Quaternion to Rotation Matrix

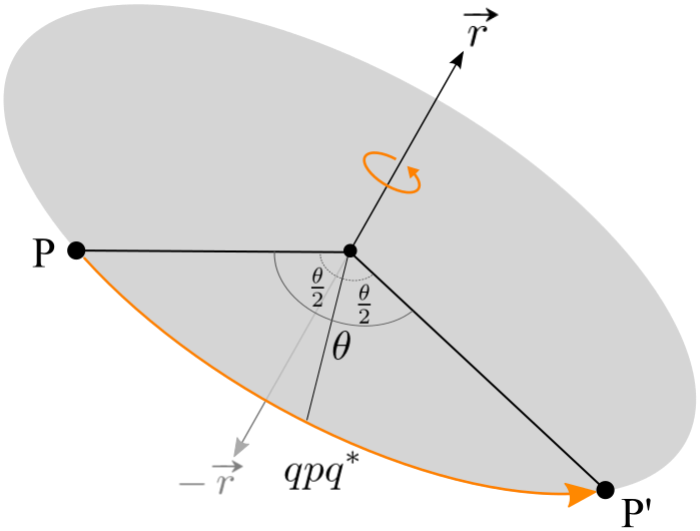
OpenGL API provides **glRotatef()** to rotate a 3D vector about an arbitrary axis using Rodrigues' rotation formula. Here explains how to convert Rodrigues' rotation formula to 4x4 matrix.

In addition, Quaternion can be also used for rotating a vector around an axis in 3D space. Multiplying Quaternions implies a rotation of a vector in 3D space and it is commonly used in 3D computer graphics algorithms because it is simpler and cheaper than the matrix multiplication. This page is focused on how to convert a Quaternion rotation to a 4x4 rotation matrix form using Quaternion algebra. Note that OpenGL accepts only 4x4 matrix to transform vertices.

$$M_R = \begin{pmatrix} 1-2y^2-2z^2 & 2xy-2sz & 2xz+2sy & 0 \\ 2xy+2sz & 1-2x^2-2z^2 & 2yz-2sx & 0 \\ 2xz-2sy & 2yz+2sx & 1-2x^2-2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{where } q = s + ix + jy + kz, \quad |q| = 1$$

Overview



*qpq** means rotating P to P' along R axis by θ

Suppose there is a 3D vector, $\vec{p} = (x_p, y_p, z_p)$ and rotate it along an arbitrary axis (unit vector) $\vec{r} = (x_r, y_r, z_r)$ by angle θ.

The vector $\vec{p}$ can be converted as Quaternion form;
(We only use x, y and z components for $\vec{p}$.)

$$p = 0 + ix_p + jy_p + kz_p$$

And, the Quaternion representation for the rotation axis vector $\vec{r}$ and the half rotation angle $\frac{\theta}{2}$ can be written as

$$\begin{aligned} q &= \cos\tfrac{\theta}{2} + \sin\tfrac{\theta}{2}\vec{r} \\ &= \cos\tfrac{\theta}{2} + i\sin\tfrac{\theta}{2}x_r + j\sin\tfrac{\theta}{2}y_r + k\sin\tfrac{\theta}{2}z_r \end{aligned}$$

(*Note that we only use half angle to define the rotation quaternion* ***q*** *because we are going to multiply* ***q*** *twice.*)

To rotate the quaternion ***p***, we simply multiply the quaternion ***q***. However, the length of ***p*** is changed during multiplication, so we multiply ***q**** (conjugate of *q*) again at the back of ***p*** in order to cancel out the length changes. Since we multiply ***q*** twice, front and back, we only need a half rotation angle for ***q***. Therefore, the final equation for the rotation in Quaternion becomes qpq^*.

This special double multiplication is called "conjugation by *q*". Here explains why this double multiplication is required and the proof of the length conservation.

Quaternion to Matrix

In this section, we calculate qpq^* from arbitrary Quaternions, ***p*** and ***q***. Let's say ***p*** and ***q*** are defined as below, and ***q*** is normalized ($|q| = 1$):

$$p = s_p + ix_p + jy_p + kz_p$$

$$q = s + ix + jy + kz$$
$$q^* = s - ix - jy - kz$$

For a typical usage, rotating a 3D vector, we can set the scalar component to 0 for ***p***. It makes the multiplication simpler, but here we use a generic 4-component quaternion for ***p***.

At first, we multiply ***p*** by ***q*** (applying ***q*** on ***p***):

$$
\begin{aligned}
qp &= (s + ix + jy + kz)(s_p + ix_p + jy_p + kz_p) \\
&= ss_p + isx_p + jsy_p + ksz_p + \\
&\quad ixs_p + iixx_p + ijxy_p + ikxz_p + \\
&\quad jys_p + jiyx_p + jjyy_p + jkyz_p + \\
&\quad kzs_p + kizx_p + kjzy_p + kkzz_p \\
&= ss_p + isx_p + jsy_p + ksz_p + \\
&\quad ixs_p - xx_p + kxy_p - jxz_p + \\
&\quad jys_p - kyx_p - yy_p + iyz_p + \\
&\quad kzs_p + jzx_p - izy_p - zz_p \\
&= ss_p - xx_p - yy_p - zz_p + \\
&\quad i(sx_p + xs_p + yz_p - zy_p) + \\
&\quad j(sy_p - xz_p + ys_p + zx_p) + \\
&\quad k(sz_p + xy_p - yx_p + zs_p)
\end{aligned}
$$

The above equation is simplified by Quaternion quadrantal (perpendicular) vectors property.

$$
\begin{aligned}
ii = jj = kk = -1 \\
ij = k, \quad ji = -k \\
jk = i, \quad kj = -i \\
ki = j, \quad ik = -j
\end{aligned}
$$

Let's set each component of qp to $\mathbb{S}$, $\mathbb{X}$, $\mathbb{Y}$and $\mathbb{Z}$ respectively.

$$
\begin{aligned}
\mathbb{S} &= ss_p - xx_p - yy_p - zz_p \\
\mathbb{X} &= sx_p + xs_p + yz_p - zy_p \\
\mathbb{Y} &= sy_p - xz_p + ys_p + zx_p \\
\mathbb{Z} &= sz_p + xy_p - yx_p + zs_p
\end{aligned}
$$

Then, multiply the conjugate of ***q*** at the back of the previous multiplication.

$$
\begin{aligned}
qpq^* &= (\mathbb{S} + i\mathbb{X} + j\mathbb{Y} + k\mathbb{Z})(s - ix - jy - kz) \\
&= \mathbb{S}s - i\mathbb{S}x - j\mathbb{S}y - k\mathbb{S}z + \\
&\quad i\mathbb{X}s - ii\mathbb{X}x - ij\mathbb{X}y - ik\mathbb{X}z + \\
&\quad j\mathbb{Y}s - ji\mathbb{Y}x - jj\mathbb{Y}y - jk\mathbb{Y}z + \\
&\quad k\mathbb{Z}s - ki\mathbb{Z}x - kj\mathbb{Z}y - kk\mathbb{Z}z \\
&= \mathbb{S}s - i\mathbb{S}x - j\mathbb{S}y - k\mathbb{S}z + \\
&\quad i\mathbb{X}s + \mathbb{X}x - k\mathbb{X}y + j\mathbb{X}z + \\
&\quad j\mathbb{Y}s + k\mathbb{Y}x + \mathbb{Y}y - i\mathbb{Y}z + \\
&\quad k\mathbb{Z}s - j\mathbb{Z}x + i\mathbb{Z}y + \mathbb{Z}z \\
&= \mathbb{S}s + \mathbb{X}x + \mathbb{Y}y + \mathbb{Z}z + \\
&\quad i(-\mathbb{S}x + \mathbb{X}s - \mathbb{Y}z + \mathbb{Z}y) + \\
&\quad j(-\mathbb{S}y + \mathbb{X}z + \mathbb{Y}s - \mathbb{Z}x) + \\
&\quad k(-\mathbb{S}z - \mathbb{X}y + \mathbb{Y}x + \mathbb{Z}s)
\end{aligned}
$$

Since the equation becomes too long, we compute each term and component independently and combine them later. This page provides the complete computation of qpq^*. You may skip to the final result of qpq^* below.

$$\begin{aligned}
\mathbb{S}s &= (ss_p - xx_p - yy_p - zz_p)s \\
&= s^2 s_p - sxx_p - syy_p - szz_p \\
\mathbb{S}x &= (ss_p - xx_p - yy_p - zz_p)x \\
&= sxs_p - x^2 x_p - xyy_p - xzz_p \\
\mathbb{S}y &= (ss_p - xx_p - yy_p - zz_p)y \\
&= sys_p - xyx_p - y^2 y_p - yzz_p \\
\mathbb{S}z &= (ss_p - xx_p - yy_p - zz_p)z \\
&= szs_p - xzx_p - yzy_p - z^2 z_p
\end{aligned}$$

$$\begin{aligned}
\mathbb{X}s &= (sx_p + xs_p + yz_p - zy_p)s \\
&= s^2 x_p + sxs_p + syz_p - szy_p \\
&= sxs_p + s^2 x_p - szy_p + syz_p \\
\mathbb{X}x &= (sx_p + xs_p + yz_p - zy_p)x \\
&= sxx_p + x^2 s_p + xyz_p - xzy_p \\
&= x^2 s_p + sxx_p - xzy_p + xyz_p \\
\mathbb{X}y &= (sx_p + xs_p + yz_p - zy_p)y \\
&= syx_p + xys_p + y^2 z_p - yzy_p \\
&= xys_p + syx_p - yzy_p + y^2 z_p \\
\mathbb{X}z &= (sx_p + xs_p + yz_p - zy_p)z \\
&= szx_p + xzs_p + yzz_p - z^2 y_p \\
&= xzs_p + szx_p - z^2 y_p + yzz_p
\end{aligned}$$

$$
\begin{aligned}
\mathbb{Y}s &= (sy_p - xz_p + ys_p + zx_p)s \\
&= s^2y_p - sxz_p + sys_p + szx_p \\
&= sys_p + szx_p + s^2y_p - sxz_p
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{Y}x &= (sy_p - xz_p + ys_p + zx_p)x \\
&= sxy_p - x^2z_p + xys_p + xzx_p \\
&= xys_p + xzx_p + sxy_p - x^2z_p
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{Y}y &= (sy_p - xz_p + ys_p + zx_p)y \\
&= syy_p - xyz_p + y^2s_p + yzx_p \\
&= y^2s_p + yzx_p + syy_p - xyz_p
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{Y}z &= (sy_p - xz_p + ys_p + zx_p)z \\
&= szy_p - xzz_p + yzs_p + z^2x_p \\
&= yzs_p + z^2x_p + szy_p - xzz_p
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{Z}s &= (sz_p + xy_p - yx_p + zs_p)s \\
&= s^2z_p + sxy_p - syx_p + szs_p \\
&= szs_p - syx_p + sxy_p + s^2z_p
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{Z}x &= (sz_p + xy_p - yx_p + zs_p)x \\
&= sxz_p + x^2y_p - xyx_p + xzs_p \\
&= xzs_p - xyx_p + x^2y_p + sxz_p
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{Z}y &= (sz_p + xy_p - yx_p + zs_p)y \\
&= syz_p + xyy_p - y^2x_p + yzs_p \\
&= yzs_p - y^2x_p + xyy_p + syz_p
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{Z}z &= (sz_p + xy_p - yx_p + zs_p)z \\
&= szz_p + xzy_p - yzx_p + z^2s_p \\
&= z^2s_p - yzx_p + xzy_p + szz_p
\end{aligned}
$$

The scalar component of qpq^*:

$$
\begin{aligned}
&\mathbb{S}s + \mathbb{X}x + \mathbb{Y}y + \mathbb{Z}z \\
&= s^2 s_p - sxx_p - syy_p - szz_p + \\
&\quad x^2 s_p + sxx_p - xzy_p + xyz_p + \\
&\quad y^2 s_p + yzx_p + syy_p - xyz_p + \\
&\quad z^2 s_p - yzx_p + xzy_p + szz_p \\
&= (s^2 + x^2 + y^2 + z^2)s_p + \\
&\quad (-sx + sx + yz - yz)x_p + \\
&\quad (-sy - xz + sy + xz)y_p + \\
&\quad (-sz + xy - xy + sz)z_p \\
&= (s^2 + x^2 + y^2 + z^2)s_p + 0x_p + 0y_p + 0z_p \\
&= (s^2 + x^2 + y^2 + z^2)s_p
\end{aligned}
$$

The x component of qpq^*:

$$
\begin{aligned}
&-\mathbb{S}x + \mathbb{X}s - \mathbb{Y}z + \mathbb{Z}y \\
&= -sxs_p + x^2 x_p + xyy_p + xzz_p + \\
&\quad sxs_p + s^2 x_p - szy_p + syz_p + \\
&\quad -yzs_p - z^2 x_p - szy_p + xzz_p + \\
&\quad yzs_p - y^2 x_p + xyy_p + syz_p \\
&= (-sx + sx - yz + yz)s_p + \\
&\quad (x^2 + s^2 - z^2 - y^2)x_p + \\
&\quad (xy - sz - sz + xy)y_p + \\
&\quad (xz + sy + xz + sy)z_p \\
&= 0s_p + (s^2 + x^2 - y^2 - z^2)x_p + (2xy - 2sz)y_p + (2xz + 2sy)z_p \\
&= (s^2 + x^2 - y^2 - z^2)x_p + (2xy - 2sz)y_p + (2xz + 2sy)z_p
\end{aligned}
$$

The y component of qpq^*:

$$
\begin{aligned}
&-\mathbb{S}y + \mathbb{X}z + \mathbb{Y}s - \mathbb{Z}x \\
&= -sys_p + xyx_p + y^2 y_p + yzz_p + \\
&\quad xzs_p + szx_p - z^2 y_p + yzz_p + \\
&\quad sys_p + szx_p + s^2 y_p - sxz_p + \\
&\quad -xzs_p + xyx_p - x^2 y_p - sxz_p \\
&= (-sy + xz + sy - xz)s_p + \\
&\quad (xy + sz + sz + xy)x_p + \\
&\quad (y^2 - z^2 + s^2 - x^2)y_p + \\
&\quad (yz + yz - sx - sx)z_p \\
&= 0s_p + (2xy + 2sz)x_p + (s^2 - x^2 + y^2 - z^2)y_p + (2yz - 2sx)z_p \\
&= (2xy + 2sz)x_p + (s^2 - x^2 + y^2 - z^2)y_p + (2yz - 2sx)z_p
\end{aligned}
$$

The z component of qpq^*:

$$
\begin{aligned}
&-\mathbb{S}z - \mathbb{X}y + \mathbb{Y}x + \mathbb{Z}s \\
&= -szs_p + xzx_p + yzy_p + z^2z_p + \\
&\quad - xys_p - syx_p + yzy_p - y^2z_p + \\
&\quad xys_p + xzx_p + sxy_p - x^2z_p + \\
&\quad szs_p - syx_p + sxy_p + s^2z_p \\
&= (-sz - xy + xy + sz)s_p + \\
&\quad (xz - sy + xz - sy)x_p + \\
&\quad (yz + yz + sx + sx)y_p + \\
&\quad (z^2 - y^2 - x^2 + s^2)z_p \\
&= 0s_p + (2xz - 2sy)x_p + (2yz + 2sx)y_p + (s^2 - x^2 - y^2 + z^2)z_p \\
&= (2xz - 2sy)x_p + (2yz + 2sx)y_p + (s^2 - x^2 - y^2 + z^2)z_p
\end{aligned}
$$

Finally, combine s, x, y and z components into the rotation quaternion, qpq^*:

$$
\begin{aligned}
qpq^* &= \mathbb{S}s + \mathbb{X}x + \mathbb{Y}y + \mathbb{Z}z + \\
&\quad i(-\mathbb{S}x + \mathbb{X}s - \mathbb{Y}z + \mathbb{Z}y) + \\
&\quad j(-\mathbb{S}y + \mathbb{X}z + \mathbb{Y}s - \mathbb{Z}x) + \\
&\quad k(-\mathbb{S}z - \mathbb{X}y + \mathbb{Y}x + \mathbb{Z}s) \\
&= (s^2 + x^2 + y^2 + z^2)s_p + \\
&\quad i\left[(s^2 + x^2 - y^2 - z^2)x_p + (2xy - 2sz)y_p + (2xz + 2sy)z_p\right] + \\
&\quad j\left[(2xy + 2sz)x_p + (s^2 - x^2 + y^2 - z^2)y_p + (2yz - 2sx)z_p\right] + \\
&\quad k\left[(2xz - 2sy)x_p + (2yz + 2sx)y_p + (s^2 - x^2 - y^2 + z^2)z_p\right]
\end{aligned}
$$

Since ***q*** is a unit quaternion, substitute the followings to the above equation to simplify it a little further.

$$
\begin{aligned}
s^2 + x^2 - y^2 - z^2 &= 1 - 2y^2 - 2z^2 \\
s^2 - x^2 - y^2 + z^2 &= 1 - 2x^2 - 2y^2 \\
s^2 - x^2 + y^2 - z^2 &= 1 - 2x^2 - 2z^2
\end{aligned}
$$

$$
|q|^2 = s^2 + x^2 + y^2 + z^2 = 1
$$

The final simplified rotation quaternion qpq^* becomes:

$$
\begin{aligned}
qpq^* &= s_p + \\
&\quad i\left[(1 - 2y^2 - 2z^2)x_p + (2xy - 2sz)y_p + (2xz + 2sy)z_p\right] + \\
&\quad j\left[(2xy + 2sz)x_p + (1 - 2x^2 - 2z^2)y_p + (2yz - 2sx)z_p\right] + \\
&\quad k\left[(2xz - 2sy)x_p + (2yz + 2sx)y_p + (1 - 2x^2 - 2y^2)z_p\right]
\end{aligned}
$$

Now, we only take the x, y and z components (without *i*, *j* and *k*), and convert it to a matrix form. It becomes multiplying a 3x3 matrix to a 3D vector $\vec{p}$ to transform.

$$\begin{pmatrix} (1-2y^2-2z^2)x_p + (2xy-2sz)y_p + (2xz+2sy)z_p \\ (2xy+2sz)x_p + (1-2x^2-2z^2)y_p + (2yz-2sx)z_p \\ (2xz-2sy)x_p + (2yz+2sx)y_p + (1-2x^2-2y^2)z_p \end{pmatrix} =$$

$$\begin{pmatrix} 1-2y^2-2z^2 & 2xy-2sz & 2xz+2sy \\ 2xy+2sz & 1-2x^2-2z^2 & 2yz-2sx \\ 2xz-2sy & 2yz+2sx & 1-2x^2-2y^2 \end{pmatrix} \begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix}$$

The 3x3 matrix itself is the rotation matrix equivalent to the quaternion rotation:

$$M_R = \begin{pmatrix} 1-2y^2-2z^2 & 2xy-2sz & 2xz+2sy \\ 2xy+2sz & 1-2x^2-2z^2 & 2yz-2sx \\ 2xz-2sy & 2yz+2sx & 1-2x^2-2y^2 \end{pmatrix}$$

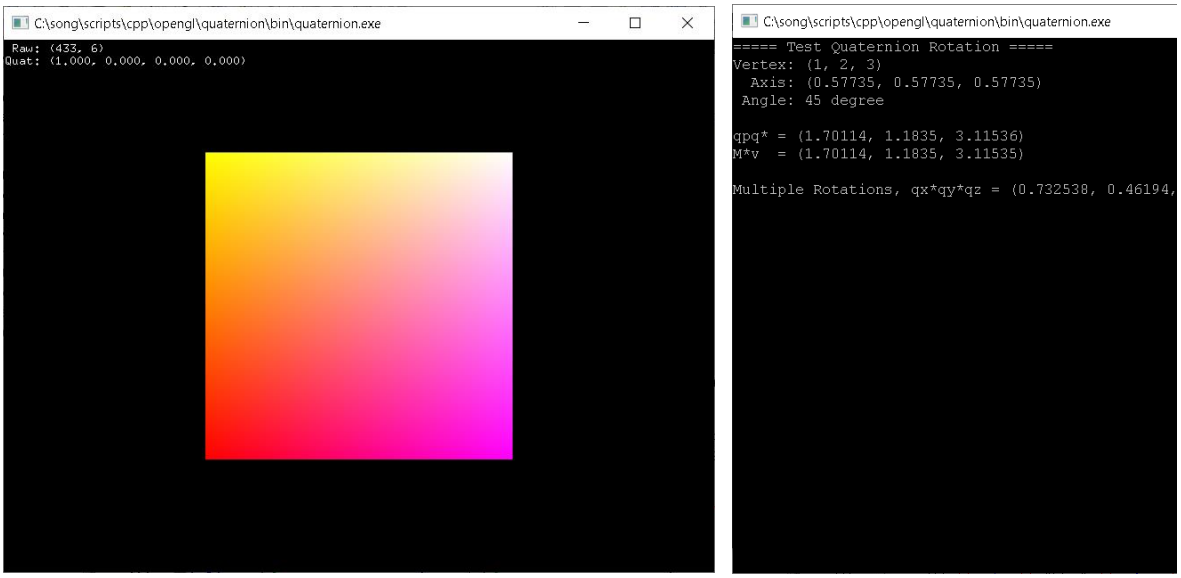
$$\text{where } q = s + ix + jy + kz, \quad |q| = 1$$

Or, as 4x4 matrix:

$$M_R = \begin{pmatrix} 1-2y^2-2z^2 & 2xy-2sz & 2xz+2sy & 0 \\ 2xy+2sz & 1-2x^2-2z^2 & 2yz-2sx & 0 \\ 2xz-2sy & 2yz+2sx & 1-2x^2-2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{where } q = s + ix + jy + kz, \quad |q| = 1$$

Example: Rotation with Quaternion



This example provides C++ Quaternion class and tests rotating a vertex, (1, 2, 3) along a rotation axis, (0.57735, 0.57735, 0.57735) by 45 degree. We need to convert the vertex and rotation axis to quaternion forms first, then rotate it with quaternion multiplication, qpq^*. The Quaternion class provides **getMatrix()** to convert the rotation quaternion to 4x4 matrix. It verifies the rotation result by comparing with rotation matrix.

```
Vector3 v(1, 2, 3);                             // 3D vertex to rotate
Vector3 r(0.57735f, 0.57735f, 0.57735f);        // rotation axis (unit vector)
float a = 45.0f;                                // rotation angle in degree

// convert to quaternions
Quaternion p = Quaternion(0, v.x, v.y, v.z);  // quaternion form of v
Quaternion q = Quaternion(r, a * 0.5f * D2R); // rot. quat. w/half-angle
Quaternion c = q;                               // copy of q
c.conjugate();                                  // q* (conjugate of q)

// rotate p by multiplying qpq*
Quaternion p2 = q * p * c;

// vector part of p2 contains the rotated 3D vertex
Vector3 v2(p2.x, p2.y, p2.z);                    // quaternion to vector
std::cout << "v2: " << v2 << std::endl;          // print the result

// OR, convert quaternion to 4x4 roatation matrix
Matrix4 m = q.getMatrix();
v2 = m * v;                                  // rotation using matrix instead

// OR, use matrix rotation directly
Matrix4 m
m.rotate(a, r);                               // rotate A degree along R axis
v2 = m * v;                                   // rotation using matrix instead
```

If there is a sequence of multiple rotations, we can simply multiply the quaternions one after another, similar to matrix transformations. Multiplication of 2 quaternions requires 16 products and 12 additions. On the contrary, multiplying two 4x4 matrices needs 64 products and 48 additions.

```
// 3 rotations in quaternion form
Quaternion qx = Quaternion(Vector3(1, 0, 0), 22.5f * D2R); // 45° about x
Quaternion qy = Quaternion(Vector3(0, 1, 0), 22.5f * D2R); // 45° about y
Quaternion qz = Quaternion(Vector3(0, 0, 1), 22.5f * D2R); // 45° about z

// compose multiple rotations, order is qz -> qy -> qx
Quaternion q = qx * qy * qz;
std::cout << "q: " << q << std::endl;         // print the result
```