where

$$x_2 - x_1 = \gamma(x'_2 - x'_1) \tag{18.4a}$$

$$y_2 - y_1 = y'_2 - y'_1 \tag{18.4b}$$

$$z_2 - z_1 = z'_2 - z'_1. \tag{18.4c}$$

The change in the x separation is known as the Lorentz–Fitzgerald contraction. If we know the shape of the object in the rest frame, we can determine the shape in the observer's frame by applying an affine transformation (see Chapter 17) that rescales the object's x dimension by γ. Listing 18.1 shows how this transformation is done using a two-dimensional wire frame model of a ring. The `ContractedRing` class defines a ring of unit radius in the object's rest frame using an array of `Point2D` objects. `Point2D` represents a location and can be transformed because it is part of the standard Java 2D API. These points are transformed into the observer's frame and the transformed shape is drawn by connecting the points using line segments.

Listing 18.1 The `ContractedRing` class implements the Lorentz–Fitzgerald contraction of a ring moving in the x direction.

```java
package org.opensourcephysics.sip.ch18;
import java.awt.*;
import java.awt.geom.*;
import org.opensourcephysics.display.*;

public class ContractedRing implements Drawable {
   double vx = 0, time = 0;
   Point2D[] labPoints, pixPoints;

   public ContractedRing(double x0, double y0, double vx,
         int numberOfPoints) {
      labPoints = new Point2D[numberOfPoints];
      pixPoints = new Point2D[numberOfPoints];
      double theta = 0, dtheta = 2*Math.PI/(numberOfPoints-1);
      // unit radius circle
      for(int i = 0;i<numberOfPoints;i++) {
         double x = Math.cos(theta); // x-coordinate
         double y = Math.sin(theta); // y-coordinate
         labPoints[i] = new Point2D.Double(x, y);
         theta += dtheta;
      }
      this.vx = vx;
      // Lorentz-Fitzgerald contraction
      AffineTransform at = AffineTransform.getScaleInstance(
            Math.sqrt(1-vx*vx), 1);
      at.transform(labPoints, 0, labPoints, 0, labPoints.length);
      // translate to initial position
      at = AffineTransform.getTranslateInstance(x0, y0);
      at.transform(labPoints, 0, labPoints, 0, labPoints.length);
   }

   public void setTime(double t) {
      double dt = t-time;
      // convert position to position at new time
      AffineTransform at = AffineTransform.getTranslateInstance(
            vx*dt, 0);
      at.transform(labPoints, 0, labPoints, 0, labPoints.length);
      time = t;
   }

   void drawShape(DrawingPanel panel, Graphics2D g2) {
      // convert from lab coordinates to pixels
      AffineTransform at = panel.getPixelTransform();
      at.transform(labPoints, 0, pixPoints, 0, labPoints.length);
      g2.setColor(Color.RED);
      for(int i = 1, n = labPoints.length;i<n;i++) {
         g2.draw(new Line2D.Double(pixPoints[i-1], pixPoints[i]));
      }
   }

   public void draw(DrawingPanel panel, Graphics g) {
      Graphics2D g2 = (Graphics2D) g;
      drawShape(panel, g2);
   }
}
```

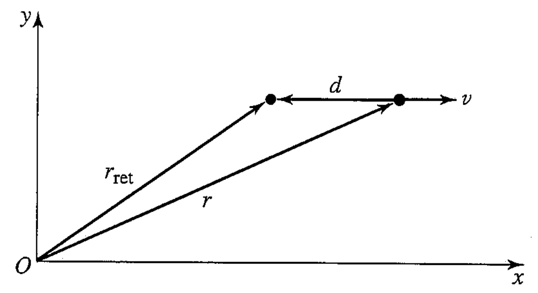

Figure 18.1 The geometry used to derive the retarded position and time for an observer at the origin. The observed point is moving with constant speed v in the x direction.

Exercise 18.1 Lorentz–Fitzgerald contraction
Write a test program that instantiates and displays a `ContractedRing` object. Measure the dimensions of the on-screen object to verify that the Lorentz–Fitzgerald contraction is computed correctly. ■

We now introduce retardation effects. Let $\mathbf{r} = (x, y)$ be the current location of an arbitrary point on the object as shown in Figure 18.1. Because of the finite speed of light, an observer at the origin sees a point moving with a speed v in the x direction not at its current location, but at the previous location

$$\mathbf{r}_{\text{ret}} = (x - \delta, y). \tag{18.5}$$

The x-coordinate is *retarded* by $\delta = v\tau$, where τ is the travel time of light from $\mathbf{r}_{\text{ret}}$ to the observer. The distance from the retarded position to the observer is

$$r_{\text{ret}} = \sqrt{(x - \delta)^2 + y^2}. \tag{18.6}$$