#### qstring

Chris Larsen - larsencr@gmail.com

String manipulation utility for analytically multplying quaternions.

Copyright: The code presented will eventually be released publicly. It's free for non-commercial use and terms of commercial use can be negotiated but in either case the user must retain my copyright notice.

#### **Euler's Rotation Theorem**

- If two rotations are performed there is a single rotation which provides the same net effect.
- Improve code efficiency by collapsing multiple, consecutive rotations into a single rotation.

#### Procedure

- Convert both rotations to quaternions
  - $-\theta$  is the angle of rotation,
  - V is the rotation vector, normalized if necessary.
- Then the quaternion  $\cos(\theta/2) + \overline{v}_x \sin(\theta/2)i + \overline{v}_y \sin(\theta/2)j + \overline{v}_z \sin(\theta/2)k$  represents the rotation.
- Code samples below use C++ with quaternion class definition.

#### Example

Start with the code

```
GFX_rotate(-90.0f, 1.0f, 0.0f, 0.0f);
GFX_rotate(-rotz, 0.0f, 0.0f, 1.0f);
```

Replace with version of GFX\_rotate()
 which uses quaternions:

### Example (cont'd)

Reduce to a single rotation:

### How to Simplify?

- The first quaternion is
  - a constant, and
  - the sine and cosine of its half angle can be calculated analytically
- θ is -90° so θ/2 is -45°
- cosine(-45°) is 1/sqrt(2), and sin(-45°) is -1/sqrt(2). C/C++ provides the math constant M\_SQRT1\_2 so q0 is const quaternion q0 (M\_SQRT1\_2,

```
const quaternion q0(M_SQRT1_2,
-M_SQRT1_2, 0.0f, 0.0f);
```

# How to Simplify? (cont'd)

Code becomes

- Eliminated two calls to transcendental functions (sine & cosine), and these are generally expensive.
- Still can do better

# Quaternion Multiplication

- In the general case, quaternion multiplication requires
  - 16 floating point multiplications, and
  - 12 floating point additions/subtractions
- Each of the quaternions has two zero components so many of the multiplications don't need to be done.
- Since many of the intermediate products are zero, many of the additions aren't needed.

# Quaternion Multiplication (cont'd)

Computing the product directly the code becomes

- This reduces the quaternion multiplication to
  - 4 floating point multiplications, and
  - 0 additions/subtractions

### But that was only two ...

- This was a simple case, only two quaternions (with a lot of zeroes) needed to be multiplied out by hand
- What about when things get more complicated?

#### Three Quaternions

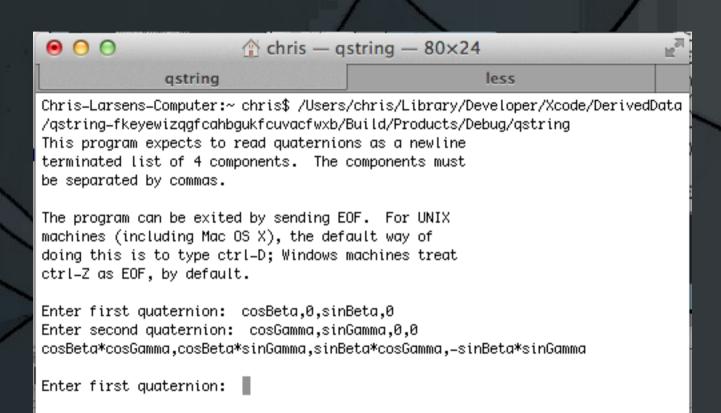
 I don't really want to calculate q0\*q1\*q2 by manually:

```
float alpha(rotz * DEG_TO_RAD_DIV_2);
float cosAlpha(cosf(alpha)),
        sinAlpha(sinf(alpha));
quaternion q0(cosAlpha, 0.0f, 0.0f, sinAlpha);
float beta(roty * DEG_TO_RAD_DIV_2);
float cosBeta(cosf(beta)), sinBeta(sinf(beta));
quaternion q1(cosBeta, 0.0f, sinBeta, 0.0f);
float gamma(rotx * DEG_TO_RAD_DIV_2);
float cosGamma(cosf(gamma)),
        sinGamma(sinf(gamma));
quaternion q2(cosGamma, sinGamma, 0.0f, 0.0f);
GFX_rotate(q0*q1*q2);
```

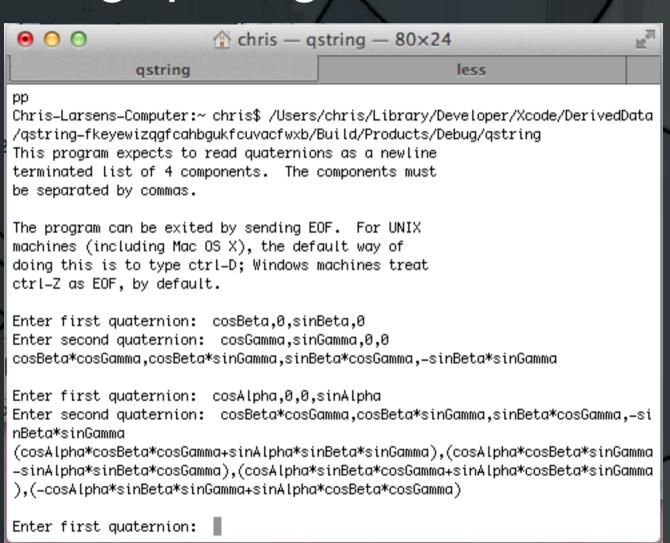
### Automation!

- Wrote a program called qstring
  - Reads two quaternions,
  - Calculates product analytically,
  - Deals with special product cases
    - When either multiplier or multiplicand is zero or one.

# Using qstring - 1<sup>st</sup> Product



# Using qstring - 2<sup>nd</sup> Product



## Actual Code Using This Example

```
↑ chris — less — 80×24

                bash
                                                       less
void create_direction_vector( vec3 *dst, vec3 *up_axis, float rotx, float roty,
float rotz )
   TStack 1:
   // Convert all angles to radians & divide by 2.
   float
           alpha = rotz*DEG_TO_RAD_DIV_2;
           cosAlpha(cosf(alpha)), sinAlpha(sinf(alpha));
   float
           beta = roty*DEG_TO_RAD_DIV_2;
   float
   float
           cosBeta(cosf(beta)), sinBeta(sinf(beta));
           gamma = rotx*DEG_TO_RAD_DIV_2;
   float
           cosGamma(cosf(gamma)), sinGamma(sinf(gamma));
   float
           cAcB(cosAlpha*cosBeta);
   float
   float
           sAsB(sinAlpha*sinBeta);
           cAsB(cosAlpha*sinBeta);
   float
           sAcB(sinAlpha*cosBeta);
   float
   I.loadRotation(quaternion(cAcB*cosGamma+sAsB*sinGamma,
                             cAcB*sinGamma-sAsB*cosGamma,
                             cAsB*cosGamma+sAcB*sinGamma.
                             sAcB*cosGamma=cAsB*sinGamma)):
   *up_axis = -*up_axis:
   *dst = vec3(vec4(*up_axis, 0.0f) * l.back(), true);
```

#### Cost Calculation

- General case quaternion multiplication:
  - 32 floating point multiplies
  - 24 floating point additions
- Special case multiplication with 50% zeroes:
  - 12 floating point multiplies
  - 4 floating point additions

# GitHub Repository

http://github.com/crlarsen/qstring