A Fast and Accurate Estimate for SLERP

David Eberly, Geometric Tools, Redmond WA 98052
https://www.geometrictools.com/

Created: September 11, 2018

Contents

This is a brief summary of my paper [3] that describes how to estimate the SLERP of two unit-length quaternions by a polynomial of two variables. Table 1 of the original paper lists the maximum errors depending on number of polynomial terms, but those errors were transcribed from the Remez algorithm used for an error-balancing scheme and are incorrect. This document provides a similar analysis of errors but includes a new error-balancing algorithm, one that has the essence of what I had intended in the original paper. For simplicity, throughout this document I will use the term *quaternion* with the understanding that it refers to a *unit-length quaternion*.

1 Introduction

The *spherical interpolation* (SLERP) of quaternions is a common operation in keyframe animation [4]. The operation can be a significant bottleneck in an animation-heavy application. The standard implementation of SLERP typically involves trigonometric function evaluations, divisions and branching. This document describes how to estimate SLERP accurately using only multiplications and additions. The algorithm is based on ideas from Chebyshev polynomials, power series solutions for differential equations and error balancing.

The SLERP of two quaternions q_0 and q_1 is

$$S(t, q_0, q_1) = \left(\frac{\sin((1-t)\theta)}{\sin(\theta)}\right) q_0 + \left(\frac{\sin(t\theta)}{\sin(\theta)}\right) q_1 \tag{1}$$

where $\theta \in [0, \pi)$ is the angle between q_0 and q_1 considered as 4-dimensional vectors and where $t \in [0, 1]$. The angle between q_0 and q_1 is determined by $\cos\theta = q_0 \cdot q_1$. The curve generated by $S(t, q_0, q_1)$ is the shortest path on the hypersphere containing q_0 and q_1. It is also a constant-speed parameterization; that is, $|dS/dt|$ is a constant for all t and, in fact, $|dS/dt| = \theta$. The angle $\theta = \pi$ is excluded because q_0 and q_1 are antipodal, in which case there are infinitely many shortest paths connecting them.

Quaternions are a compact representation of rotations. They provide a double covering of the set of rotations in that if q represents a rotation, then $-q$ represents the same rotation. In keyframe animation, a sequence of quaternions is chosen to represent orientations at various times, say, $\{q_i\}_{i=0}^{m-1}$, and SLERP is applied to pairs (q_i, q_{i+1}) to generate intermediate orientations between the associated times. A standard practice is to preprocess the sequence so that the angle between two consecutive quaternions is in $[0, \pi/2]$. Listing 1 contains pseudocode for the preprocessing.

Listing 1. Pseudocode for preprocessing a sequence of quaternions so that consecutive quaternions have acute angle between them.

```
Quaternion q[m];  // the sequence to be preprocessed
for (int i = 1; i < m; ++i)
{
    Real cosTheta = Dot(q[i - 1], q[i]);
    if (cosTheta < 0)
    {
        q[i] = -q[i];
    }
}
```

For the remainder of the document, it is assumed that SLERP is applied to quaternions q_0 and q_1 for which $\theta \in [0, \pi/2]$. Implications are that $\cos\theta = q_0 \cdot q_1 \geq 0$ and $\sin\theta \geq 0$.

Listing 2 contains pseudocode for a typical implementation of SLERP.

Listing 2. A typical implementation of SLERP is a direct conversion of equation (1) to source code. The assumption here—and throughout this document—is that the angle between q_0 and q_1 is in $[0, \pi/2]$.

```
Quaternion SLERP (Real t, Quaternion q0, Quaternion q1)
{
    Real cosTheta = Dot(q0, q1);
    if (cosTheta < 1)
    {
        // θ is in (0, π/2]
        Real theta = acos(cosTheta);
        Real invSinTheta = 1 / sin(theta);
        Real c0 = sin((1 - t) * theta) * invSinTheta;
        Real c1 = sin(t * theta) * invSinTheta;
        return c0 * q0 + c1 * q1;
    }
    else
    {
        // θ is 0, so just return one of the inputs
        return q0;
    }
}
```

A branch statement is used to guard against a division by zero. In the common case that $\cos\theta < 1$, the code involves one call to acos, three calls to sin and one division. For a large number of SLERP calls, the evaluation can be expensive.

One way to avoid the expensive function calls is to approximate sin by a polynomial and acos by a square root and a polynomial [1]. However, a simple observation about the coefficients used in SLERP and some basic ideas from linear differential equations lead to an estimate that is less expensive to evaluate, using only multiplications and additions; there are no branches or divisions.

2 Chebyshev Polynomials

The coefficient $\sin(t\theta)/\sin(\theta)$ in equation (1) is evaluated for $t \in [0, 1]$. It has the same form as $\sin(n\theta)/\sin(\theta)$ for nonnegative integers n, an expression that is related to *Chebyshev polynomials of the second kind* [5]. The polynomials are defined recursively by

$$u_0(x) = 1, \quad u_1(x) = 2x, \quad u_n(x) = 2xu_{n-1}(x) - u_{n-2}(x) \ \text{ for } n \geq 2 \tag{2}$$

and have the property

$$u_n(\cos(\theta)) = \sin((n+1)\theta)/\sin(\theta) \tag{3}$$

They are solutions to the second-order linear differential equation

$$\left(1 - x^2\right) u_n''(x) - 3xu_n'(x) + n(n+2)u_n(x) = 0 \tag{4}$$

For the purpose of this document, I refer to $u_{n-1}(x)$ which is a solution to the differential equation

$$\left(x^2-1\right)u''_{n-1}(x)+3xu'_{n-1}(x)+\left(1-n^2\right)u_{n-1}(x)=0 \tag{5}$$

I also restrict the domain $x=\cos(\theta)\in[0,1]$ for angles $\theta\in[0,\pi/2]$.

Equation (5) allows for a continuous variable t rather than the discrete variable n, so if we define $u_{t-1}(\cos(\theta))=\sin(t\theta)/\sin(\theta)$ for real-valued $t\in[0,1]$, the SLERP equation is rewritten as

$$S(t,q_0,q_1)=u_{-t}(\cos(\theta))\,q_0+u_{t-1}(\cos(\theta))\,q_1 \tag{6}$$

Equation (6) suggests that we can construct formulas for u_{-t} and u_{1-t} that depend only on $\cos(\theta)=q_0\cdot q_1$, thereby avoiding the explicit computation of θ and the calls to the sine function.

3 Power Series Solutions of Differential Equations

Define $f(x,t)=u_{t-1}(x)$, which is viewed as a function of x for a specified real-valued parameter t. It is a solution to equation (5) with n formally replaced by t,

$$\left(x^2-1\right)f''(x,t)+3xf'(x,t)+\left(1-t^2\right)f(x,t)=0 \tag{7}$$

The prime symbols denote differentiation with respect to x.

We may specify an initial value for $f(1,t)$. Obtaining a unique solution to a linear second-order differential equation normally requires specifying the derivative value $f'(1,t)$; however, the differential equation is singular at $x=1$ because the coefficient of f'' is 0 at $x=1$. The uniqueness is actually guaranteed by specifying only the value for $f(1,t)$. When x is 1, θ is 0 and evaluation of $u_{t-1}(1)$ is in the limiting sense,

$$u_{t-1}(1)=\lim_{\theta\to0}u_{t-1}(\cos(\theta))=\lim_{\theta\to0}\frac{\sin(t\theta)}{\sin(\theta)}=\lim_{\theta\to0}\frac{t\cos(t\theta)}{\cos(\theta)}=t \tag{8}$$

The next-to-last equality of equation (8) uses an application of l'Hôpital's Rule. The initial condition is therefore $f(1,t)=t$.

A standard undergraduate course on differential equations shows how to solve the differential equation using power series [2]. Because we want an expansion at $x=1$, the powers are $(x-1)^i$. The next equation lists power series for $f(x,t)$, $f'(x,t)$ and $f''(x,t)$ where the coefficients are necessarily functions of t,

$$f=\sum_{i=0}^{\infty}a_i(t)(x-1)^i,\quad f'=\sum_{i=0}^{\infty}ia_i(t)(x-1)^{i-1},\quad f''=\sum_{i=0}^{\infty}i(i-1)a_i(t)(x-1)^{i-2} \tag{9}$$

Substituting these into equation (7),

$$\begin{aligned}0&=\left(x^2-1\right)f''+3xf'+\left(1-t^2\right)f\\&=\left[(x-1)^2+2(x-1)\right]f''+3\left[(x-1)+1\right]f'+\left(1-t^2\right)f\\&=\textstyle\sum_{i=0}^{\infty}\left[(i+1)(2i+3)a_{i+1}+((i+1)^2-t^2))a_i\right](x-1)^i\end{aligned} \tag{10}$$

For the power series to be identically zero for all x, it is necessary that the coefficients are all zero. The condition $f(1,t)=t$ implies $a_0(t)=t$. The remaining coefficients are determined by a recurrence equation,

$$a_0(t)=t;\quad a_i(t)=\frac{t^2-i^2}{i(2i+1)}\,a_{i-1}(t),\quad i\geq1 \tag{11}$$

It is apparent from equation (11) that $a_i(t)$ is a polynomial in t with degree $2i+1$. Observe that $a_0(0) = 0$, which implies $a_i(0) = 0$ for $i \geq 0$; this is equivalent to $f(x, 0) = 0$. Similarly, $a_0(1) = 1$ and $a_1(1) = 0$, which implies $a_i(1) = 0$ for $i \geq 1$; this is equivalent to $f(x, 1) = 1$.

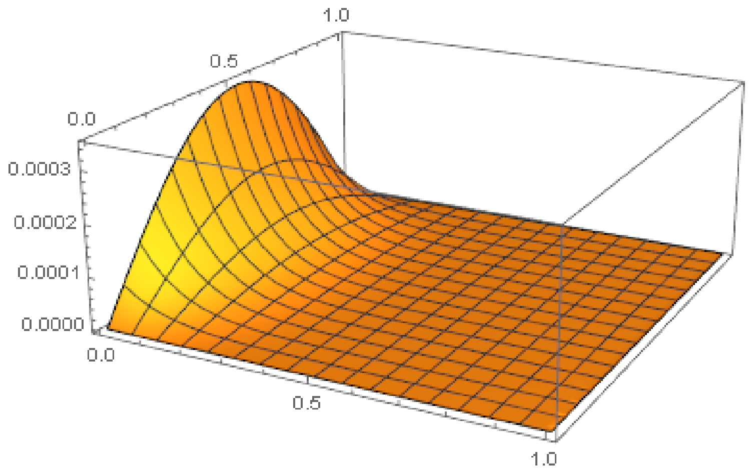
4 Truncation Error Bounds

We now have a power series for $f(x,t)$ using powers of $(x-1)$ and whose coefficients are polynomials in t that may be generated iteratively. The power series can be truncated to obtain an approximation,

$$f(x,t) = \sum_{i=0}^{n} a_i(t)(x-1)^i + \sum_{i=n+1}^{\infty} a_i(t)(x-1)^i = \sum_{i=0}^{n} a_i(t)(x-1)^i + e(x,t,n) \tag{12}$$

where $e(x,t,n)$ is the truncation error. It is simple to show that $e(x, 0, n) = 0$, $e(x, 1, n) = 0$ and $e(1, t, n) = 0$. Figure 1 shows a graph of $e(x, t, 8)$.

Figure 1. The graph of the truncation error $e(x, t, 8)$. The global maximum occurs for $x = 0$ and for some $t \in (0, 1)$.



For $0 < t < 1$, the coefficients from Equation (11) alternate in sign with $a_0 = t > 0$, $a_1 = t(t^2-1)/3 < 0$, $a_2 = t(t^2-1)(t^2-4)/30 > 0$, and so on. When $x \in [0, 1)$, the terms $(x-1)^i$ alternate in sign. The terms $a_i(t)(x-1)^i$ are all positive for $t \in (0, 1)$ and for $x \in [0, 1)$, which implies that the truncation error is nonnegative. The truncation error is also decreasing for $t \in (0, 1)$ and $x \in [0, 1)$ because $\partial e/\partial x < 0$. The maximum truncation error must occur when $x = 0$ for some $t \in (0, 1)$. Moreover, the partial sum of the first $n+1$ terms of the power series is always an overestimate of the actual value for $f(x,t)$.

The function at $x = 0$ $(\theta = \pi/2)$ is

$$f(0,t) = \sin(\pi t/2) = \sum_{i=0}^{\infty} (-1)^i a_i(t) = \sum_{i=0}^{n} (-1)^i a_i(t) + e_n(t) \tag{13}$$

where the last equality defines the function $e_n(t) = e(0, t, n) = \sum_{i=n+1}^{\infty}(-1)^i a_i(t)$. Table 1 lists the maximum truncation errors for $1 \le n \le 16$. Mathematica [6] was used to generate the table.

Table 1. The maximum truncation errors for estimating $f(x, t)$. These occur for $x = 0$ and for some $t \in (0, 1)$. The index n, the maximum error e and the location t are listed in the table.

n	e	t	n	e	t
1	$8.260470 \star 10^{-2}$	0.536171	9	$1.651390 \star 10^{-4}$	0.509133
2	$3.534481 \star 10^{-2}$	0.526101	10	$7.906542 \star 10^{-5}$	0.508368
3	$1.573194 \star 10^{-2}$	0.520518	11	$3.798801 \star 10^{-5}$	0.507723
4	$7.165432 \star 10^{-3}$	0.516943	12	$1.830625 \star 10^{-5}$	0.507171
5	$3.313713 \star 10^{-3}$	0.514448	13	$8.844315 \star 10^{-6}$	0.506693
6	$1.549189 \star 10^{-3}$	0.512604	14	$4.282478 \star 10^{-6}$	0.506275
7	$7.301926 \star 10^{-4}$	0.511182	15	$2.077657 \star 10^{-6}$	0.505907
8	$3.463652 \star 10^{-4}$	0.510053	16	$1.009723 \star 10^{-6}$	0.505580

The maximum truncation error for $n + 1$ is roughly half that for n.

5 Increased Accuracy using Error Balancing

A simple error-balancing algorithm is to truncate the series but then modulate the last term of the estimate by a constant that is chosen to minimize the maximum error. The error function for $(x, t) \in [0, 1]^2$ is

$$\varepsilon(x, t, n) = f(x, t) - \left(\sum_{i=0}^{n-1} a_i(t)(x-1)^i + u_n a_n(t)(x-1)^n\right) \tag{14}$$

and the estimate itself is inside the parentheses of the right-hand side of the equation. The constant is constrained to $u_n \ge 1$. The computations were performed in C++, which showed that indeed the balancing occurred. The balanced error function is bimodal, as shown in figure 2 for $n = 8$.

Figure 2. The graph of the balanced error $\varepsilon(x, t, 8)$. The global maximum occurs for $x = 0$ and for some $t \in (0, 1)$. The global minimum occurs for a pair (x, t) for some $x > 0$ and for some $t > 0$.

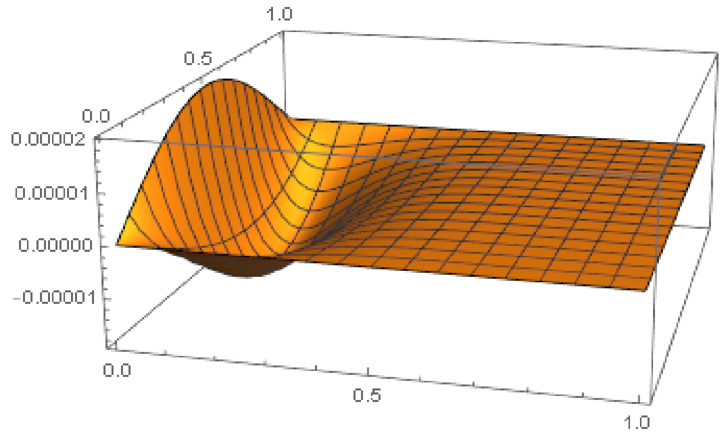


Table 2 lists the maximum balanced errors of $|\varepsilon(x, t, n)|$ for $1 \leq n \leq 16$. These were computed using C++ and floating-point arithmetic, so some numerical rounding errors are to be expected.

Table 2. The maximum balanced errors for estimating $f(x,t)$. The table lists the index n, the u-value, the maximum error e, the location (x_0,t_0) where $\varepsilon(x,t,n)$ attains its global maximum and the location (x_1,t_1) where the balanced error attains its global minimum. It is the case that $\varepsilon(x_1,t_1,n) = -\varepsilon(x_0,x_0,n)$.

n	u	$\lvert\varepsilon\rvert$	x_0	t_0	x_1	t_1
1	1.51497046463192	$1.824927 \star 10^{-2}$	0.0	0.433593	0.500000	0.609375
2	1.64101846329868	$5.276024 \star 10^{-3}$	0.0	0.445312	0.349609	0.566406
3	1.71248771436512	$1.805473 \star 10^{-3}$	0.0	0.457031	0.269531	0.550781
4	1.75935455970466	$6.724403 \star 10^{-4}$	0.0	0.460937	0.220703	0.539062
5	1.79270498268306	$2.638615 \star 10^{-4}$	0.0	0.464843	0.187500	0.531250
6	1.81774808652699	$1.073123 \star 10^{-4}$	0.0	0.468750	0.162109	0.527343
7	1.83728783018887	$4.480529 \star 10^{-5}$	0.0	0.472656	0.144531	0.523437
8	1.85298109240830	$1.908783 \star 10^{-5}$	0.0	0.476562	0.128906	0.523437
9	1.86587355099618	$8.262877 \star 10^{-6}$	0.0	0.476562	0.117187	0.519531
10	1.87666328810155	$3.623664 \star 10^{-6}$	0.0	0.480468	0.107421	0.519531
11	1.88582968525589	$1.606393 \star 10^{-6}$	0.0	0.480468	0.099609	0.515625
12	1.89371240325272	$7.187185 \star 10^{-7}$	0.0	0.484375	0.091796	0.515625
13	1.90057151205838	$3.240655 \star 10^{-7}$	0.0	0.484375	0.085937	0.515625
14	1.90659055672586	$1.471217 \star 10^{-7}$	0.0	0.484375	0.080078	0.511718
15	1.91192105598748	$6.718082 \star 10^{-8}$	0.0	0.484375	0.076171	0.511718
16	1.91666919924319	$3.084173 \star 10^{-8}$	0.0	0.484375	0.070312	0.511718

6 Application to SLERP with Error Bounds

From equation (6), the SLERP of two quaternions q_0 and q_1 with $q_0 \cdot q_1 \geq 0$ and for some $t \in [0,1]$ is

$$s = u_{-t}(\cos\theta)q_0 + u_{t-1}(\cos\theta)q_1 = f(x,1-t)q_0 + f(x,t)q_1 \doteq \hat{f}(x,1-t,n)q_0 + \hat{f}(x,t,n)q_1 \tag{15}$$

where $x = \cos\theta$ and the approximation to $f(x,t)$ is $\hat{f}(x,t,n)$ given by

$$\hat{f}(x,t,n) = \sum_{i=0}^{n-1} a_i(t)(x-1)^i + u_n a_n(t)(x-1)^n \tag{16}$$

Define $f_0(x,t) = f(x,1-t)$ and $f_1(x,t) = f(x,t)$. Let $\hat{f}_i(x,t,n)$ be the approximations to f_i(x,t); then $\hat{f}_i(x,t,n) = f_i(x,t) - \varepsilon_i(x,t,n)$ with $|\varepsilon_i(x,t,n)| \leq \mu_n$. The number μ_n is the maximum error obtained by error balancing.

The approximation to SLERP is $\hat{s} = \hat{f}_0 q_0 + \hat{f}_1 q_1$. A measure of error is the length ℓ of the difference of the quaternions treated as 4-tuple vectors,

$$\ell^2 = |\varepsilon_0 q_0 + \varepsilon_1 q_1|^2 = \varepsilon_0^2 + 2\varepsilon_0\varepsilon_1 q_0 \cdot q_1 + \varepsilon_1^2 \leq \varepsilon_0^2 + 2|\varepsilon_0||\varepsilon_1| + \varepsilon_1^2 = (|\varepsilon_0| + |\varepsilon_1|)^2 \leq 4\mu_n^2 \tag{17}$$

where I have used $0 \leq q_0 \cdot q_1 = \cos\theta \leq 1$. Therefore, $\ell \leq 2\mu_n$. If you wish to have no more than error $\lambda > 0$ for ℓ, require $\ell \leq 2\mu_n \leq \lambda$. Table 2 may be used to select the appropriate n to ensure $\mu_n \leq \lambda/2$.

7 Increased Accuracy using Domain Reduction

Throughout the document, the constraint on the quaternions is that the angle θ between them is in $[0, \pi/2]$. The largest errors in estimation occur when the angle is near $\pi/2$. If we impose the constraint $\theta \in [0, \pi/4]$, the maximum balanced errors are smaller than those when allowing $\theta \in [0, \pi/2]$. Table 3 lists n, u_n and the maximum error.

Table 3. The maximum balanced errors for estimating $f(x,t)$ with $x \in [\sqrt{1/2}, 1]$. The table lists the index n, the u-value, the maximum error e and the location (x_0, t_0) where $|\varepsilon(x,t,n)|$ attains its global maximum.

n	u	$\lvert\varepsilon\rvert$	x_0	t_0
1	1.10214708745480	$8.683070 \star 10^{-4}$	0.870715	0.613281
2	1.12393511831760	$6.603950 \star 10^{-5}$	0.707106	0.445312
3	1.13518715649843	$6.194899 \star 10^{-6}$	0.707106	0.453125
4	1.14210623875260	$6.457696 \star 10^{-7}$	0.778041	0.542968
5	1.14680204913020	$7.179185 \star 10^{-8}$	0.766600	0.535156
6	1.15020192041993	$8.336272 \star 10^{-9}$	0.758591	0.527343
7	1.15277890488505	$9.989402 \star 10^{-10}$	0.751727	0.527343
8	1.15479905903339	$1.226171 \star 10^{-10}$	0.707106	0.476562
9	1.15642677247524	$1.533473 \star 10^{-12}$	0.707106	0.476562
10	1.15776545554399	$1.947220 \star 10^{-12}$	0.707106	0.476562
11	1.15888605639338	$2.503552 \star 10^{-13}$	0.707106	0.472656
12	1.15983392670751	$3.252953 \star 10^{-14}$	0.707106	0.468750
13	1.16057553887367	$4.329869 \star 10^{-15}$	0.707106	0.429687
14	1.16066097468138	$6.661338 \star 10^{-16}$	0.707106	0.367187
15	1.15067637711763	$3.330669 \star 10^{-16}$	0.707106	0.367187
16	1.06647791713476	$3.330669 \star 10^{-16}$	0.707106	0.367187

Given quaternions q_0 and q_1 with $q_0 \cdot q_1 \in [0,1]$, in which case $\theta \in [0, \pi/2]$, let $q_h = (q_0 + q_1)/|q_0 + q_1| = S(1/2, q_0, q_1)$. Note that $|q_0 + q_1| = 2\cos(\theta/2)$. The angle between q_0 and q_h is the same as the angle between q_h and q_1, namely, $\theta/2 \in [0, \pi/4]$. It can be shown that

$$S(t, q_0, q_1) = \begin{cases} S(2t, q_0, q_h), & t \in [0, 1/2] \\ S(2t-1, q_h, q_1), & t \in [1/2, 1] \end{cases} \tag{18}$$

The SLERP functions on the right-hand side involve angles in $[0, \pi/4]$, so the approximation is more accurate for those evaluations than using the original.

For an animation setting where you use SLERP on pairs of quaternions from a sequence, the q_h values can be precomputed during the preprocessing that ensures the angle between a consecutive pair of quaternions is acute.

8 An Implementation for the CPU

Naturally, we wish to compute the approximation of SLERP as efficiently as possible. To illustrate, consider the case $n = 2$, the coefficient $\hat{f}(x,t)$ is computed by the following where μ refers to the modulator of the last term of the approximation,

$$\begin{aligned} \hat{f}(x,t) &= a_0(t) + a_1(t)(x-1) + (1+\mu)a_2(t)(x-1)^2 \\ &= t\left(1 + \left(\tfrac{(t^2-1)(x-1)}{3}\right) + (1+\mu)\left(\tfrac{(t^2-1)(x-1)}{3}\right)\left(\tfrac{(t^2-4)(x-1)}{10}\right)\right) \\ &= t\left(1 + b_0 + (1+\mu)b_0 b_1\right) \\ &= t\left(1 + b_0\left(1 + (1+\mu)b_1\right)\right) \end{aligned} \tag{19}$$

where $b_0 = (x-1)(t^2-1)/3$ and $b_1 = (x-1)(t^2-4)/10$. For $n = 3$, the approximation is $\hat{f}(x,t) = t(1 + b_0(1 + b_1(1 + (1+\mu)b_2)))$ for b_0 and b_1 as in the case $n = 2$ and for $b_2 = (x-1)(t^3-9)/21$. For general n, the nested expression is similar,

$$b_i(x,t) = (x-1)\left(\frac{t^2 - (i+1)^2}{(i+1)(2i+3)}\right) = (x-1)(u_i t^2 - v_i) \tag{20}$$

for $1 \le i \le n$. The last equality defines the constants u_i and v_i; these are precomputed in an implementation.

Pseudocode for the approximation for a standard floating-point unit is shown in Listing 3 for $n = 8$. It must compute both $\hat{f}(x,t)$ and $\hat{f}(x, 1-t)$ to produce the SLERP estimate $\hat{s} = \hat{f}(x, 1-t)q_0 + \hat{f}(x,t)q_1$.

Listing 3. An implementation of the approximation to SLERP with $n = 8$.

```
// These constants are precomputed or inlined by the compiler.
Real mu = 1.85298109240830;  // from Table 2
Real u[8] = // 1/[i(2i+1)] for i >= 1
{
    1.0/(1*3), 1.0/(2*5), 1.0/(3*7), 1.0/(4*9), 1.0/(5*11), 1.0/(6*13), 1.0/(7*15), mu/(8*17)
};
Real v[8] = // i/(2i+1) for i >= 1
{
    1.0/3, 2.0/5, 3.0/7, 4.0/9, 5.0/11, 6.0/13, 7.0/15, mu*8.0/17
};

// It is assumed that the angle between q0 and q1 is acute.
Quaternion SLERP(Real t, Quaternion q0, Quaternion q1)
{
    Real xm1 = Dot(q0, q1) − 1;  // in [−1,0]
    Real d = 1 − t, sqrT = t * t, sqrD = d * d;

    Real bT[8], bD[8];  // bT[] stores t−related values, bD[] stores (1−t)−related values
    for (int i = 7; i >= 0; −−i)
    {
        bT[i] = (u[i] * sqrT − v[i]) * xm1;
        bD[i] = (u[i] * sqrD − v[i]) * xm1;
    }

    Real f0 = t * (
        1 + bT[0] * (1 + bT[1] * (1 + bT[2] *(1 + bT[3] * (
        1 + bT[4] * (1 + bT[5] * (1 + bT[6] *(1 + bT[7]))))))));

    Real f1 = d * (
        1 + bD[0] * (1 + bD[1] * (1 + bD[2] * (1 + bD[3] * (
        1 + bD[4] * (1 + bD[5] * (1 + bD[6] * (1 + bD[7]))))))));

    Quaternion slerp = f0 * q0 + f1 * q1;
    return slerp;
}
```

Observe that only multiplications and additions are used—no transcendental function calls, no divisions and no branching.

9 An Implementation for SIMD Using Intel® SSE

The choice $n = 8$ is convenient for a fast SIMD implementation. The source code presented next is for an Intel® processor with SSE support. If you have support for some of the advanced SSE instructions, you can replace the relevant multiply-add pairs of instructions by fused-multiply-add instructions. If available, you canuse a dot-product intrinsic instead of the dot-product function provided here. Listing 4 contains the SSE implementation.

Listing 4. An Intel® SSE implementation of the approximation to SLERP with $n = 8$.

```
// Precomputed constants.
Real mu = 1.85298109240830;  // from Table 2
const __m128 u0123 = _mm_setr_ps(1.f/(1*3), 1.f/(2*5), 1.f/(3*7), 1.f/(4*9));
const __m128 u4567 = _mm_setr_ps(1.f/(5*11), 1.f/(6*13), 1.f/(7*15), opmu/(8*17));
const __m128 v0123 = _mm_setr_ps(1.f/3, 2.f/5, 3.f/7, 4.f/9);
const __m128 v4567 = _mm_setr_ps(5.f/11, 6.f/13, 7.f/15, opmu*8/17);
const __m128 signMask = _mm_set1_ps(−0.f);
const __m128 one = _mm_set1_ps(1.f);

// Dot product of 4−tuples.
__m128 Dot (const __m128 tuple0, const __m128 tuple1)
{
    __m128 t0 = _mm_mul_ps(tuple0, tuple1);
    __m128 t1 = _mm_shuffle_ps(tuple1, t0, _MM_SHUFFLE(1,0,0,0));
    t1 = _mm_add_ps(t0, t1);
    t0 = _mm_shuffle_ps(t0, t1, _MM_SHUFFLE(2,0,0,0));
    t0 = _mm_add_ps(t0, t1);
    t0 = _mm_shuffle_ps(t0, t0, _MM_SHUFFLE(3,3,3,3));
    return t0;
}

// Common code for computing the scalar coefficients of SLERP.
__m128 Coefficient1 (const __m128 t, const __m128 xm1)
{
    __m128 sqrT = _mm_mul_ps(t, t);
    __m128 b0123, b4567, b, c;

    // (b4,b5,b6,b7) = (x-1)*(u4*t^2-v4, u5*t^2-v5, u6*t^2-v6, u7*t^2-v7)
    b4567 = _mm_mul_ps(u4567, sqrT);
    b4567 = _mm_sub_ps(b4567, v4567);
    b4567 = _mm_mul_ps(b4567, xm1);
    // (b7,b7,b7,b7)
    b = _mm_shuffle_ps(b4567, b4567, _MM_SHUFFLE(3,3,3,3));
    c = _mm_add_ps(b, one);
    // (b6,b6,b6,b6)
    b = _mm_shuffle_ps(b4567, b4567, _MM_SHUFFLE(2,2,2,2));
    c = _mm_mul_ps(b, c);
    c = _mm_add_ps(one, c);
    // (b5,b5,b5,b5)
    b = _mm_shuffle_ps(b4567, b4567, _MM_SHUFFLE(1,1,1,1));
    c = _mm_mul_ps(b, c);
    c = _mm_add_ps(one, c);
    // (b4,b4,b4,b4)
    b = _mm_shuffle_ps(b4567, b4567, _MM_SHUFFLE(0,0,0,0));
    c = _mm_mul_ps(b, c);
    c = _mm_add_ps(one, c);

    // (b0,b1,b2,b3) =
    //   (x-1)*(u0*t^2-v0, u1*t^2-v1, u2*t^2-v2, u3*t^2-v3)
    b0123 = _mm_mul_ps(u0123, sqrT);
    b0123 = _mm_sub_ps(b0123, v0123);
    b0123 = _mm_mul_ps(b0123, xm1);
    // (b3,b3,b3,b3)
    b = _mm_shuffle_ps(b0123, b0123, _MM_SHUFFLE(3,3,3,3));
    c = _mm_mul_ps(b, c);
    c = _mm_add_ps(one, c);
    // (b2,b2,b2,b2)
    b = _mm_shuffle_ps(b0123, b0123, _MM_SHUFFLE(2,2,2,2));
    c = _mm_mul_ps(b, c);
    c = _mm_add_ps(one, c);
    // (b1,b1,b1,b1)
    b = _mm_shuffle_ps(b0123, b0123, _MM_SHUFFLE(1,1,1,1));
    c = _mm_mul_ps(b, c);
    c = _mm_add_ps(one, c);
    // (b0,b0,b0,b0)
    b = _mm_shuffle_ps(b0123, b0123, _MM_SHUFFLE(0,0,0,0));
    c = _mm_mul_ps(b, c);
    c = _mm_add_ps(one, c);
    c = _mm_mul_ps(t, c);

    return c;
}

__m128 SlerpSSE1 (float t[1], const __m128 q0, const __m128 q1)
{
    __m128 x = Dot(q0, q1);  // cos(theta) in all components
    __m128 sign = _mm_and_ps(signMask, x);
    x = _mm_xor_ps(sign, x);
    __m128 localQ1 = _mm_xor_ps(sign, q1);
    __m128 xm1 = _mm_sub_ps(x, one);
    __m128 splatT = _mm_set1_ps(t);
    __m128 splatD = _mm_sub_ps(one, splatT);
    __m128 cT = Coefficient(splatT, xm1);
    __m128 cD = Coefficient(splatD, xm1);
    cT = _mm_mul_ps(cT, localQ1);
    cD = _mm_mul_ps(cD, q0);
    __m128 slerp = _mm_add_ps(cT, cD);
    return slerp;
}
```

References

[1] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications, Inc., 1965.

[2] Martin Braun. *Differential Equations and Their Applications: An Introduction to Applied Mathematics*. Springer-Verlag, NY, 4th edition, 1992. Texts in Applied Mathematics, Volume 11.

[3] David Eberly. A fast and accurate algorithm for computing SLERP. *Journal of Graphics, GPU, and Game Tools*, 15(3):161–176, 2011.
https://www.tandfonline.com/doi/full/10.1080/2151237X.2011.610255.

[4] Ken Shoemake. Animating rotation with quaternion calculus. ACM SIGGRAPH Course Notes 10, Computer Animation: 3-D Motion, Specification, and Control, 1997.

[5] Wikipedia. Chebyshev polynomials.
`https://en.wikipedia.org/wiki/Chebyshev_polynomials`.
accessed September 10, 2017.

[6] Wolfram Research, Inc. *Mathematica 11.3*. Wolfram Research, Inc., Champaign, Illinois, 2018.