic fields have the same units. For example, the (Lorentz) force on a particle of charge q and velocity $\mathbf{v}$ in an electric field $\mathbf{E}$ and a magnetic field $\mathbf{B}$ has the form

$$\mathbf{F} = q\left(\mathbf{E} + \frac{\mathbf{v}}{c} \times \mathbf{B}\right) \quad \text{(Gaussian units)}. \tag{10.3}$$

These virtues of the Gaussian system of units lead us to adopt this system for this chapter even though SI units are used in introductory texts.

10.2 ■ ELECTRIC FIELDS

The electric field is an example of a *vector field* because it defines a vector quantity at every point in space. One way to visualize this field is to divide space into a discrete grid and to

draw arrows in the direction of $\mathbf{E}$ at the vertices of this grid. The length of the arrow can be chosen to be proportional to the magnitude of the electric field. Another possibility is to use color or gray scale to represent the magnitude. Because we have found that using an arrow's color rather than its length to represent field strength produces a more effective representation of vector fields over a wider dynamic range; the `Vector2DFrame` class in the Open Source Physics frames package uses the color representation (see Appendix 10A).

The `ElectricFieldApp` program computes the electric field due to an arbitrary number of point charges. A charge is created using the control's x, y, and q parameters when the Calculate button is clicked. Each time the Calculate button is clicked, another charge is created. Whenever a charge is added or moved, the electric field is recomputed in the `calculateField` method. The Reset button removes all charges.

Because the number of charges can change, we need a way to obtain the position and charge for each point charge so that the electric field can be computed. The drawing panel for `frame` contains a list of all drawable objects, but we need a way to obtain only those objects that are of type `Charge`. The following statements show one way of doing this.

```
List chargeList = frame.getDrawables(Charge.class);
Iterator it = chargeList.iterator();
```

The argument `Charge.class` tells the `frame.getDrawables` method to return only a list of objects that can be cast to the `Charge` class.[1] These objects are then placed in an object of type `ArrayList` which implements the Java interface `List`. The `iterator` method returns an object called `it` which implements the `Iterator` interface. We then use the object `it` to loop through all the charges using the `next` and `hasNext` methods of the interface `Iterator`. As the name implies, an iterator is a convenient way to access a list without explicitly counting its elements. You will modify `ElectricFieldApp` to include a moving test charge in Problem 10.1.

Listing 10.1 `ElectricFieldApp` computes and displays the electric field from a list of point charges.

```
package org.opensourcephysics.sip.ch10;
import java.awt.event.MouseEvent;
import java.util.*;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.frames.Vector2DFrame;

public class ElectricFieldApp extends AbstractCalculation implements
    InteractiveMouseHandler {
  int n = 20;                                          // grid points on a side
  double a = 10;                                       // viewing side length
  double[][][] eField = new double[2][n][n]; // stores electric field
  Vector2DFrame frame = new Vector2DFrame("x", "y", "Electric field");

  public ElectricFieldApp() {
    frame.setPreferredMinMax(-a/2, a/2, -a/2, a/2);
    frame.setZRange(false, 0, 2);
    frame.setAll(eField); // sets the vector field
```

[1] The syntax `Charge.class` uses a small portion of the Java reflection API to determine the type of object that is being requested. The reflection API is an advanced feature of Java.