Incremental Spherical Linear Interpolation

Tony Barrera *
Barrera Kristiansen AB

Anders Hast †
Creative Media Lab,
University of Gävle

Ewert Bengtsson ‡
Centre for Image Analysis
Uppsala University

Abstract

Animation is often done by setting up a sequence of key orientations, represented by quaternions. The in between orientations are obtained by spherical linear interpolation (SLERP) of the quaternions, which then can be used to rotate the objects. However, SLERP involves the computation of trigonometric functions, which are computationally expensive. Since it is often required that the angle between each quaternion should be the same, we propose that incremental SLERP is used instead. In this paper we demonstrate five different methods for incremental SLERP, whereof one is new, and their pros and cons are discussed.

1 Introduction

We propose that two approaches previously published for spherical linear interpolation (SLERP) [Shoemake 1985] of vectors and intensities [Barrera 2004; Hast 2003], also are used for SLERP of quaternions, a technique often used in animation. We also demonstrate two other ways of performing incremental SLERP by using quaternion multiplication and by using matrices directly. Furthermore we propose a new way of doing incremental SLERP by posing the problem as solving a differential equation using the Euler method.

1.1 Quaternions and Animation

Animation is often done by setting up a sequence of key orientations, represented by quaternions. The in between orientations are obtained by spherical linear interpolation of the quaternions, which then can be used in order to rotate the objects [Svarovsky 2000]

$$p' = qpq^{-1} \tag{1}$$

where q is a unit quaterion and p is a position i 3D in quaternion form

$$p = (x, y, z, 0) \tag{2}$$

Unit quaternions are simply another representation of rotation matrices. It is possible to convert quaternions into matrices and vice versa. Actually it can be proven that a unit quaternion will be the eigenvector of the corresponding orthonormal rotation matrix [Kuipers 1999]. The quaternion consists of four elements. The first three are the sine of half the rotation angle multiplied by a unit vector **n** that the rotation is performed around. The last one is the cosine of half the rotation angle

$$q = (\mathbf{v}, s) = (\mathbf{n}\sin(\theta/2), \cos(\theta/2)) \tag{3}$$

For a unit quaternion, the conjugate is the same as the inverse, which is defined as

$$q^{-1} = (-\mathbf{v}, s) \tag{4}$$

1.2 SLERP

The theory behind SLERP, was introduced to the computer graphics society by Shoemake [Shoemake 1985] and an interesting proof is given by Glassner [Glassner 1999]. SLERP is different from linear interpolation in the way that the angle between each quaternion will be constant, i.e. the movement will have constant speed. Linear interpolation (lerp) will yield larger angles in the middle of the interpolation sequence. This will cause animated movements to accelerate in the middle, which is undesirable [Parent 2002]. Moreover, will quaternions interpolated by lerp be shorter and normalization is therefore necessary. However, quaternions obtained by SLERP do not need to be normalized, since SLERP will always yield unit quaternions as long as the quaternions being interpolated in between are unit quaternions.

The formula used is

$$q(t) = q_1 \frac{\sin((1-t)\theta)}{\sin(\theta)} + q_2 \frac{\sin(t\theta)}{\sin(\theta)} \tag{5}$$

where $t \in [0,1]$, and θ is the angle between q_1 and q_2 computed as

$$\theta = cos^{-1}(q_1 \cdot q_2) \tag{6}$$

2 Fast incremental SLERP

It is shown in [Hast 2003] that equation (5) can be rewritten as

$$q(n) = q_1 \cos(nK_\theta) + q_o \sin(nK_\theta) \tag{7}$$

where q_o is the quaternion obtained by applying one step of Gram-Schmidt's orthogonalization algorithm [Nicholson 1995] and then it is normalized. This quaternion is orthogonal to q_1 and lies in the same hyper plane spanned by q_1 and q_2. Furthermore, if there are k steps then the angle between each quaternion is $K_\theta = \frac{\cos^{-1}(q_1 \cdot q_2)}{k}$. In the subsequent matlab code examples, it is assumed that q_1 and q_2 are unit quaternions and that the following five lines are included in the beginning of the code

```
qo=q2-(dot(q2,q1))*q1;
qo=qo/norm(qo);
t=acos(dot(q1,q2));
kt=t/k;
q(1,:)=q1;
```

*e-mail: tony.barrera@spray.se
†e-mail: aht@hig.se
‡e-mail: ewert@cb.uu.se

The code that follows on these three lines can be used for computing incremental SLERP. Usually SLERP is computed using trigonometric functions in the inner loop in the following way

```
b=kt;
for n=2:k+1
  q(n,:)=q1*cos(b)+qo*sin(b);
  b=b+kt;
end
```

The cost for computing SLERP in this way is one scalar addition, one sine and one cosine evaluation, two scalar-quaternion multiplications and one quaternion addition. This is not very efficient and the following subsections will show how these trigonometric functions can be avoided.

2.1 De Moivre

In [Hast 2003] it is shown that complex multiplication can be used in order to compute SLERP efficiently. Hence, the computationally expensive trigonometric functions are avoided.

Complex numbers are defined in an orthonormal system where the basis vectors are $\mathbf{e_1} = 1$ and $\mathbf{e_2} = i$. The De Moivre's formula [Marsden 1996] states that

$$(\cos(\theta) + i\sin(\theta))^n = \cos(n\theta) + i\sin(n\theta) \tag{8}$$

The right part of the formula looks very similar to equation (7). In fact is possible to compute the left part instead, by using one complex multiplication. However, there are still some things to arrange before we can compute SLERP of quaternions instead of complex numbers.

Let Z be a complex number computed by

$$Z = \cos(K_\theta) + i\sin(K_\theta) \tag{9}$$

Furthermore, we treat complex numbers as if they were vectors in 2D-space, by defining a product which is similar to the ordinary dot product

$$(q_1, q_o) \bullet (\Re(Z), \Im(Z)) = q_1\Re(Z) + q_o\Im(Z) \tag{10}$$

Thus, we compute the SLERP as

$$q(n) = (q_1, q_o) \bullet Z^n \tag{11}$$

Finally, we prove that this will yield the desired result. Rewrite equation (11) by using equation (8)

$$\begin{aligned} q(n) &= (q_1, q_o) \bullet Z^n \\ &= (q_1, q_o) \bullet (\cos(n\theta) + i\sin(n\theta)) \end{aligned} \tag{12}$$

Then, expand this equation by using equation (10)

$$q(n) = q_1\cos(nK_\theta) + q_o\sin(nK_\theta) \tag{13}$$

The matlab code for this approach is

```
Z1=cos(kt)+i*sin(kt);
Z=Z1;
for n=2:k+1
  q(n,:)=q1*real(Z)+qo*imag(Z);
  Z=Z1*Z;
end
```

The cost for computing the intensity will be one complex multiplication and one two scalar-quaternion multiplications and one quaternion addition.

2.2 Chebyshev

In [Barrera 2004] a faster approach is derived using the Chebyshev recurrence [Burden 2001]. This will make the computation in the inner loop much more effective than using the complex multiplication.

The Chebyshev's recurrence relation is

$$T_{n+1}(u) = 2uT_n(u) - T_{n-1}(u) \tag{14}$$

where $T_n(u)$ are Chebyshev polynomials [Burden 2001] of the first kind that can be written as

$$T_n(u) = \cos(n\phi) = \cos(n\cos^{-1}u) \tag{15}$$

In order to obtain the Chebyshev recurrence for solving equation (7) let

$$u = \cos(K_\theta) \tag{16}$$

$$\phi = K_\theta \tag{17}$$

Then equation (15) gives

$$\cos(nK_\theta) = \cos(n\cos^{-1}u) \tag{18}$$

This equation can be solved by equation (14). Since sin is just a phase shifted cos it will also be solved by the proposed equation as long as the angle is the same for both. Let

$$q(n) = q_1\cos(nK_\theta) + q_o\sin(nK_\theta) \tag{19}$$

For the setup starting with $n = 0$, we subsequently have

$$\begin{aligned} q_{-1} &= q_1\cos(-K_\theta) + q_o\sin(-K_\theta) \\ &= q_1\cos(K_\theta) - q_o\sin(K_\theta) \end{aligned} \tag{20}$$

and

$$\begin{aligned} q_0 &= q_1\cos(0) + q_o\sin(0) \\ &= q_1 \end{aligned} \tag{21}$$

Now, we can put together our algorithm. We can also optimize it somewhat by removing the factor 2 from the loop

```
tm1=q1*cos(kt)-qo*sin(kt);
t0=q1;
u=2*cos(kt);
for n=2:k+1
  tp1=u*t0-tm1;
  q(n,:)=tp1;
  tm1=t0;
  t0=tp1;
end
```

The cost for this approach is one scalar-quaternion multiplication, one quaternion subtraction and two quaternion moves.

2.3 Quaternion Power Function

The power function variant of SLERP for quaternions [Shankel 2000]can also be used for incremental SLERP. It is defined as

$$q(n) = q_1(q_1^{-1}q_a)^n \tag{22}$$

We have to compute q_a in such a way that we will obtain intermediate quaternions in between q_1 and q_2

$$q_a = q_1\cos(K_\theta) + q_o\sin(K_\theta) \tag{23}$$

then we obtain the following algorithm

```
q1i=[-q1(1), -q1(2), -q1(3), q1(4)];
qa=q1*cos(kt)+qo*sin(kt);
qb=qmul(q1i,qa);
q0=q1;
for n=2:k+1
    q0=qmul(q0,qb);
    q(n,:)=q0;
end
```

Here *qmul* is an implementation of quaternion multiplication, which is defined as

$$q_1 q_2 = (\mathbf{v}_1, s_1)(\mathbf{v}_2, s_2) \tag{24}$$
$$= (s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2, s_1 s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2) \tag{25}$$

The cost for this approach is only one quaternion multiplication in the inner loop.

2.4 Euler

Here we introduce how it is possible to perform incremental SLERP by solving a ordinary differential equation using the Euler method [Gerald 1994]. It is easy to show that

$$y = y_1 \cos(K_\theta) + y_o \sin(K_\theta) \tag{26}$$

is a solution to the ordinary differential equation [Simmons 1991]

$$y'' = -k^2 y \tag{27}$$

where k is a constant. The Euler method gives an approximation to the solution by the following forward difference

$$y_{i+1} = y_i + y' \tag{28}$$

However, in order to make this an exact computation we shall compute the derivate using Eulers method as well

$$y'_{i+1} = y'_i + y'' \tag{29}$$

The trick is to use the fact that $y'' = -k^2 y$, then we can nest the recurrence as

$$y'_{i+1} = y'_i - k^2 y_i \tag{30}$$
$$y_{i+1} = y_i + y'_{i+1} \tag{31}$$

Substitute equation (30) into (31)

$$y_{i+1} = y_i + y'_i - k^2 y_i \tag{32}$$

Now, we make use of the fact that $y'_i = y_i - y_{i-1}$ and put that into equation (32). Thus

$$y_{i+1} = y_i + y_i - y_{i-1} - k^2 y_i \tag{33}$$

After rearranging the terms we get

$$y_{i+1} = (2 - k^2) y_i - y_{i-1} \tag{34}$$

If we let $(2 - k^2) = 2u$, where $u = \cos(K_\theta)$ then equation (34) is the same as the Chebyshev recurrence (14). This implies that the nested Euler method can be used to compute incremental SLERP exactly. Therefore

$$k^2 = 2 - 2\cos(K_\theta) \tag{35}$$

In this case we have

$$y_{-1} = y_1 \cos(K_\theta) - y_o \sin(K_\theta) \tag{36}$$
$$y_0 = y_1 \tag{37}$$
$$\tag{38}$$

And

$$y'_0 = y_0 - y_{-1} \tag{39}$$
$$= y_1 - y_1 \cos(K_\theta) + y_o \sin(K_\theta) \tag{40}$$
$$= y_1 (1 - \cos(K_\theta)) + y_o \sin(K_\theta) \tag{41}$$

Putting it all together yields the following algorithm

```
p=1-cos(kt);
y=q1;
k2=2*p;
y1=qo*sin(kt)+p*q1;

for n=2:k+1
    y1=y1-k2*y;
    y=y+y1;
    q(n,:)=y;
end
```

The cost is one scalar-quaternion multiplication, one quaternion subtraction and one quaternion addition.

2.5 Matrix Multiplication

The previously presented algorithms can be useful if quaternions are used to perform the actual rotation of the objects. However, if the quaternions must be transformed into matrices it is better to do the interpolation using matrices all along.

Let

$$M_1 = quattomat(q_1) \tag{42}$$
$$M_a = quattomat(q_1 \cos(K_\theta) + q_o \sin(K_\theta)) \tag{43}$$

where *quattomat* [Watt 1992] is a function that transforms a quaternion into a matrix. The there must exist a matrix, M that transforms M_1 into M_a

$$M_1 M = M_a \tag{44}$$

Thus

$$M = M_1^{-1} M_a \tag{45}$$

The code is

```
M1=quattomat(q1);
Ma=quattomat(q1*cos(kt)+qo*sin(kt));
M=inv(M1)*Ma;
Q=M1;
for n=2:k+1
    Q=Q*M;
end
```

3 Conclusions

It is possible to evaluate SLERP for quaternions used in animation in a very efficient way using incremental SLERP instead of evaluating the original SLERP functions containing trigonometric functions in the loop.

All presented algorithms will yield the same intermediate quaternions, or as in the case for the matrix version, the corresponding rotation matrix. They are all interpolation approaches and not approximations. The only error introduced is by the floating point arithmetic itself. They are all therefore numerically stable algorithms.

The fastest approach for quaternion SLERP is the one derived from Chebyshev's recurrence relation. The others may seem redundant. However, the purpose of this paper is to show that there

are many ways to avoid the computationally expensive trigonometric functions in the inner loop. Perhaps the other approaches might have other uses in other contexts. The algorithm derived from De Moivre's formula might turn out to be useful if complex multiplication is implemented in hardware. The same goes for the Quaternion power function approach. The one using the Euler method is interesting as an alternative to the Chebyshev approach since the variable swap is replaced by a quaternion addition.

If the rotation is done by matrix multiplication instead of quaternion rotation, then the Matrix multiplication approach can be even faster than the Chebyshev approach since matrix multiplication often is implemented in hardware and quaternion multiplication (for the rotation of the objects) is usually not. This of course heavily depend of which platform is being used for animation.

Anyhow, many graphics applications and especially animation could benefit from having quaternion arithmetics implemented in hardware. That would make incremental SLERP very fast. Nonetheless, a software implementation using any of the proposed approaches for SLERP will still be much faster than using trigonometric functions in the inner loop.

References

T. BARRERA, A. HAST, E. BENGTSSON 2004. *Faster shading by equal angle interpolation of vectors* IEEE Transactions on Visualization and Computer Graphics, pp. 217-223.

R. L. BURDEN, J. D. FAIRES 2001. *Numerical Analysis* Brooks/Cole, Thomson Learning, pp. 507-516.

C. F. GERALD, P. O. WHEATLEY 1994. *Applied Numerical Analysis, 5:th ed.* Addison Wesley, pp. 400-403.

A. GLASSNER 1999. *Situation Normal* Andrew Glassner's Notebook- Recreational Computer Graphics, Morgan Kaufmann Publishers, pp. 87-97.

A. HAST, T. BARRERA, E. BENGTSSON 2003. *Shading by Spherical Linear Interpolation using De Moivre's Formula* WSCG'03, Short Paper, pp. 57-60.

J. B. KUIPERS 1999. *Quaternions and rotation Sequences - A Primer with Applications to Orbits, Aerospace, and Virtual Reality* Princeton University Press, pp. 54-57, 162,163.

J. E. MARSDEN, M. J. HOFFMAN 1996. *Basic Complex Analysis* W. H. Freeman and Company, pp. 17.

W. K. NICHOLSON 1995. *Linear Algebra with Applications* PWS Publishing Company, pp. 275,276.

R. PARENT 2002. *Computer Animation - Algorithms and Techniques* Academic Press, pp. 97,98.

J. SHANKEL 2000. *Interpolating Quaternions* Game Programming Gems. Edited by M. DeLoura. Charles River Media, pp. 205-213

K. SHOEMAKE 1985. *Animating rotation with quaternion curves* ACM SIGGRAPH, pp. 245-254.

G. F. SIMMONS 1991. *Differential Equations with Applications and Historical Notes, 2:nd ed.* MacGraw Hill, pp. 64,65.

J. SVAROVSKY 2000. *Quaternions for Game Programming* Game Programming Gems. Edited by M. DeLoura. Charles River Media, pp. 195-299.

A. WATT, M. WATT 1992. *Advanced Animation and Rendering Techniques - Theory and Practice* Addison Wesley, pp. 363.