Quaternions, rotations and compression

Geomerics

AUTHOR
Chris Doran
April 16, 2015

1 Introduction

This blog was generated by a question from Glenn Fiedler regarding good schemes for compressing the relative rotation between two near-by configurations. Before we look at that problem we start with some background.

1.1 Unit quaternions and rotors

To fully understand quaternions it is my belief that you should see them in their natural setting of the geometric algebra of three-dimensional space. In this context you can see how they arise naturally as the generators of rotations, and how they act on vectors. There are also some neat tie-ins with quantum mechanics, that I hope to get to in a later post.

For now the key idea we need is that a rotation can be represented by a unit quaternion. In geometric algebra we refer to unit quaternions as rotors, a concept that then extends to arbitrary spaces. Here I will stick with the term unit quaternion.

A general quaternion has four components

$$q = w + x\mathbf{i} + y\mathbf{j} + x\mathbf{k}. \tag{1}$$

A unit quaternion is normalised to 1:

$$qq^* = w^2 + x^2 + y^2 + z^2 = 1. \tag{2}$$

A unit quaternion defines a rotation matrix through a simple two-to-one relation. The mapping is two-to-one because both q and -q specify the same rotation.

Unit quaternions can be thought of as unit vectors in a 4 dimensional space, so they lie on the surface of a 3-sphere. The 3-sphere can therefore be thought of as the space of the group of unit quaternions, and it is an example of a Lie group manifold. There is some fascinating geometry behind this idea, which again will have to wait for another post.

The reason unit quaternions form a group is that the product of any two unit quaternions results in a third unit quaternion. This proof is quite simple:

$$(q_1q_2)(q_1q_2)^* = q_1q_2q_2^*q_1^* = q_1q_1^* = 1. \tag{3}$$

This multiplicative behaviour is the way we expect to combine rotations.

The fact that the group manifold is a 3-sphere helps us understand how to interpolate between rotations. The shortest path between two unit quaternions is a great circle on the 3-sphere. To find points on this path we can just linearly interpolate between the two unit quaternions and normalise the result. Quaternions get close to linearising many relationships involving rotations, which makes them so much more convenient than rotation matrices and Euler angles.

One further result is useful when studying unit quaternions. Every unit quaternion can be written as the exponential of a quaternion vector (a quaterion with no w component),

$$q = e^{B/2} \tag{4}$$

The factor of two here ensures that the magnitude of B is precisely the angle of rotation. The direction of the quaternion vector is the rotation axis. In the language of geometric algebra we would call a quaternion vector a 'bivector' and think of it as representing a plane.

1.2 Relative rotations

The problem posed by Glenn concerns network physics, where you want to update the state of an object each frame. Rotating objects cannot rotate that much between frames so each new unit quaternion is close to its predecessor, which should allow for a good compression scheme.

Suppose that our base quaternion is q_0, and the next unit quaternion in sequence is q_1. We write

$$q_1 = q_1q_0^*q_0 = Rq_0 \tag{5}$$

and focus attention on the relative rotation defined by R. Essentially we have rotated q_0 back to the identity, so that the relative rotation is guaranteed to be small. We can now write

$$R = \exp(B) = \cos|B| + \sin|B|\frac{B}{|B|} \tag{6}$$

and with some simple trig functions we can recover the quaternion vector B, which is guaranteed to have a small magnitude. If f is the frequency of update, then the angular velocity is given by

$$\omega = 2|B|f \tag{7}$$

A typical physics solver would have an upper limit on the angular velocities present in a system and typically we find $|B|$ around 0.1, though this would depend on the update rate / frame rate.

At this point you can fall back to your favourite compression scheme for small vectors in three dimensions. Glenn Fiedler's original problem contained a number of axis-aligned boxes, which meant most updates took place along the three axial directions, allowing for further compression. But this is a bit of an accident of his setup, and in general you would expect B to be fairly evenly distributed.

1.3 The Cayley transform

The preceding scheme is fine, but the trig functions in both the encode and decode step are a bit of an overhead. There is an alternative way to represent a rotation in terms of a quaternion vector that dates back to Cayley and is quite useful here. Instead of the exponential form we write

$$R = \frac{1+B}{1-B} = \frac{1-|B|^2}{1+|B|^2} + \frac{2B}{1+|B|^2}. \tag{8}$$

This inverts simply to give

$$B_i = \frac{R_i}{1+R_0} \tag{9}$$

where the subscript i denotes the component of the vector part of the quaternion. These relations ensure that both the encode and decode steps are simple.

The end result is much the same as the previous scheme: a quaternion vector with small overall magnitude. This can be compressed aggressively with little loss of accuracy.

The geometry behind the Cayley transform is interesting. The transform is performing a stereographic projection of the 3-sphere to a 3-dimensional Euclidean space. Near the identity this is a good way of linearising the space, which is appropriate for handling small rotations.

1.4 Rigid-body dynamics

The preceding schemes both assume that we only know the preceding value of a unit quaternion, and use this as a basis for computing the next one. But for most physics applications this is probably a bad idea. You can do a lot better if you keep the preceding two orientations, and use these to compute the next orientation assuming that no torques are present. This is the same as using position and velocity to update the position, assuming no forces are present. In most situations the rigid body will behave like a free particle, or subject to a force like gravity that only induces small variations. Bigger changes only come about due to collisions, which should be resolved within one or two frames.

To see what the evolution looks like in the absence of torques, we need to form the angular velocity for a time-varying unit quaternion. This is given by

$$B = 2\dot{q}q^* \tag{10}$$

In the absence of external torques B is constant, and the motion is described by

$$q = e^{Bt/2}q_0. \tag{11}$$

Now let's assume that our time-step is δ, and we keep the values at two previous values, q_0 and

$$q_{-1} = e^{-B\delta/2}q_0. \tag{12}$$

Our predicted value at q_1 is

$$q_1 = e^{B\delta/2}q_0 = (q_{-1}q_0^*)q_0 = q_0q_{-1}q_0. \tag{13}$$

Geometrically what this does is reflect the point q_{-1} through the point q_0 to give the new predicted point. In fact, the quaternion products here are achieving the geometric product of vectors in 4 dimensions!

In most cases q_1 will give a better base prediction for the actual value than q_0 allowing for even more aggressive compression.