

Lecture 3:

Coordinate Systems and Transformations

Topics:

1. Coordinate systems and frames
2. Change of frames
3. Affine transformations
4. Rotation, translation, scaling, and shear
5. Rotation about an arbitrary axis

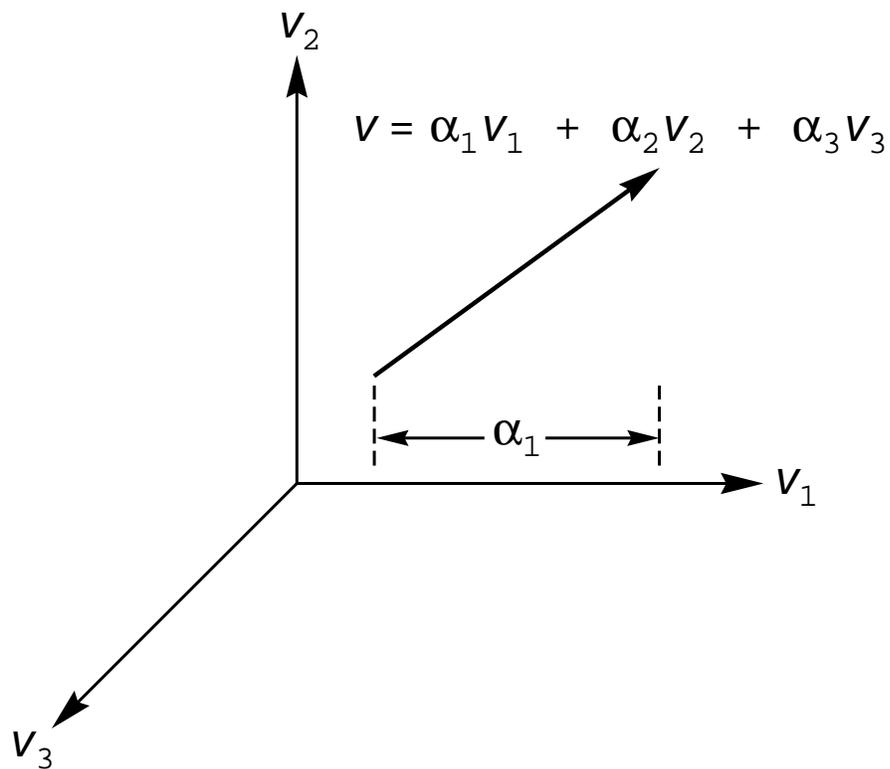
Chapter 4, Sections 4.3, 4.5, 4.6, 4.7, 4.8, 4.9.

Coordinate systems and frames

Recall that a vector $v \in \mathbb{R}^3$ can be represented as a linear combination of three linearly independent basis vectors v_1, v_2, v_3 ,

$$v = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3.$$

The scalars $\alpha_1, \alpha_2, \alpha_3$ are the **coordinates** of v . We typically choose $v_1 = (1, 0, 0), v_2 = (0, 1, 0), v_3 = (0, 0, 1)$.



Suppose we want to (linearly) change the basis vectors v_1, v_2, v_3 to u_1, u_2, u_3 . We express the new basis vectors as combinations of the old ones,

$$u_1 = a_{11}v_1 + a_{12}v_2 + a_{13}v_3,$$

$$u_2 = a_{21}v_1 + a_{22}v_2 + a_{23}v_3,$$

$$u_3 = a_{31}v_1 + a_{32}v_2 + a_{33}v_3,$$

and thus obtain a 3×3 ‘change of basis’ matrix

$$M = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

If the two representations of a given vector v are

$$v = \mathbf{a}^T \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}, \quad \text{and} \quad v = \mathbf{b}^T \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix},$$

where $\mathbf{a} = (\alpha_1 \ \alpha_2 \ \alpha_3)^T$ and $\mathbf{b} = (\beta_1 \ \beta_2 \ \beta_3)^T$, then

$$\mathbf{a}^T \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = v = \mathbf{b}^T \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \mathbf{b}^T M \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix},$$

which implies that

$$\mathbf{a} = M^T \mathbf{b} \quad \text{and} \quad \mathbf{b} = (M^T)^{-1} \mathbf{a}.$$

This 3D coordinate system is not, however, rich enough for use in computer graphics. Though the matrix M could be used to rotate and scale vectors, it cannot deal with **points**, and we want to be able to **translate** points (and objects).

In fact an arbitrary affine transformation can be achieved by multiplication by a 3×3 matrix and shift by a vector. However, in computer graphics we prefer to use **frames** to achieve the same thing.

A **frame** is a richer coordinate system in which we have a reference point P_0 in addition to three linearly independent basis vectors v_1, v_2, v_3 , and we represent vectors v and points P , differently, as

$$\begin{aligned} v &= \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3, \\ P &= P_0 + \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3. \end{aligned}$$

We can use vector and matrix notation and re-express the vector v and point P as

$$v = (\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad 0) \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{pmatrix}, \quad \text{and} \quad P = (\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad 1) \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{pmatrix}.$$

The coefficients $\alpha_1, \alpha_2, \alpha_3, 0$ and $\alpha_1, \alpha_2, \alpha_3, 1$ are the **homogeneous coordinates** of v and P respectively.

Change of frames

Suppose we want to change from the frame (v_1, v_2, v_3, P_0) to a new frame (u_1, u_2, u_3, Q_0) . We express the new basis vectors and reference point in terms of the old ones,

$$u_1 = a_{11}v_1 + a_{12}v_2 + a_{13}v_3,$$

$$u_2 = a_{21}v_1 + a_{22}v_2 + a_{23}v_3,$$

$$u_3 = a_{31}v_1 + a_{32}v_2 + a_{33}v_3,$$

$$Q_0 = a_{41}v_1 + a_{42}v_2 + a_{43}v_3 + P_0,$$

and thus obtain a 4×4 matrix

$$M = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & 1 \end{pmatrix}.$$

Similar to 3D vector coordinates, we suppose now that \mathbf{a} and \mathbf{b} are the homogeneous representations of the same point or vector with respect to the two frames. Then

$$\mathbf{a}^T \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{pmatrix} = \mathbf{b}^T \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{pmatrix} = \mathbf{b}^T M \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{pmatrix},$$

which implies that

$$\mathbf{a} = M^T \mathbf{b} \quad \text{and} \quad \mathbf{b} = (M^T)^{-1} \mathbf{a}.$$

Affine transformations

The transposed matrix

$$M^T = \begin{pmatrix} a_{11} & a_{21} & a_{31} & a_{41} \\ a_{12} & a_{22} & a_{32} & a_{42} \\ a_{13} & a_{23} & a_{33} & a_{43} \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

simply represents an arbitrary **affine transformation**, having 12 degrees of freedom. These degrees of freedom can be viewed as the nine elements of a 3×3 matrix plus the three components of a vector shift.

The most important affine transformations are **rotations**, **scalings**, and **translations**, and in fact all affine transformations can be expressed as combinations of these three.

Affine transformations preserve line segments. If a line segment

$$P(\alpha) = (1 - \alpha)P_0 + \alpha P_1$$

is expressed in homogeneous coordinates as

$$\mathbf{p}(\alpha) = (1 - \alpha)\mathbf{p}_0 + \alpha\mathbf{p}_1,$$

with respect to some frame, then an affine transformation matrix M sends the line segment P into the new one,

$$M\mathbf{p}(\alpha) = (1 - \alpha)M\mathbf{p}_0 + \alpha M\mathbf{p}_1.$$

Similarly, affine transformations map triangles to triangles and tetrahedra to tetrahedra. Thus many objects in OpenGL can be transformed by transforming their vertices only.

Rotation, translation, scaling, and shear

Translation is an operation that displaces points by a fixed distance in a given direction. If the displacement vector is d then the point P will be moved to

$$P' = P + d.$$

We can write this equation in homogeneous coordinates as

$$\mathbf{p}' = \mathbf{p} + \mathbf{d},$$

where

$$\mathbf{p} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}, \quad \mathbf{p}' = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}, \quad \mathbf{d} = \begin{pmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \\ 0 \end{pmatrix}.$$

so that

$$x' = x + \alpha_x, \quad y' = y + \alpha_y, \quad z' = z + \alpha_z.$$

So the transformation matrix T which gives $\mathbf{p}' = T\mathbf{p}$ is clearly

$$T = T(\alpha_x, \alpha_y, \alpha_z) = \begin{pmatrix} 1 & 0 & 0 & \alpha_x \\ 0 & 1 & 0 & \alpha_y \\ 0 & 0 & 1 & \alpha_z \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

called the **translation matrix**. One can check that the inverse is

$$T^{-1}(\alpha_x, \alpha_y, \alpha_z) = T(-\alpha_x, -\alpha_y, -\alpha_z).$$

Rotation depends on an axis of rotation and the angle turned through. Consider first rotation in the plane, about the origin. If a point (x, y) with coordinates

$$x = \rho \cos \phi, \quad y = \rho \sin \phi,$$

is rotated through an angle θ , then the new position is (x', y') , where

$$x' = \rho \cos(\phi + \theta), \quad y' = \rho \sin(\phi + \theta).$$

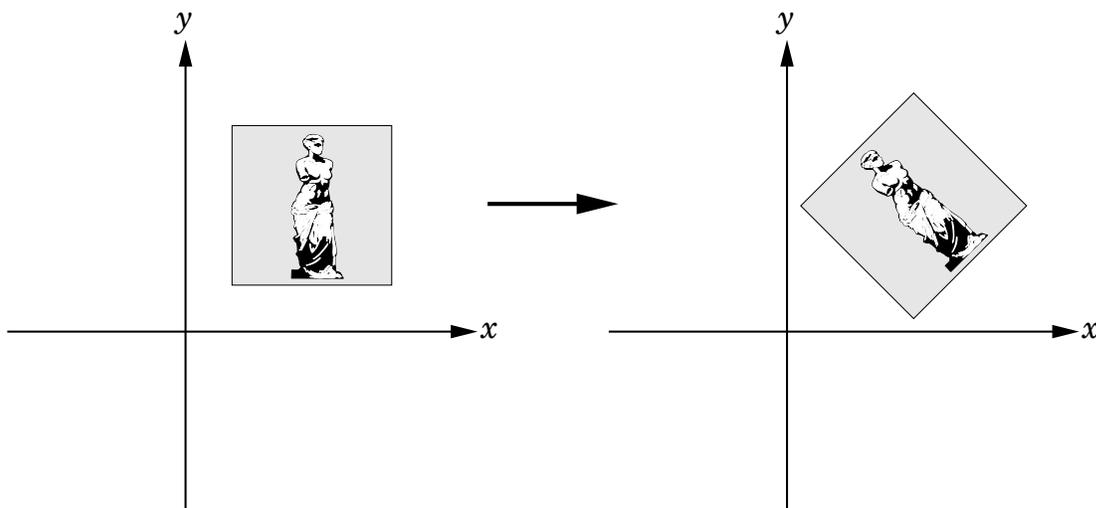
Expanding these latter expressions, we find

$$x' = x \cos \theta - y \sin \theta,$$

$$y' = x \sin \theta + y \cos \theta,$$

or

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$



Thus the three rotation matrices corresponding to rotation about the z , x , and y axes in \mathbb{R}^3 are:

$$R_z = R_z(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$R_x = R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$R_y = R_y(\theta) = \begin{pmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

All three give positive rotations for positive θ with respect to the right hand rule for the axes x, y, z . If $R = R(\theta)$ denotes any of these matrices, its inverse is clearly

$$R^{-1}(\theta) = R(-\theta) = R^T(\theta).$$

Translations and rotations are examples of **solid-body transformations**: transformations which do not alter the size or shape of an object.

Scaling can be applied in any of the three axes independently. If we send a point (x, y, z) to the new point

$$x' = \beta_x x, \quad y' = \beta_y y, \quad z' = \beta_z z,$$

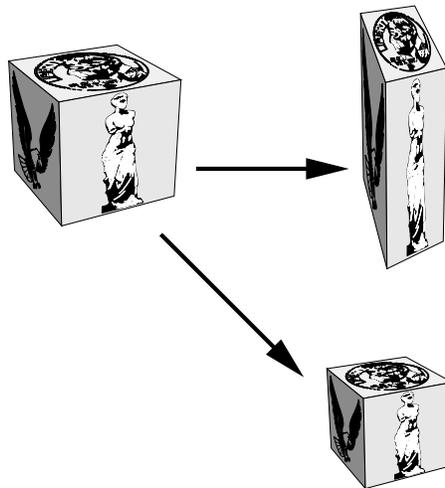
then the corresponding matrix becomes

$$S = S(\beta_x, \beta_y, \beta_z) = \begin{pmatrix} \beta_x & 0 & 0 & 0 \\ 0 & \beta_y & 0 & 0 \\ 0 & 0 & \beta_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

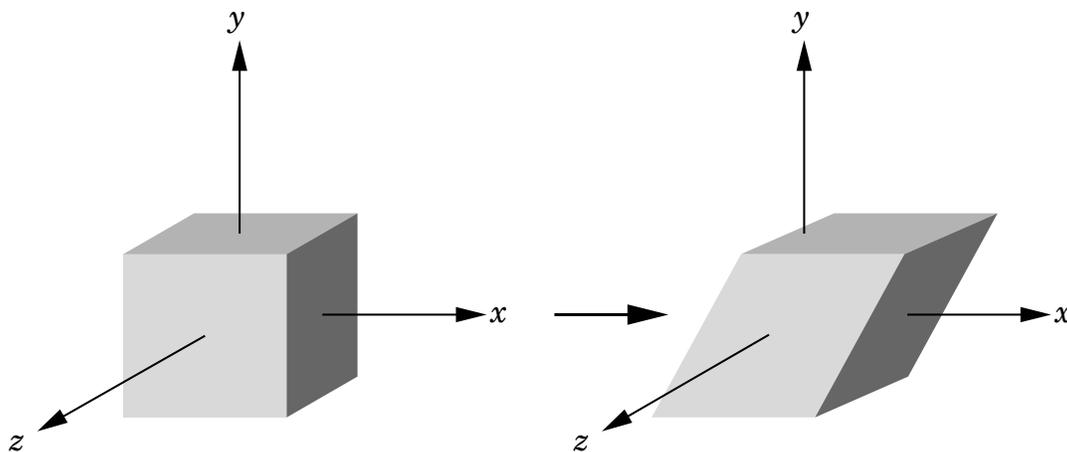
with inverse

$$S^{-1}(\beta_x, \beta_y, \beta_z) = S(1/\beta_x, 1/\beta_y, 1/\beta_z).$$

If the scaling factors $\beta_x, \beta_y, \beta_z$ are equal then the scaling is **uniform**: objects retain their shape but alter their size. Otherwise the scaling is **non-uniform** and the object is deformed.



Shears can be constructed from translations, rotations, and scalings, but are sometimes of independent interest.



A shear in the x direction is defined by

$$x' = x + (\cot \theta)y, \quad y' = y, \quad z' = z,$$

for some scalar a . The corresponding shearing matrix is therefore

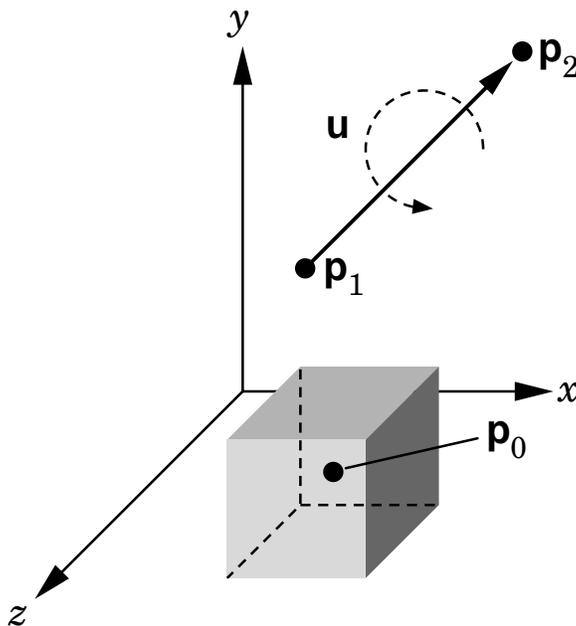
$$H_x(\theta) = \begin{pmatrix} 1 & \cot \theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

with inverse

$$H_x^{-1}(\theta) = H_x(-\theta).$$

Rotation about an arbitrary axis

How do we find the matrix which rotates an object about an arbitrary point p_0 and around a direction $u = p_2 - p_1$, through an angle θ ?



The answer is to concatenate some of the matrices we have already developed. We will assume that u has length 1. The first step is to use translation to reduce the problem to that of rotation about the origin:

$$M = T(p_0) R T(-p_0).$$

To find the rotation matrix R for rotation around the vector u , we first align u with the z axis using two rotations θ_x and θ_y . Then we can apply a rotation of θ around the z -axis and afterwards undo the alignments, thus

$$R = R_x(-\theta_x) R_y(-\theta_y) R_z(\theta) R_y(\theta_y) R_x(\theta_x).$$

It remains to calculate θ_x and θ_y from $u = (\alpha_x, \alpha_y, \alpha_z)$. The first rotation $R_x(\theta_x)$ will rotate the vector u around the x axis until it lies in the $y = 0$ plane. Using simple trigonometry, we find that

$$\cos \theta_x = \alpha_z/d, \quad \sin \theta_x = \alpha_y/d,$$

where $d = \sqrt{\alpha_y^2 + \alpha_z^2}$, so, without needing θ_x explicitly, we find

$$R_x(\theta_x) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \alpha_z/d & -\alpha_y/d & 0 \\ 0 & \alpha_y/d & \alpha_z/d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

In the second alignment we find

$$\cos \theta_y = d, \quad \sin \theta_y = -\alpha_x,$$

and so

$$R_y(\theta_y) = \begin{pmatrix} d & 0 & \alpha_x & 0 \\ 0 & 1 & 0 & 0 \\ -\alpha_x & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Example in OpenGL

The following OpenGL sequence sets the model-view matrix to represent a 45-degree rotation about the line through the origin and the point $(1, 2, 3)$ with a fixed point of $(4, 5, 6)$:

```
void myinit(void)
{
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(4.0, 5.0, 6.0);
    glRotatef(45.0, 1.0, 2.0, 3.0);
    glTranslatef(-4.0, -5.0, -6.0);
}
```

OpenGL concatenates the three matrices into the single matrix

$$C = T(4, 5, 6) R(45, 1, 2, 3) T(-4, -5, -6).$$

Each vertex p specified after this code will be multiplied by C to yield q ,

$$q = Cp.$$

Spinning the Cube

The following program rotates a cube, using the three buttons of the mouse.

There are three callbacks:

```
glutDisplayFunc(display);
glutIdleFunc(spincube);
glutMouseFunc(mouse);
```

The display callback sets the model-view matrix with three angles determined by the mouse callback, and then draws the cube (see Section 4.4 of the book).

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT    GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);
    colorcube();
    glutSwapBuffers();
}
```

The mouse callback selects the axis of rotation.

```
void mouse(int btn, int state, int x, int y)
{
    if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
    if(btn == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
    if(btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
}
```

The idle callback increments the angle of the chosen axis by 2 degrees.

```
void spincube()
{
    theta[axis] += 2.0;
    if(theta[axis] >= 360.0) theta[axis] -= 360.0;
    glutPostRedisplay();
}
```

A little bit about quaternions

Quaternions offer an alternative way of describing rotations, and have been used in hardware implementations.

In the complex plane, rotation through an angle ϕ can be expressed as multiplication by the complex number

$$e^{i\phi} = \cos \phi + i \sin \phi.$$

This rotates a given point $re^{i\theta}$ in the complex plane to the new point

$$re^{i\theta} e^{i\phi} = re^{i(\theta+\phi)}.$$

Analogously, quaternions can be used to elegantly describe rotations in three dimensions. A **quaternion** consists of a scalar and a vector,

$$a = (q_0, q_1, q_2, q_3) = (q_0, \mathbf{q}).$$

We can write the vector part \mathbf{q} as

$$\mathbf{q} = q_1 \mathbf{i} + q_2 \mathbf{j} + q_3 \mathbf{k},$$

where \mathbf{i} , \mathbf{j} , \mathbf{k} play a similar role to that of the unit vectors in \mathbb{R}^3 , and obey the properties

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1,$$

and

$$\mathbf{ij} = \mathbf{k} = -\mathbf{ji}, \quad \mathbf{jk} = \mathbf{i} = -\mathbf{kj}, \quad \mathbf{ki} = \mathbf{j} = -\mathbf{ik}.$$

These properties imply that the sum and multiple of two quaternions $a = (q_0, \mathbf{q})$ and $b = (p_0, \mathbf{p})$ are

$$a + b = (q_0 + p_0, \mathbf{q} + \mathbf{p}),$$

and

$$ab = (q_0p_0 - \mathbf{q} \cdot \mathbf{p}, q_0\mathbf{p} + p_0\mathbf{q} + \mathbf{q} \times \mathbf{p}).$$

The magnitude and inverse of a are

$$|a| = \sqrt{q_0^2 + \mathbf{q} \cdot \mathbf{q}}, \quad a^{-1} = \frac{1}{|a|}(q_0, -\mathbf{q}).$$

Rotation. Suppose now that \mathbf{v} is a unit vector in \mathbb{R}^3 , and let \mathbf{p} be an arbitrary point. Denote by \mathbf{p}' the rotation of \mathbf{p} about the axis \mathbf{v} , placed at the origin. To find \mathbf{p}' using quaternions, we let r be the quaternion

$$r = (\cos(\theta/2), \sin(\theta/2)\mathbf{v}),$$

of unit length, whose inverse is clearly

$$r^{-1} = (\cos(\theta/2), -\sin(\theta/2)\mathbf{v}).$$

Then if p and p' are the quaternions

$$p = (0, \mathbf{p}), \quad p' = (0, \mathbf{p}'),$$

we claim that

$$p' = r^{-1}pr.$$

Let's check that this is correct! Following the rule for multiplication, a little computation shows that \mathbf{p}' does indeed have the form $(0, \mathbf{p}')$, and that

$$\mathbf{p}' = \cos^2 \frac{\theta}{2} \mathbf{p} + \sin^2 \frac{\theta}{2} (\mathbf{v} \cdot \mathbf{p}) \mathbf{v} + 2 \sin \frac{\theta}{2} \cos \frac{\theta}{2} (\mathbf{v} \times \mathbf{p}) + \sin^2 \frac{\theta}{2} \mathbf{v} \times (\mathbf{v} \times \mathbf{p}).$$

But using the identity

$$\mathbf{p} = (\mathbf{v} \cdot \mathbf{p}) \mathbf{v} - \mathbf{v} \times (\mathbf{v} \times \mathbf{p}), \tag{1}$$

and the multiple-angle formulas for cos and sin, this simplifies to

$$\mathbf{p}' = (\mathbf{v} \cdot \mathbf{p}) \mathbf{v} + \cos \theta (\mathbf{v} \times \mathbf{p}) \times \mathbf{v} + \sin \theta (\mathbf{v} \times \mathbf{p}).$$

This formula can easily be verified from a geometric interpretation. It is a linear combination of three orthogonal vectors. The first term $(\mathbf{v} \cdot \mathbf{p}) \mathbf{v}$ is the projection of \mathbf{p} onto the axis of rotation, and the second and third terms describe a rotation in the plane perpendicular to \mathbf{v} .

Notice also that the formula clearly represents an affine transformation; all terms are linear in \mathbf{p} . The formula can be used to find the coefficients of the corresponding transformation matrix directly.