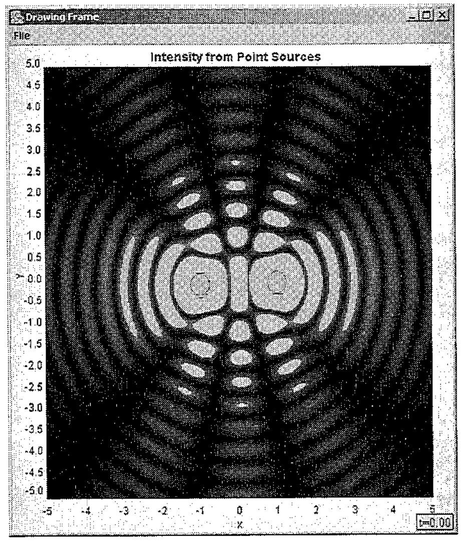

Figure 9.3 The computed energy density in the vicinity of two point sources.

point sources and displaying a two-dimensional animation of (9.61) as shown in Figure 9.3 and described in Appendix 9C.

Sources are represented by circles and are added to the frame when a custom button invokes the `createSource` method.

```
public void createSource() {
   InteractiveShape ishape = InteractiveShape.createCircle(0, 0, 0.5);
   frame.addDrawable(ishape);
   initPhasors();
   frame.repaint();
}
```

Users can create as many sources as they wish. The program later retrieves a list of sources from the frame using the latter's `getDrawables` method.

The program uses $n \times n$ arrays to store the real and imaginary values. The code fragment from the `initPhasors` method shown in the following starts the process by obtaining a list of point sources in the frame. We then use an `Iterator` to access each source as we sum the vector components at each grid point.

```
ArrayList list=frame.getDrawables();   // gets list of point sources
// creates an iterator for the list
Iterator it=list.iterator();
// these two statements are combined in the final code
```

`List` and `Iterator` are interfaces that are implemented by the objects returned by `frame.getDrawables` and `list.iterator`, respectively. As the name implies, an iterator is a convenient way to access a list without explicitly counting its elements. The iterator's `getNext` method retrieves elements from the list, and the `hasNext` method returns true if the end of the list has not been reached.

The `initPhasors` method in `HuygensApp` computes the phasors at every point by summing the phasors at each grid point. Note how the distance from the source to the observation point is computed by converting the grid's index values to world coordinates.

```
Iterator it = frame.getDrawables().iterator();  // source iterator
while(it.hasNext()) {
   InteractiveShape source = (InteractiveShape) it.next();
   // world coordinates for source
   double xs = source.getX(), ys = source.getY();
   for(int ix = 0;ix<n;ix++) {
      double x = frame.indexToX(ix);
      double dx = (xs-x);  // source -> gridpoint x separation
      for(int iy = 0;iy<n;iy++) {
         double y = frame.indexToY(iy);
         double dy = (ys-y);  // charge -> gridpoint y separation
         double r = Math.sqrt(dx*dx+dy*dy);
         realArray[ix][iy] += (r==0) ? 0 : Math.cos(PI2*r)/r;
         imagArray[ix][iy] += (r==0) ? 0 : Math.sin(PI2*r)/r;
      }
   }
}
```

To calculate the real and imaginary components of the phasor, the distance from the source to the grid point is determined in terms of the wavelength λ, and time is determined in terms of the period T. For example, for green light one unit of distance is $\approx 5 \times 10^{-7}$ m and one unit of time is $\approx 1.6 \times 10^{-15}$ s.

The simulation is performed by multiplying the phasors by $e^{-i\omega t}$ in the `doStep` method. Multiplying each phasor by $e^{-i\omega t}$ mixes the phasor's real and imaginary components. We then obtain the physical field from (9.61) by taking the real part:

$$E(\mathbf{r}, t) = \mathrm{Re}[e^{-i\omega t}\mathcal{E}(\mathbf{r})] = \mathrm{Re}[\mathcal{E}] \cos \omega t - \mathrm{Im}[\mathcal{E}] \sin \omega t. \tag{9.62}$$

Listing 9.10 shows the entire `HuygensApp` class. A custom button is used to create sources at the origin. Because the source is an `InteractiveShape`, it can be repositioned using the mouse. The program also implements the `InteractiveMouseHandler` interface to recalculate the phasors when the source is moved. (See Section 5.7 for a discussion of interactive handlers.)

Listing 9.10 The `HuygensApp` class simulates the energy density from one or more point sources.

```
package org.opensourcephysics.sip.ch09;
import java.util.*;
import java.awt.event.*;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.display2d.*;
import org.opensourcephysics.frames.*;

public class HuygensApp extends AbstractSimulation implements
    InteractiveMouseHandler {
   static final double PI2 = Math.PI*2;
   Scalar2DFrame frame = new Scalar2DFrame("x", "y",
                        "Intensity from point sources");
   double time = 0;
```