(d) An object can be rotated about its center by first translating the object to the center of rotation, performing the rotation, and then translating the object back to its original position. Implement a method that performs a rotation about the center of the rectangle by invoking a sequence of `translate` and `rotate` methods.

(e) Affine transformations have the property that transformed parallel lines remain parallel. Demonstrate that this property is plausible by transforming a rectangle using arbitrary values for the transformation matrix. ■

To facilitate the creation of simple geometric shapes using world coordinates, the Open Source Physics library defines the `DrawableShape` and `InteractiveShape` classes in the display package. These classes define convenience methods to create common drawable shapes whose (x, y) location is their geometric center. These shapes can later be transformed without having to instantiate new objects. The following code fragment shows how these classes are used.

```
// a circle of radius 3 in world units located at (-1,2)
DrawableShape circle = InteractiveShape.createCircle(-1,2,3);
circle.transform(AffineTransform.getShearInstance(1,2));
frame.addDrawable(circle);
// a rectangle of width 2 and height 1 centered at (3,4)
InteractiveShape rect = InteractiveShape.createRectangle(3,4,2,1);
rect.transform(new AffineTransform(2,1,0,1,0,0));
frame.addDrawable(rect);
```

Because `DrawableShape` and `InteractiveShape` classes are written using the Java 2D API, the objects that they define are fundamentally different from the objects that use the `awt` API introduced in Section 3.3 because Java 2D shapes can be transformed. In addition, the Java 2D API is not restricted to pixel coordinates nor is it restricted to solid single-pixel lines.[1]

Exercise 17.3 Open Source Physics shape classes

The Open Source Physics shape classes can be manipulated using a wide variety of linear algebra-based tools. Modify the `Affine2DApp` program so that it instantiates and transforms a rectangular `DrawableShape` into a trapezoid. Test your program by repeating Exercise 17.2. ■

17.2 ■ THREE-DIMENSIONAL TRANSFORMATIONS

There are several available APIs for three-dimensional visualizations using Java. Although Sun has developed the Java 3D package, this package is currently not included in the standard Java runtime environment. The `gl4java` and `jogl` libraries are also popular because they are based on the Open GL language. Because 3D graphics libraries are in a state of active development and because we want a three-dimensional visualization framework designed for physics simulations, we have developed a three-dimensional visualization framework that relies only on the standard Java API. This section describes the mathematics that forms the basis of all three-dimensional libraries.

[1]See Chapter 4 in the *Open Source Physics: A User's Guide with Examples* for a more complete discussion of the `DrawableShape` and `InteractiveShape` classes.

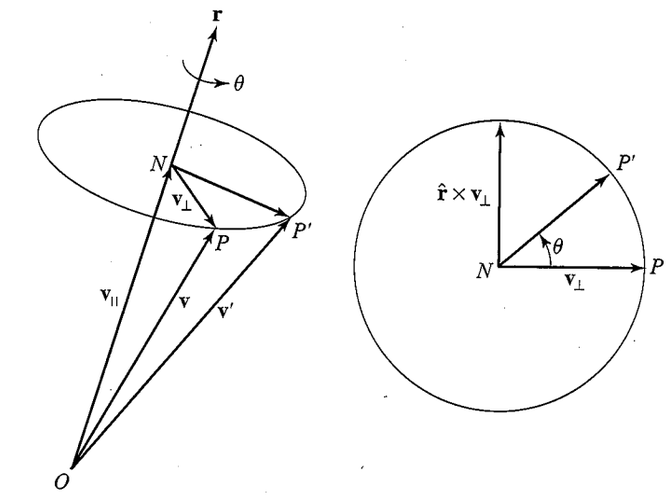

Figure 17.2 A rotation of the vector $\mathbf{v}$ about an axis $\hat{\mathbf{r}}$ to produce the vector $\mathbf{v}'$ can be decomposed into parallel $\mathbf{v}_\parallel$ and perpendicular $\mathbf{v}_\perp$ components.

The simplest rotation is a rotation about one of the coordinate axes with the center of rotation at the origin. This transformation can be written using a 3×3 matrix acting on the coordinates (x, y, z). For example, a rotation about the z-axis can be written as

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathcal{R}_z \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{17.8}$$

The extension of homogeneous coordinates to three dimensions is straightforward. We add a w-coordinate to the spatial coordinates to create a homogenous point $(x, y, z, 1)$. This point is transformed as

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \mathcal{R}_z & \mathbf{d}^T \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{17.9}$$

Although rotations about one of the coordinate axes are easy to derive and can be combined using the rules of linear algebra to produce an arbitrary orientation, the general case of rotation about the origin by an angle θ around an arbitrary axis $\hat{\mathbf{r}}$ can be constructed directly. The strategy is to decompose the vector $\mathbf{v}$ into components that are parallel and perpendicular to the direction $\hat{\mathbf{r}}$ as shown in Figure 17.2. The parallel part $\mathbf{v}_\parallel$ does not change, while the perpendicular part $\mathbf{v}_\perp$ is a two-dimensional rotation in a plane perpendicular to $\hat{\mathbf{r}}$. The parallel part is the projection of $\hat{\mathbf{r}}$ onto $\mathbf{v}$,

$$\mathbf{v}_\parallel = (\mathbf{v} \cdot \hat{\mathbf{r}})\hat{\mathbf{r}}, \tag{17.10}$$