Fast Slerp Summary

David Eberly
Geometric Tools, LLC
http://www.geometrictools.com/

January 9, 2012

This is a brief summary of my paper: *A Fast and Accurate Algorithm for Computing Slerp*, Journal of Graphics, GPU, and Game Tools, vol. 15, no. 3, pages 161-176. (http://www.tandfonline.com/loi/ujgt20)

The SLERP of two unit-length quaternions q_0 and q_1 is

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

for $t \in [0, 1]$. Defining $x = \cos(\theta)$ and $f(x, t) = \sin(t\theta)/\sin(\theta)$, the paper shows that

$$f(x, t) = \sum_{i=0}^{\infty} a_i(t)(x-1)^i$$

where

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

It is clear that $a_i(t)$ is a polynomial in t of degree $2i + 1$. It is also clear that $f(x, 0) = 0$ and $f(x, 1) = 1$.

The power series for $f(x, t)$ may be truncated to obtain an approximation that always underestimates the true value. However, the error may be balanced (centered about zero, positive and negative errors can occur)

$$f(x, t) = \sum_{i=0}^{n} a_i(t)(x-1)^i + \mu_n a_n(t)(x-1)^n + \varepsilon_n(x, t)$$

for some positive constant μ_n and where the last term $\varepsilon_n(x, t)$ is the approximation error. A bound on the error is of the form

$$|\varepsilon_n(x, t)| = \left| f(x, t) - \left(\sum_{i=0}^{n} a_i(t)(x-1)^i + \mu a_n(t)(x-1)^n \right) \right| \leq e_n$$

The following table shows choices for n and the corresponding μ_n and e_n.

n	μ_n	e_n
1	0.62943436108234530	$5.745259 \star 10^{-3}$
2	0.73965850021313961	$1.092666 \star 10^{-3}$
3	0.79701067629566813	$2.809387 \star 10^{-4}$
4	0.83291820510335812	$8.409177 \star 10^{-5}$
5	0.85772477879039977	$2.763477 \star 10^{-5}$
6	0.87596835698904785	$9.678992 \star 10^{-6}$
7	0.88998444919711206	$3.551215 \star 10^{-6}$
8	0.90110745351730037	$1.349968 \star 10^{-6}$
9	0.91015881189952352	$5.277561 \star 10^{-7}$
10	0.91767344933047190	$2.110597 \star 10^{-7}$
11	0.92401541194159076	$8.600881 \star 10^{-8}$
12	0.92944142668012797	$3.560875 \star 10^{-8}$
13	0.93413793373091059	$1.494321 \star 10^{-8}$
14	0.93824371262559758	$6.344653 \star 10^{-9}$
15	0.94186426368404708	$2.721482 \star 10^{-9}$
16	0.94508125972497303	$1.177902 \star 10^{-9}$

An implementation of the approximation for a standard floating-point unit is shown next for $n = 8$. It must compute both $f(x, t)$ and $f(x, 1-t)$ to produce $p = f(x, 1-t)q_0 + f(x, t)q_1$.

```
// Precomputed constants.
const float opmu = 1.90110745351730037f;
const float u[8] = // 1/[i(2i+1)] for i >= 1
{
    1.f/(1*3), 1.f/(2*5), 1.f/(3*7), 1.f/(4*9), 1.f/(5*11), 1.f/(6*13), 1.f/(7*15), opmu/(8*17)
};
const float v[8] = // i/(2i+1) for i >= 1
{
    1.f/3, 2.f/5, 3.f/7, 4.f/9, 5.f/11, 6.f/13, 7.f/15, opmu*8/17
};

FTuple4 SlerpFPU (float t, FTuple4 q0, FTuple4 q1)
{
    float x = q0.Dot(q1);  // cos(theta)
    float sign = (x >= 0 ? 1 : (x = -x, -1));
    float xm1 = x - 1;

    float d = 1 - t, sqrT = t*t, sqrD = d*d;

    float bT[8], bD[8];
    for (int i = 7; i >= 0; --i)
    {
        bT[i] = (u[i]*sqrT - v[i])*xm1;
        bD[i] = (u[i]*sqrD - v[i])*xm1;
    }

    float cT = sign*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]))))))));

    float cD = 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]))))))));

    FTuple4 slerp = q0*cD + q1*cT;
    return slerp;
}
```

The code uses the formula

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

in order to reduce the operation count. The evaluation of $f(x,t)$ becomes a nested sequence of multiplications and additions of 1. The u_i and v_i are constants known at compile time, so they are precomputed as shown in the code listing.

Source code is available online at http://www.geometrictools.com/JGT/FastSlerp.cpp and contains the FPU-based implementation and various Intel SSE2 implementations.