Quaternions

Ken Shoemake
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104

Abstract

Of the many mathematical tools used in computer graphics, most are well covered in standard texts; quaternions are not. This article is intended to provide tutorial material on quaternions, including what they are, why they are useful, how to use them, where to use them, and when to be careful.

Introduction

Computer graphics uses quaternions as coordinates for rotations and orientations.

At SIGGRAPH 1985, quaternion curve methods were introduced to computer graphics to facilitate rotation animation. Although this is a rather specialized environmental niche, quaternions work so well they are able to compete successfully both with more general coordinates such as matrices, and with other special coordinates such as Euler angles.

Quaternion use has since expanded to include new curve methods and new applications, including physically based modeling, constraint systems, and user interfaces. This is because when a quaternion implementation is compared to other alternatives, it is usually simpler, cheaper, and better behaved.

That's the good news. The bad news is that researchers and implementors must learn some new mathematics, and quaternions are not taught in core mathematics and science courses. However, neither are homogeneous coordinates; and a broad view reveals that quaternions are simply a more sophisticated use of the four-component homogeneous coordinates to which we are already accustomed.

Rotations

What can be so unusual about three-dimensional rotations that they warrant their own coordinate system? Put simply, because rotations do not commute, they cannot be treated as vectors. Suppose $T_x(d)$ is translation along the x axis by d, and $R_x(\theta)$ is rotation around the x axis by θ; similarly for T_y, T_z, R_y, and R_z. Then $T_y(45\text{cm})\circ T_x(90\text{cm})$ commutes, meaning it equals $T_x(90\text{cm})\circ T_y(45\text{cm})$; but

$R_y(45°) \circ R_x(90°) \neq R_x(90°) \circ R_y(45°)$. (As usual, B◦A means the composition of action A followed by action B.) Note also that if we take the composition $R_y(180°) \circ R_x(180°)$ we get the same result as $R_z(180°)$!

It takes some study to unravel all the implications of these elementary observations, but they are far-reaching and often surprising. The set of all 3-dimensional rotations is not organized as a simple 3-dimensional vector space, but as a closed curved 3-dimensional manifold, which is also a (continuous) group. This group is known as SO(3). (For special and orthogonal.) Manifolds are generalizations of surfaces; this one has the topology of real 3-dimensional projective space, $\mathbf{R}P^3$, and also a natural distance measure related to rotation angle. Although a vector space such as the translations trivially splits into a product of lines, SO(3) does not split. Instead, it has a more sophisticated description as a fiber bundle over the sphere of directions, S^2, with fiber space the planar rotations, SO(2).

Unit quaternions have the remarkable property of capturing all of the geometry, topology, and group structure of 3-dimensional rotations in the simplest possible way. (Technically, they form what is called a universal covering.)

Quaternion definitions

Quaternions can be defined in several different, equivalent ways. It is helpful to know them all, since each form is useful. Historically, quaternions were conceived by Hamilton as like extended complex numbers, $w+\boldsymbol{i}x+\boldsymbol{j}y+\boldsymbol{k}z$, with $\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = -1$, $\boldsymbol{ij} = \boldsymbol{k} = -\boldsymbol{ji}$, with real w, x, y, z. (In honor of Hamilton, mathematicians denote the quaternions by $\mathbf{H}$.) Notice the non-commutative multiplication, their novel feature; otherwise, quaternion arithmetic is pretty much like real arithmetic. Hamilton was also quite aware of the more abstract possibility of treating quaternions as simply quadruples of real numbers $[x, y, z, w]$, with operations of addition and multiplication suitably defined. But it happens that the components naturally group into the imaginary part, (x, y, z), for which Hamilton coined the term *vector*, and the purely real part, w, which he called a *scalar*. Later authors (notably Gibbs) appropriated Hamilton's terminology and extracted from the clean operations of quaternion arithmetic the somewhat messier—but more general—operations of vector arithmetic. Courses today teach Gibbs' dot and cross products, so it is convenient to reverse history and describe the quaternion product using them. Thus we usually will want to write a quaternion as $[\mathbf{v}, w]$, with $\mathbf{v} = (x, y, z)$. We will

identify real numbers s with quaternions $[\mathbf{0}, s]$, and vectors $\mathbf{v} \in \mathbf{R}^3$ with quaternions $[\mathbf{v}, 0]$. Here are some basic facts.

Forms

$$\begin{aligned} q &\stackrel{\text{def}}{=} [\mathbf{v}, w] && ; \mathbf{v} \in \mathbf{R}^3, w \in \mathbf{R} \\ &= [(x, y, z), w] && ; x, y, z, w \in \mathbf{R} \\ &= [x, y, z, w] && ; x, y, z, w \in \mathbf{R} \\ &= \boldsymbol{i}x+\boldsymbol{j}y+\boldsymbol{k}z+w && ; x, y, z, w \in \mathbf{R},\ \boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1 \end{aligned}$$

Addition

$$\begin{aligned} q + q' &= [\mathbf{v}, w] + [\mathbf{v}', w'] \\ &\stackrel{\text{def}}{=} [\mathbf{v} + \mathbf{v}', w + w'] \end{aligned}$$

Multiplication

$$\begin{aligned} qq' &= [\mathbf{v}, w][\mathbf{v}', w'] \\ &\stackrel{\text{def}}{=} [\mathbf{v}\times\mathbf{v}'+w\mathbf{v}'+w'\mathbf{v}, ww'-\mathbf{v}\cdot\mathbf{v}'] \end{aligned}$$

Multiplication facts

$$(pq)q' = p(qq')$$

$$\begin{aligned} 1q = q1 &= [\mathbf{0}, 1][\mathbf{v}, w] \\ &= [1\mathbf{v}, 1w] = [\mathbf{v}, w] \end{aligned}$$

$$\begin{aligned} sq = qs &= [\mathbf{0}, s][\mathbf{v}, w] \\ &= [s\mathbf{v}, sw] \end{aligned}$$

$$\begin{aligned} \mathbf{v}\mathbf{v}' &= [\mathbf{v}, 0][\mathbf{v}', 0] \\ &= [\mathbf{v}\times\mathbf{v}', -\mathbf{v}\cdot\mathbf{v}'] \end{aligned}$$

Bilinearity

$$p(sq+s'q') = spq + s'pq'$$

$$(sq+s'q')p = sqp + s'q'p$$

Conjugate

$$\begin{aligned} q^* &= [\mathbf{v}, w]^* \\ &\stackrel{\text{def}}{=} [-\mathbf{v}, w] \end{aligned}$$

Conjugation facts

$$(q^*)^* = q$$

$$(pq)^* = q^*p^*$$

$$(p+q)^* = p^*+q^*$$

Norm

$$\begin{aligned} \mathrm{N}(q) &\stackrel{\text{def}}{=} qq^* = q^*q \\ &= w^2 + \mathbf{v}\cdot\mathbf{v} \\ &= w^2 + x^2 + y^2 + z^2 \end{aligned}$$

Norm facts

$$\mathrm{N}(qq') = \mathrm{N}(q)\mathrm{N}(q')$$

$$\mathrm{N}(q^*) = \mathrm{N}(q)$$

Inverse

$$q^{-1} = q^* / \mathrm{N}(q)$$

Unit quaternion facts

Let $\mathrm{N}(q) = \mathrm{N}(q') = \mathrm{N}(\hat{\mathbf{v}}) = 1$. Then:

$$q = [\hat{\mathbf{v}} \sin \Omega, \cos \Omega] \quad \text{(for some } \hat{\mathbf{v}}\text{)}$$

$$\mathrm{N}(qq') = 1$$

$$q^{-1} = q^*$$

$$\hat{\mathbf{v}}^2 = -1$$

$$\begin{aligned} e^{\hat{\mathbf{v}}\Omega} &= 1 + \hat{\mathbf{v}}\Omega + (\hat{\mathbf{v}}\Omega)^2/2! + \cdots \\ &= [\hat{\mathbf{v}} \sin \Omega, \cos \Omega] \end{aligned}$$

Notice that N(q) is a scalar, so the description of q^{-1} is well-defined. Otherwise, the non-commutativity of multiplication requires explicit expressions, such as pq^{-1}, instead of p / q.

The preceding list contains both definitions and consequences. It is a useful exercise to treat the consequences as lemmas, and prove them from the definitions. Each proof should be a straightforward computation.

Quaternions as rotations

The relationship of quaternions to three-dimensional rotations is contained in

Theorem 1. Let p be a point in three-dimensional (projective) space, represented as a quaternion using its homogeneous coordinates, $p = (x{:}y{:}z{:}w) \cong [(x, y, z), w] = [\mathbf{v}, w]$; and let q be any non-zero quaternion. Then:

1) The product qpq^{-1} takes $p = [\mathbf{v}, w]$ to $p' = [\mathbf{v}', w]$, with N($\mathbf{v}$) = N($\mathbf{v}'$).
2) Any non-zero real multiple of q gives the same action.
3) If N(q) = 1, then $q = [\hat{\mathbf{v}} \sin \Omega, \cos \Omega]$ acts to rotate around unit axis $\hat{\mathbf{v}}$ by 2Ω.

Proof. Let us first dispose of part 2. This is trivial, since the inverse of sq is $q^{-1}s^{-1}$, and scalar multiplication commutes. Thus $(sq)p(sq)^{-1} = sqpq^{-1}s^{-1} = qpq^{-1}ss^{-1} = qpq^{-1}$. So we can henceforth assume q is a unit quaternion, as stipulated in part 3, without loss of generality. For q a unit quaternion, $q^{-1} = q^*$, so we can write the action as qpq^*.

Now part 1 is simpler. Trivially, the action on a scalar gives that scalar, since scalar multiplication commutes. Similarly, the action on a vector, $[\mathbf{v}, 0]$, gives some vector, as we now show. The scalar part of any quaternion S(q) can be extracted using the formula $2S(q) = q + q^*$. So consider $2S(qpq^*) = (qpq^*) + (qpq^*)^* = qpq^* + qp^*q^*$. Since quaternion multiplication is bilinear, we can factor this as $q(p+p^*)q^* = q(2S(p))q^* = 2S(p)$. Hence the action of q on $p = [\mathbf{v}, w]$ gives $[\mathbf{v}', w]$. Because multiplication preserves norms, N(p) = N(p'); and since w is unchanged, N($\mathbf{v}$) = N($\mathbf{v}'$).

Finally we turn to part 3, the heart of the theorem, and consider the situation in Figure 1, where N($\mathbf{v}_0$) = N($\mathbf{v}_1$) = 1 and $q = \mathbf{v}_1\mathbf{v}_0^* = [\mathbf{v}_0{\times}\mathbf{v}_1, \mathbf{v}_0{\cdot}\mathbf{v}_1]$. We can identify Ω as the angle between $\mathbf{v}_0$ and $\mathbf{v}_1$, so that $\mathbf{v}_0{\cdot}\mathbf{v}_1 = \cos \Omega$. Then let $\hat{\mathbf{v}} = (\mathbf{v}_0{\times}\mathbf{v}_1)/\,||\,\mathbf{v}_0{\times}\mathbf{v}_1\,||$, a unit vector in the direction of the cross product—thus perpendicular to $\mathbf{v}_0$ and $\mathbf{v}_1$. Now we can write $q = [\hat{\mathbf{v}} \sin \Omega, \cos \Omega]$. (We shall assume $\mathbf{v}_1 \neq \pm\mathbf{v}_0$, else $q = \pm 1$, and the action is the identity.) We show that $\mathbf{v}_2\mathbf{v}_1^* = (q\mathbf{v}_0q^*)\mathbf{v}_1^*$ has the same components (dot and cross products) as $\mathbf{v}_1\mathbf{v}_0^*$; so $q\mathbf{v}_0q^*$ lies in the same plane as $\mathbf{v}_0$ and $\mathbf{v}_1$, and also forms an angle of Ω with $\mathbf{v}_1$.

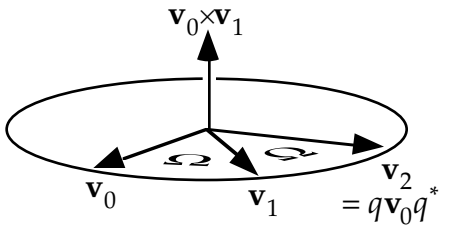

Figure 1. Geometry of action

Substituting the definition of q in $(q\mathbf{v}_0q^*)\mathbf{v}_1{}^*$ gives $(q\mathbf{v}_0(\mathbf{v}_1\mathbf{v}_0{}^*)^*)\mathbf{v}_1{}^*$, which simplifies to $q(\mathbf{v}_0\mathbf{v}_0)(\mathbf{v}_1{}^*\mathbf{v}_1{}^*)$. Since $\mathbf{v}_0$ and $\mathbf{v}_1{}^*$ are unit vectors, they square to –1, leaving q.

$$\begin{aligned}(q\mathbf{v}_0q^*)\mathbf{v}_1{}^* &= (q\mathbf{v}_0(\mathbf{v}_1\mathbf{v}_0{}^*)^*)\mathbf{v}_1{}^* \\ &= q\mathbf{v}_0(\mathbf{v}_0\mathbf{v}_1{}^*)\mathbf{v}_1{}^* \\ &= q(\mathbf{v}_0\mathbf{v}_0)(\mathbf{v}_1{}^*\mathbf{v}_1{}^*) \\ &= q(-1)(-1) \\ &= \mathbf{v}_1\mathbf{v}_0{}^*\end{aligned}$$

This confirms the equality, so the picture is accurate. Furthermore, q acts on $\mathbf{v}_1 = q\mathbf{v}_0$ to produce a vector which is also in the same plane, at an angle of Ω with $q\mathbf{v}_0q^*$. (We simply observe that $(q\mathbf{v}_1q^*)(q\mathbf{v}_0q^*)^* = (q(q\mathbf{v}_0)q^*)(q\mathbf{v}_0q^*)^* = q(q\mathbf{v}_0q^*)(q\mathbf{v}_0q^*)^* = q$.)

Interpreting this result in the terms stated in the theorem, we have just showed that the action of q on $\mathbf{v}_0$ and on $\mathbf{v}_1$ is, as required, to rotate around the axis $\hat{\mathbf{v}}$ by an angle of 2Ω. In fact, the action will be this same rotation on any vector p, as is shown by splitting p into $s_0\mathbf{v}_0+s_1\mathbf{v}_1+s\hat{\mathbf{v}}$. Bilinearity allows us to examine the action on $\mathbf{v}_0$, $\mathbf{v}_1$, and $\hat{\mathbf{v}}$ separately.

We already know the action on $\mathbf{v}_0$ and on $\mathbf{v}_1$, so consider the action on $\hat{\mathbf{v}}$. Observe that multiplication fails to commute simply because of the cross product. But in the product $q\hat{\mathbf{v}} = [\hat{\mathbf{v}}\sin\Omega, \cos\Omega][\hat{\mathbf{v}}, 0]$ the cross product is zero, so $q\hat{\mathbf{v}} = \hat{\mathbf{v}}q$, and $q\hat{\mathbf{v}}q^* = \hat{\mathbf{v}}qq^* = \hat{\mathbf{v}}$, which is consistent with the interpretation of $\hat{\mathbf{v}}$ as the axis of rotation.

Thus the action of q on every vector is a rotation around $\hat{\mathbf{v}}$ by 2Ω, which concludes the proof of part 3, and of Theorem 1. ∎

Corollary. Every three-dimensional rotation is the action of some unit quaternion.

Proof. Using part 3 of Theorem 1 we can get any axis and any angle. ∎

Quaternion rotation facts

Observe that the combination of rotation by q_1 followed by q_2 is given by $q = q_2q_1$, since

$$q_2(q_1pq_1{}^*)q_2{}^* = (q_2q_1)p(q_2q_1)^* = qpq^*.$$

Because quaternion multiplication is bilinear, it can be expressed in matrix form, and in two different ways. Multiplication on the left by $q = [(x, y, z), w] = [\mathbf{v}, w]$ gives qp as $\mathbf{L}_q p$, where p is now treated as a 4-dimensional column vector. The matrix $\mathbf{L}_q$ is

$$\mathbf{L}_q = \begin{bmatrix} w & -z & y & x \\ z & w & -x & y \\ -y & x & w & z \\ -x & -y & -z & w \end{bmatrix} = \begin{bmatrix} w\mathbf{I}+(\mathbf{v}\times\text{-}) & \mathbf{v} \\ -\mathbf{v}^{\mathrm{T}} & w \end{bmatrix}$$

The 2×2 matrix is a block partition of the $\mathbf{L}_q$ matrix, and $(\mathbf{v}\times\text{-})$ is the 3×3 matrix expressing the cross product of $\mathbf{v}$ with an arbitrary vector, namely

$$(\mathbf{v}\times\text{-}) = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

Multiplication on the right by q gives pq as $\mathbf{R}_q p$, where $\mathbf{R}_q$ is

$$\mathbf{R}_q = \begin{bmatrix} w & z & -y & x \\ -z & w & x & y \\ y & -x & w & z \\ -x & -y & -z & w \end{bmatrix} = \begin{bmatrix} w\mathbf{I}-(\mathbf{v}\times\text{-}) & \mathbf{v} \\ -\mathbf{v}^{\mathrm{T}} & w \end{bmatrix}$$

Multiplication on the right by q^* gives pq^* as $\mathbf{R}_{q^*} p$, where $\mathbf{R}_{q^*}$ is

$$\mathbf{R}_{q^*} = \begin{bmatrix} w & -z & y & -x \\ z & w & -x & -y \\ -y & x & w & -z \\ x & y & z & w \end{bmatrix} = \begin{bmatrix} w\mathbf{I}+(\mathbf{v}\times\text{-}) & -\mathbf{v} \\ \mathbf{v}^{\mathrm{T}} & w \end{bmatrix}$$

Multiplication on the left by q and on the right by q^* gives qpq^* as $\mathbf{Q}p$, where $\mathbf{Q} = \mathbf{L}_q\mathbf{R}_{q^*}$ is

$$\mathbf{Q} = \begin{bmatrix} w^2+x^2-y^2-z^2 & 2(xy-wz) & 2(xz+wy) & 0 \\ 2(xy+wz) & w^2-x^2+y^2-z^2 & 2(yz-wx) & 0 \\ 2(xz-wy) & 2(yz+wx) & w^2-x^2-y^2+z^2 & 0 \\ 0 & 0 & 0 & w^2+x^2+y^2+z^2 \end{bmatrix}$$

$$= \begin{bmatrix} (w\mathbf{I}+(\mathbf{v}\times\text{-}))^2+\mathbf{v}\mathbf{v}^{\mathrm{T}} & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & \mathrm{N}(q) \end{bmatrix}$$

This simplifies to

$$\mathbf{Q} = \begin{bmatrix} \mathrm{N}(q)-2(y^2+z^2) & 2(xy-wz) & 2(xz+wy) & 0 \\ 2(xy+wz) & \mathrm{N}(q)-2(x^2+z^2) & 2(yz-wx) & 0 \\ 2(xz-wy) & 2(yz+wx) & \mathrm{N}(q)-2(x^2+y^2) & 0 \\ 0 & 0 & 0 & \mathrm{N}(q) \end{bmatrix}$$

$$\cong \begin{bmatrix} 1-s(y^2+z^2) & s(xy-wz) & s(xz+wy) & 0 \\ s(xy+wz) & 1-s(x^2+z^2) & s(yz-wx) & 0 \\ s(xz-wy) & s(yz+wx) & 1-s(x^2+y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad ; s = 2 / \mathrm{N}(q)$$

The equivalence assumes **Q** acts on homogeneous coordinates, so each entry of the matrix can be divided by N(q) without changing the effect. When N(q) = 1, **Q** simplifies to

$$\mathbf{Q} = \begin{bmatrix} 1-2(y^2+z^2) & 2(xy-wz) & 2(xz+wy) & 0 \\ 2(xy+wz) & 1-2(x^2+z^2) & 2(yz-wx) & 0 \\ 2(xz-wy) & 2(yz+wx) & 1-2(x^2+y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

These observations produce the following core quaternion routines.

```
typedef struct {float x,y,z,w;} Quat;
typedef float HMatrix[4][4];
#define X 0
#define Y 1
#define Z 2
#define W 3

/* Return quaternion product qL * qR. */
Quat Qt_Mul(Quat qL, Quat qR)
{
    Quat qq;
    qq.w = qL.w*qR.w - qL.x*qR.x - qL.y*qR.y - qL.z*qR.z;
    qq.x = qL.w*qR.x + qL.x*qR.w + qL.y*qR.z - qL.z*qR.y;
    qq.y = qL.w*qR.y + qL.y*qR.w + qL.z*qR.x - qL.x*qR.z;
    qq.z = qL.w*qR.z + qL.z*qR.w + qL.x*qR.y - qL.y*qR.x;
    return (qq);
}

/* Return norm of quaternion, the sum of the squares of the components. */
#define Qt_Norm(q) ((q).x*(q).x + (q).y*(q).y + (q).z*(q).z + (q).w*(q).w)

/* Construct rotation matrix from (possibly non-unit) quaternion.
 * Assumes matrix is used to multiply column vector on the left:
 * vnew = mat vold.  Works correctly for right-handed coordinate system
 * and right-handed rotations. */
void Qt_ToMatrix(Quat q, HMatrix mat)
{
    double Nq = Qt_Norm(q);
    double s = (Nq > 0.0) ? (2.0 / Nq) : 0.0;
    double xs = q.x*s,       ys = q.y*s,       zs = q.z*s;
    double wx = q.w*xs,      wy = q.w*ys,      wz = q.w*zs;
    double xx = q.x*xs,      xy = q.x*ys,      xz = q.x*zs;
    double yy = q.y*ys,      yz = q.y*zs,      zz = q.z*zs;
    mat[X][X] = 1.0 - (yy + zz); mat[Y][X] = xy + wz; mat[Z][X] = xz - wy;
    mat[X][Y] = xy - wz; mat[Y][Y] = 1.0 - (xx + zz); mat[Z][Y] = yz + wx;
    mat[X][Z] = xz + wy; mat[Y][Z] = yz - wx; mat[Z][Z] = 1.0 - (xx + yy);
    mat[X][W] = mat[Y][W] = mat[Z][W] = 0.0;
    mat[W][X] = mat[W][Y] = mat[W][Z] = 0.0;
    mat[W][W] = 1.0;
}
```

```
/* Construct a unit quaternion from rotation matrix.  Assumes matrix is
 * used to multiply column vector on the left: vnew = mat vold.  Works
 * correctly for right-handed coordinate system and right-handed rotations.
 * Translation and perspective components ignored. */
Quat Qt_FromMatrix(HMatrix mat)
{
/* This algorithm avoids near-zero divides by looking for a large component
 *  — first w, then x, y, or z.  When the trace is greater than zero,
 * |w| is greater than 1/2, which is as small as a largest component can be.
 * Otherwise, the largest diagonal entry corresponds to the largest of |x|,
 * |y|, or |z|, one of which must be larger than |w|, and at least 1/2. */
    Quat qu;
    float tr, s;

    tr = mat[X][X] + mat[Y][Y]+ mat[Z][Z];
    if (tr >= 0.0) {
            s = sqrt(tr + mat[W][W]);
            qu.w = s*0.5;
            s = 0.5 / s;
            qu.x = (mat[Z][Y] - mat[Y][Z]) * s;
            qu.y = (mat[X][Z] - mat[Z][X]) * s;
            qu.z = (mat[Y][X] - mat[X][Y]) * s;
        } else {
            int h = X;
            if (mat[Y][Y] > mat[X][X]) h = Y;
            if (mat[Z][Z] > mat[h][h]) h = Z;
            switch (h) {
#define caseMacro(i,j,k,I,J,K) \
            case I:\
                s = sqrt( (mat[I][I] - (mat[J][J]+mat[K][K])) + mat[W][W] );\
                qu.i = s*0.5;\
                s = 0.5 / s;\
                qu.j = (mat[I][J] + mat[J][I]) * s;\
                qu.k = (mat[K][I] + mat[I][K]) * s;\
                qu.w = (mat[K][J] - mat[J][K]) * s;\
                break
            caseMacro(x,y,z,X,Y,Z);
            caseMacro(y,z,x,Y,Z,X);
            caseMacro(z,x,y,Z,X,Y);
#undefine caseMacro
            }
        }
    if (mat[W][W] != 1.0) {
        s = 1.0/sqrt(mat[W][W]);
        qu.w *= s;    qu.x *= s;    qu.y *= s;    qu.z *= s;
    }
    return (qu);
}
```

This last routine converts a rotation matrix to a unit quaternion, but it may not be the same as the one with which you created the matrix. Part 2 of Theorem 1 implies that quaternions are homogeneous coordinates for rotations. Thus q and $-q$ give the same rotation matrix, and the extraction routine can return either one; sometimes the choice matters.

Remember that SO(3) has the topology of 3-dimensional real projective space ($\mathbf{R}P^3$), but unit quaternions form a hypersphere (S^3) in four dimensions, which is topologically different. The identification of opposite points is what makes up the difference.

Remember, too, that SO(3) has a natural metric. To measure the distance from rotation Q to Q′, look at $Q' \circ Q^{-1}$. Since $Q' = (Q' \circ Q^{-1}) \circ Q$, the composition $Q' \circ Q^{-1}$ is the rotation connecting them. By part 3 of Theorem 1 the scalar part of $q'q^*$ is cos Ω, and indicates a rotation of 2Ω. But it is also $ww' + \mathbf{v} \cdot \mathbf{v}' = ww' + xx' + yy' + zz'$, the 4D dot product of q and q', thus cosine of the angular separation of q and q' on the sphere.

Quaternion curves

Any continuous quaternion curve which does not pass through [**0**, 0] gives a continuous sequence of rotations, and so may be used for animation. However the consequences of part 3 of Theorem 1 explored in the last section tell us that to gain control of what happens with the rotations we should confine our attention to unit quaternion curves.

Curves on a sphere are harder to create, understand, and control than ordinary splines. The recent literature contains three main approaches: geometric transliteration, differential equations, and arc blends. Here is a reference for each: J. Schlag, "Using Geometric Construction to Interpolate Orientations with Quaternions" in *Graphics Gems II*, Academic Press, 1991, pp. 377–380; A. Barr, B. Currin, S. Gabriel, and J. Hughes, "Smooth Interpolation of Orientations with Angular Velocity Constraints using Quaternions" in *Proceedings of SIGGRAPH '92*, ACM Press, 1992, pp. 313–320; W. Wang and B. Joe, "Orientation Interpolation in Quaternion Space Using Spherical Biarcs" in *Proceedings of Graphics Interface '93*, Morgan Kaufmann, 1993, pp. 24–32.

A common point of reference is the circular arc. Between any two unit quaternions q_0 and q_1 which are not opposite there is a unique shortest arc of the great circle given by the intersection of the unit sphere with a 2D plane containing q_0, q_1, and the origin. Planes which do not pass through the origin also intersect the sphere in arcs, but ones which curve more. Great arcs traversed at constant speed are, in fact, geodesics; that is, they are paths with minimal acceleration, and so as straight as possible. A unit quaternion great circle arc gives a constant speed rotation around a fixed axis, and can be written $C(t) = (q_1 q_0^{-1})^t q_0$.

The geometric transliteration approach treats great arcs as the "moral equivalent" of line segments and interprets geometric constructions like de Casteljau's algorithm in those terms. The differential equation approach notes that natural cubic splines also minimize acceleration, and imposes a spherical constraint to get equations of motion. The arc blend approach—by far the cheapest of the three—combines small

arc segments to get a smooth curve. Given only two points to interpolate, all three approaches normally give a great arc.

Quaternion calculus

Just as real differential calculus extends to vectors and complex numbers, it also extends to quaternions. Although products do not commute, otherwise differentiation looks familiar, and is still a linear operator.

In geometric transliteration the equivalent of linear interpolation, or *lerp*, is called *Slerp*. Although Slerp has a geometric form,

$$\text{Slerp}(q_0, q_1; t) = q_0 \frac{\sin \Omega(1-t)}{\sin \Omega} + q_1 \frac{\sin \Omega t}{\sin \Omega} \qquad ; \cos \Omega = q_0 \cdot q_1$$

which is useful for implementation, the algebraic form $(q_1 q_0^{-1})^t q_0$ is better for analysis.

The derivative of q^t with respect to t is $\ln(q)\, q^t$, so the derivative of $\text{Slerp}(q_0, q_1; t)$ with respect to t is $\ln(q_1 q_0^{-1})\ \text{Slerp}(q_0, q_1; t)$. Euler's identity holds, so the logarithm of a unit quaternion $q = [\hat{\mathbf{v}} \sin \Omega, \cos \Omega]$ is just $\hat{\mathbf{v}}\Omega$, a pure vector of length Ω.

Now observe that if $\ln(q_1 q_0^{-1}) = \hat{\mathbf{v}}\Omega$, then Slerp causes a rotation in 3D around axis $\hat{\mathbf{v}}$ by an amount 2Ω in unit time, so $\hat{\mathbf{v}}\Omega$ is half the angular velocity vector. Turning this around, if we have angular velocity vector ω, the differential equation of motion is

$$\dot{q} = {}^1/_2\, \omega\, q\,.$$

This is also used in dynamics simulation, where ω can vary over time.

Quaternion bundle structure

Tracking an object with a camera requires care to avoid tilt but minimize twist; the curve techniques mentioned so far do not achieve this. One approach is to exploit the fiber bundle structures of SO(3) and S^3 to limit the twist of an untilted sequence obtained by other means.

Fiber bundles are similar to product spaces, but allow a richer global structure. Recall that a product space comes with two projections, one to each factor, which concisely capture the structure of the space. Thus if X is the product of X_1 and X_2, each point $p \in X$ is formed as a pair, $\langle \pi_1(p), \pi_2(p) \rangle$, where π_1 and π_2 are the projections $\pi_1: X \to X_1$ and $\pi_2: X \to X_2$. For example, a cylindrical band is the product of a circle with a line segment, as in Figure 2.

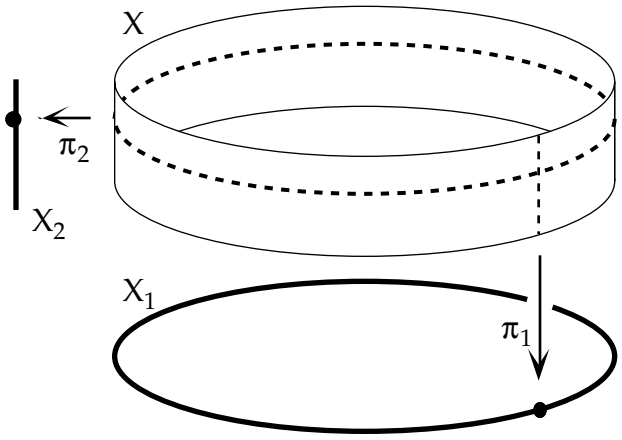

Figure 2. Product space.

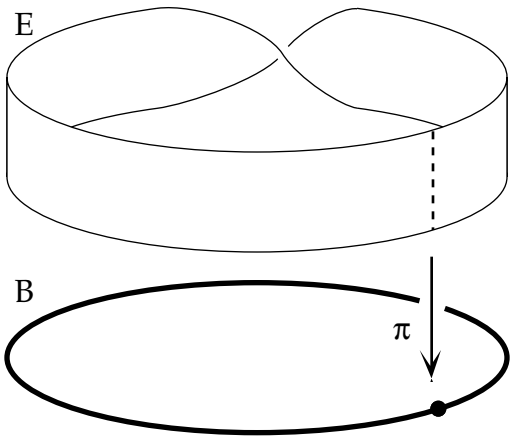

Figure 3. Fiber bundle.

Figure 3 depicts a Möbius band, which is a fiber bundle topologically different from the cylindrical band. The twist makes it impossible to define a continuous projection onto the line segment; yet projection onto the circle is still possible, and the inverse image of a point on the circle is a line segment in the band. This asymmetry reappears in fiber bundle terminology: the *total space* of the bundle projects onto the *base space*, and the inverse image of each base point—the *fiber* over that point—is a copy of the *fiber space*. (Properly speaking, the projection map is also part of the bundle.) A final, crucial, requirement is that the inverse image of some neighborhood of each base point actually is a product of that neighborhood with the fiber space.

We can exhibit both the rotations SO(3) and the unit quaternions S^3 as fiber bundles over the ordinary sphere S^2 using the projection $\pi(q) = q\hat{\mathbf{z}}q^{-1} = q[(0, 0, 1), 0]q^{-1}$. Since a rotated unit vector is still a unit vector, $q\hat{\mathbf{z}}q^{-1}$ gives a point on S^2, as promised. The fiber above $\pi(q)$ consists of quaternions of the form $q[(0, 0, \sin\tau), \cos\tau]$, rotations that differ from q by a twist of 2τ around the z axis. Topologically, the fiber space is a circle, S^1. Neither of these bundles over S^2 are product spaces, so there is no globally consistent way to line up the fibers. There is, however, a natural way to *lift* a path on S^2 to a path on either bundle, which means we can line up fibers along the path.

Now notice that under the usual camera model we look down the z axis. Thus if q controls the camera orientation, $\pi(q)$ tells us the direction we are looking! To limit twist, we want to alter the path on S^3 without disturbing its projection to S^2. Using the natural lift, we can measure the twist from frame i to frame i+1 (quaternions q_i and q_{i+1}) as follows. Let $\mathbf{v}_i = \pi(q_i)$, and set $p_i = \sqrt{\mathbf{v}_{i+1}\mathbf{v}_i^{-1}}$. (To take the square root of a quaternion geometrically, halve the arc between it and the identity.) The twist is

simply the angle of $r_i = q_i^{-1} p_i^{-1} q_{i+1}$, which is a rotation around the z axis. If we think there is too much twist, we can replace q_{i+1} by q_{i+1} [(0, 0, sin $-\Delta$), cos $-\Delta$], where 2Δ is the amount of excess twist.

Quaternion user interfaces

No one in computer graphics would deny that homogeneous matrices are invaluable, but few users would want to specify a camera transformation by typing matrix entries. While not everyone agrees, many would argue that if you must type in numbers for rotations, Euler angles are more meaningful than quaternions. Preferably, however, input will come from graphical interaction, and here quaternions excel.

The geometric construction used in the proof of Theorem 1 generated a unit quaternion by using two points on a unit sphere. (One side note: If we call $\mathbf{v}_1 \mathbf{v}_0^{-1}$ a vector quotient, every quaternion is such a quotient, though not uniquely.) We saw that the rotation so specified had a natural geometric relationship to the points used. Since it is easy to pick points on a sphere using interactive graphics, a user interface can be built on this principle. Such a method is described in K. Shoemake, "ARCBALL: A User Interface for Specifying Three-Dimensional Orientation Using a Mouse" in *Proceedings of Graphics Interface '92*, Morgan Kaufmann, 1992. It turns out to have several desirable properties not found in other interfaces.

Quaternions have also lent themselves to direct graphical camera specification, as described in M. Gleicher and A. Witkin, "Through-the-Lens Camera Control" in *Proceedings of SIGGRAPH '92*, ACM Press, 1992, pp. 331–340. Both papers mention that, although implementations are possible without quaternions, nothing else is as simple and robust.