and the perpendicular part is what remains of $\hat{\mathbf{v}}$ after we subtract the parallel part:

$$\mathbf{v}_\perp = \mathbf{v} - (\mathbf{v} \cdot \hat{\mathbf{r}})\hat{\mathbf{r}}. \tag{17.11}$$

To calculate the rotation of $\mathbf{v}_\perp$, we need two perpendicular basis vectors in the plane of rotation. If we use $\mathbf{v}_\perp$ as the first basis vector, then we can take the cross product with $\hat{\mathbf{r}}$ to produce a vector $\mathbf{w}$ that is guaranteed to be perpendicular to $\mathbf{v}_\perp$ and $\hat{\mathbf{r}}$:

$$\mathbf{w} = \hat{\mathbf{r}} \times \mathbf{v}_\perp = \hat{\mathbf{r}} \times \mathbf{v}. \tag{17.12}$$

The rotation of $\mathbf{v}_\perp$ is now calculated in terms of this new basis:

$$\mathbf{v}' = \mathcal{R}(\mathbf{v}_\perp) = \cos\theta\, \mathbf{v}_\perp + \sin\theta\, \mathbf{w}. \tag{17.13}$$

The final result is the sum of this rotated vector and the parallel part that does not change:

$$\begin{aligned}
\mathcal{R}(\mathbf{v}) &= \mathcal{R}(\mathbf{v}_\perp) + \mathbf{v}_\parallel && (17.14a)\\
&= \cos\theta\, \mathbf{v}_\perp + \sin\theta\, \mathbf{w} + \mathbf{v}_\parallel && (17.14b)\\
&= \cos\theta\, [\mathbf{v} - (\mathbf{v} \cdot \hat{\mathbf{r}})\hat{\mathbf{r}}] + \sin\theta\, (\hat{\mathbf{r}} \times \mathbf{v}) + (\mathbf{v} \cdot \hat{\mathbf{r}})\hat{\mathbf{r}}, && (17.14c)
\end{aligned}$$

or

$$\mathcal{R}(\mathbf{v}) = [1 - \cos\theta](\mathbf{v} \cdot \hat{\mathbf{r}})\hat{\mathbf{r}} + \sin\theta\, (\hat{\mathbf{r}} \times \mathbf{v}) + \cos\theta\, \mathbf{v}. \tag{17.15}$$

Equation (17.15) is known as the *Rodrigues formula* and provides a way of constructing rotation matrices in terms of the direction of the axis of rotation $\hat{\mathbf{r}} = (r_x, r_y, r_z)$, the cosine of the rotation angle $c = \cos\theta$, and the sine of the rotation angle $s = \sin\theta$. If we expand the vector products in (17.15), we obtain the matrix

$$\mathcal{R} = \begin{bmatrix} tr_xr_x + c & tr_xr_y - sr_z & tr_xr_z + sr_y \\ tr_xr_y + sr_z & tr_yr_y + c & tr_yr_z - sr_x \\ tr_xr_z - sr_y & tr_yr_z + sr_x & tr_zr_z + c \end{bmatrix}, \tag{17.16}$$

where $t = 1 - \cos\theta$. Homogeneous coordinates are transformed using

$$\begin{bmatrix} \mathcal{R} & \mathbf{0}^T \\ \mathbf{0} & 1 \end{bmatrix}, \tag{17.17}$$

where the $\mathcal{R}$ submatrix is given in (17.16).

The `Rotation3D` class constructor (see Listing 17.2) computes the rotation matrix. The `direct` method uses this matrix to transform a point. Note that the point passed to this method as an argument is copied into a temporary vector and that the point's coordinates are then changed. You will define an `inverse` method that reverses this operation in Exercise 17.5.

Listing 17.2 The `Rotation3D` class implements three-dimensional rotations using a matrix representation.

```
package org.opensourcephysics.sip.ch17;
public class Rotation3D {
   // transformation matrix
   private double[][] mat = new double[4][4];

   public Rotation3D(double theta, double[] axis) {
      double norm = Math.sqrt(axis[0]*axis[0]+axis[1]*axis[1]
           +axis[2]*axis[2]);
      double x = axis[0]/norm, y = axis[1]/norm,
           z = axis[2]/norm;
      double c = Math.cos(theta), s = Math.sin(theta);
      double t = 1-c;
      // matrix elements not listed are zero
      mat[0][0] = t*x*x+c;
      mat[0][1] = t*x*y-s*z;
      mat[0][2] = t*x*y+s*y;
      mat[1][0] = t*x*y+s*z;
      mat[1][1] = t*y*y+c;
      mat[1][2] = t*y*z-s*x;
      mat[2][0] = t*x*z-s*y;
      mat[2][1] = t*y*z+s*x;
      mat[2][2] = t*z*z+c;
      mat[3][3] = 1;
   }

   public void direct(double[] point) {
      int n = point.length;
      double[] pt = new double[n];
      System.arraycopy(point, 0, pt, 0, n);
      for(int i = 0;i<n;i++) {
         point[i] = 0;
         for(int j = 0;j<n;j++) {
            point[i] += mat[i][j]*pt[j];
         }
      }
   }
}
```


Exercise 17.4 Rodrigues formula
Show that a rotation about the z-axis is consistent with (17.15) and (17.16). That is, define the direction of rotation to be $\hat{\mathbf{r}} = (0, 0, 1)$ and show that both formulas give the same result and that this result is consistent with a two-dimensional rotation in the xy-plane. Write a test program to test the `Rotation3D` class. ■

Exercise 17.5 Inverse matrix
What is the inverse matrix of (17.16)? Hint: What happens physically if you change the sign of the rotation angle. Is the matrix orthogonal? Add an inverse method to `Rotation3D` and write a test program to test the `inverse` method. Show that the original vector is recovered if the `inverse` and `direct` methods are applied in succession. ■

A projection transforms an object in a coordinate system of dimension d into another object in a coordinate system less than d. The simplest projection is an *orthographic parallel projection*, which maps an object onto a plane perpendicular to a coordinate axis. For example, if we choose to project along the z-axis, the point (x, y, z) is mapped to the point (x, y) by dropping the third coordinate. A line is projected by projecting the endpoints and then connecting the projected values. A sphere with radius R is displayed by projecting