"points" at random locations in each cell so that the number of points is proportional to the computed intensity at the center of the cell. Suggested parameters are $\lambda = 5000\,\text{Å}$ and $N = 200\,\text{mm}$ for a 1 mm × 3 mm aperture. ■

If the number of sources N becomes large, the summation becomes the Huygens–Fresnel integral. Optics texts discuss how different approximations can be used to evaluate (9.61). For example, if the sources are much closer to each other than they are to the screen (the *far field* condition), we obtain the condition for Fraunhofer diffraction. Otherwise, we obtain Fresnel diffraction.

9.9 ■ FRAUNHOFER DIFFRACTION

The contemporary approach to diffraction is based on Fourier analysis. Consider a plane wave incident on a one-dimensional aperture. Using Huygen's idea that every point within the aperture serves as the source of spherical secondary wavelets, we can approximate the aperture as a linear array of in-phase coherent oscillators. If the spatial extent of the array is small and the distance to the observing screen is large, the wavelet amplitudes arriving at the screen will be essentially equal for wavelets within the aperture and zero for wavelets on the opaque mask. The total field is thus given by the real part of the sum:

$$E(\mathbf{r},t) = E_0 a_1 e^{i(qr_1-\omega t)} + E_0 a_2 e^{i(qr_2-\omega t)} + E_0 a_3 e^{i(qr_2-\omega t)} + \cdots + E_0 a_N e^{i(qr_N-\omega t)}, \tag{9.63}$$

where $a_i = 1$ within the aperture and $a_i = 0$ outside the aperture, and the wavenumber q and the angular frequency ω have their usual meaning. Equation (9.63) can be factored into a complex phasor and time-dependent phase shift:

$$E(\mathbf{r},t) = e^{-i\omega t}\mathcal{E}(\mathbf{r}), \tag{9.64}$$

where

$$\mathcal{E}(\mathbf{r}) = E_0 e^{iqr_1}[1 + a_1 e^{iq(r_2-r_1)} + a_2 e^{iq(r_3-r_1)} + \cdots + a_N e^{iq(r_N-r_1)}]. \tag{9.65}$$

If the grid spacing d is uniform, it follows that the phase difference between adjacent sources is $\delta = qd\sin\theta$, where θ is the angle between the aperture and a point on the screen. We substitute δ into (9.65) and observe that the field can be expressed as an overall phase times a Fourier sum:

$$\mathcal{E}(\mathbf{r}) = E_0 e^{ikqr_1}\left[1 + \sum_{n=1}^{N} a_n e^{in\delta}\right]. \tag{9.66}$$

Equation (9.66) shows that a Fraunhofer (far field) diffraction pattern can be obtained by dividing the aperture using a uniform grid and computing the complex Fourier transform. Because the intensity is proportional to the square of the electric field, the diffraction pattern is the modulus squared of the Fourier transform of the aperture's transmission function. Listing 9.11 uses the one-dimensional fast Fourier transform to compute the Fraunhofer diffraction pattern for a slit.

Listing 9.11 The `FraunhoferApp` program computes the Fraunhofer diffraction pattern from a slit.

```
package org.opensourcephysics.sip.ch09;
import org.opensourcephysics.frames.*;
import org.opensourcephysics.numerics.FFT;

public class FraunhoferApp {
   static final double PI2 = Math.PI*2;
   // Math.log is natural log
   static final double LOG10 = Math.log(10);
   static final double ALPHA = Math.log(1.0e-3)/LOG10; // cutoff value

   public static void main(String[] args) {
      PlotFrame plot = new PlotFrame("x", "intensity",
                          "Fraunhofer diffraction");
      int N = 512;
      FFT fft = new FFT(N);
      double[] cdata = new double[2*N];
      double a = 10; // aperture screen dimension
      double dx = (2*a)/N;
      double x = -a;
      for(int ix = 0;ix<N;ix++) {
         cdata[2*ix] = (Math.abs(x)<0.4) ? 1 : 0; // slit
         cdata[(2*ix)+1] = 0;
         x += dx;
      }
      fft.transform(cdata);
      fft.toNaturalOrder(cdata);
      double max = 0;
      // find the max intensity value
      for(int i = 0;i<N;i++) {
         double real = cdata[2*i];
         double imag = cdata[(2*i)+1];
         max = Math.max(max, (real*real)+(imag*imag));
         plot.append(0, i, (real*real)+(imag*imag));
      }
      plot.setVisible(true);
      // create N by 30 raster plot to show an image
      int[][] data = new int[N][30];
      // compute pixel intensity
      for(int i = 0;i<N;i++) {
         double real = cdata[2*i];
         double imag = cdata[(2*i)+1];
         // log scale increases visibility of weaker fringes
         double logIntensity =
              Math.log(((real*real)+(imag*imag))/max)/LOG10;
         int intensity = (logIntensity<=ALPHA)
                      ? 0 : (int) (254*(1-(logIntensity/ALPHA)));
         for(int j = 0;j<30;j++) {
            data[i][j] = intensity;
         }
      }
      RasterFrame frame =
         new RasterFrame("Fraunhofer Diffraction (log scale)");
      frame.setBWPalette();
      frame.setAll(data); // send the fft data to the raster frame
```