Quaternions and Spherical Linear Interpolation (SLERP) for animation

CS-116B Computer Graphics Algorithms

Quaternions: Advantages

1. **Smooth interpolation**

   "The interpolation provided by SLERP provides smooth interpolation between orientations" (p. 263).

2. **Fast concatenation and inversion of angular displacements**

   "We can concatenate a sequence of angular displacements into a single angular displacement by using the quaternion cross product" (p. 263).

3. **Fast conversion to and from matrix form**

   "…quaternions can be converted to and from matrix form a bit faster than Euler angles" (p. 263).

4. **Only four numbers**

   "Since a quaternion contains four scaler values, it is considerably more economical than a matrix, which uses nine numbers" (p. 263).

Source: Dunn & Parberry, 2011, p. 263.

Quaternions: Disadvantages

1. **Slightly bigger than Euler angles**

   “That one additional number may not seem like much, but an extra 33% can make a difference when large amounts of angular displacement are needed, for example, when storing animation data” (p. 263).

2. **Can become invalid**

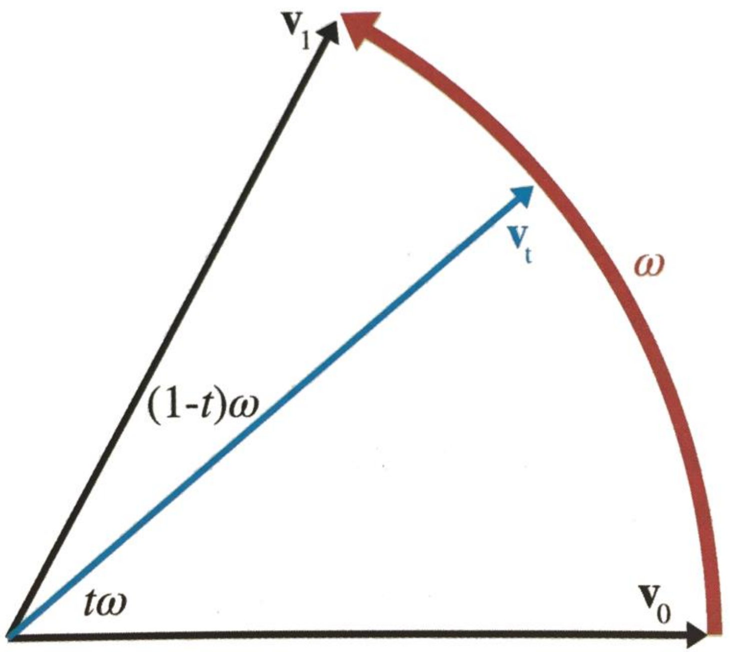
   “This can happen either through bad input data, or from accumulated floating point roundoff error” (p. 264).

3. **Difficult for humans to work with**

   “Of the three representation methods, quaternions are the most difficult for humans to work withy directly” (p. 264).

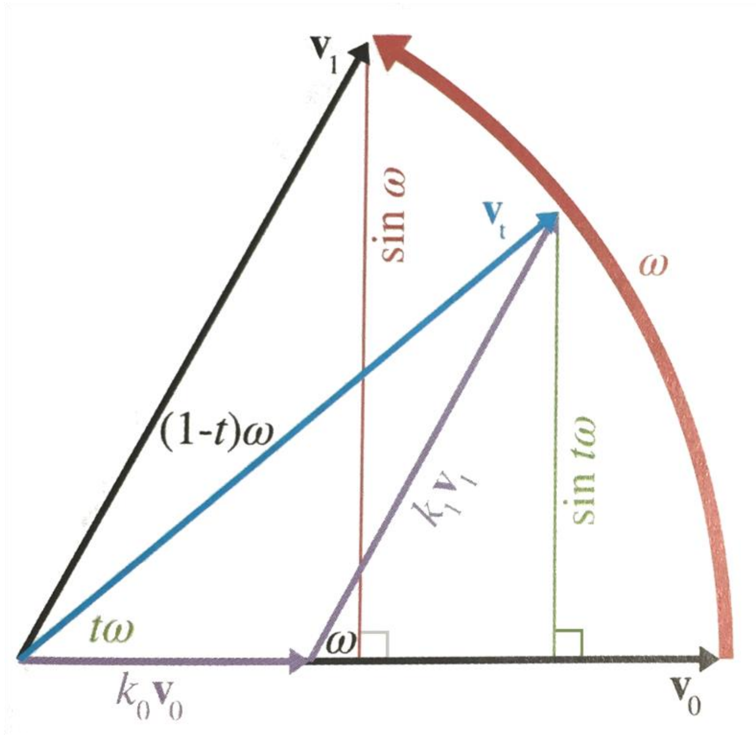
Source: Dunn & Parberry, 2011, pp. 263-264.

Interpolating a vector about an arc



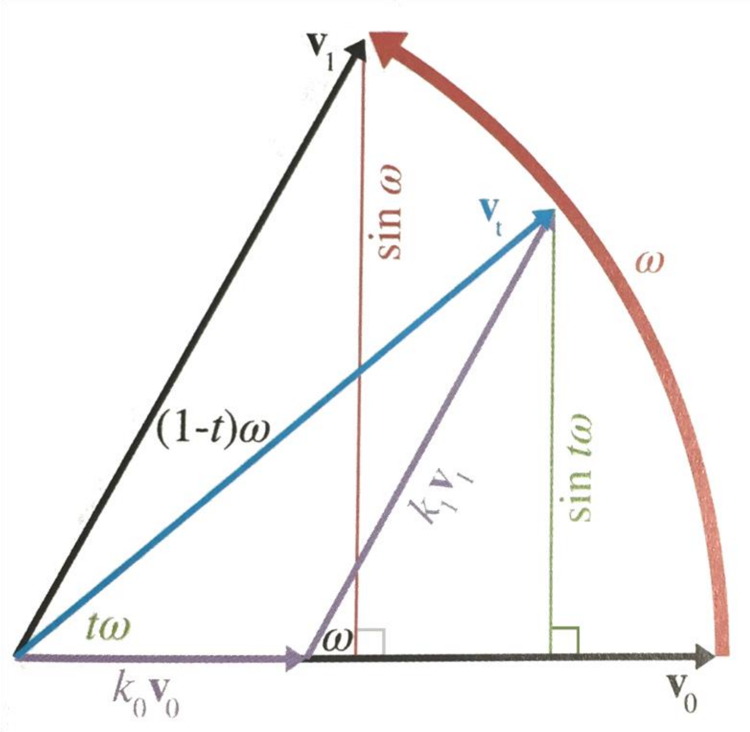
Source: Dunn & Parberry, 2011, p. 260.

Interpolating a vector about an arc



Source: Dunn & Parberry, 2011, p. 260.

Interpolating a vector about an arc



$$sin\ \omega = \frac{\sin t\omega}{k_1}$$

$$k_1 = \frac{\sin t\omega}{\sin \omega}$$

$$k_0 = \frac{\sin(1-t)\omega}{\sin \omega}$$

$$v_{\mathrm{t}} = k_0 v_0 + k_1\, v_1 = \frac{\sin(1-t)\omega}{\sin \omega}\, v_0 + \frac{\sin t\omega}{\sin \omega}\, v_1$$

$$\text{slerp}(q_0, q_1, t) = \frac{\sin(1-t)\omega}{\sin \omega}\, q_0 + \frac{\sin t\omega}{\sin \omega}\, q_1$$

Source: Dunn & Parberry, 2011, p. 260.

Quaternion SLERP

```
// The two input quaternions
float w0, x0, y0, z0;
float w1, x2, y1, z1;
float t; // The interpolation parameter
float w, x, y, z; // The output quaternion will be computed here
float cosOmega = w0*w1 + x0*x1 + y0*y1 + z0*z1; // Compute the "coside of the angle" between the quaternions using the dot product
if (cosOmega < 0.01) // if negative dot, negate one of the input quaternions to take the shorter 4D "arc"
{
  w1 = -w1;
  x1 = -x1;
  y1 = -y1;
  z1 = -z1;
  cosOmega = -cosOmega;
}
float k0, k1;
If (cosOmega > 0.9999f) // Check if they are very close together to protect against divide-by-zero
{
  k0 = 1.0f - t; // very close - just use linear interpolation
  k1 = t;
}
else
{
  float sinOmega = sqrt(1.0f - cosOmega * cosOmega)
  float omega = atan2(sinOmega, cosOmega);
  float oneOverSinOmega = 1.0f / sinOmega;
  k0 = sin((1.0f - t * omega) * oneOverSinOmega;
  k1 = sin(t * omega) * oneOverSinOmega;
}
w = w0 * k0 + w1 * k1;
x = x0 * k0 + x1 * k1;
y = y0 * k0 + y1 * k1;
z = z0 * k0 + z1 * k1;
```

Source: Dunn & Parberry, 2011, pp. 262-263.

References

Dunn, F. & Parberry, I. (2011). *3D Math Primer for Graphics and Game Development*. (2nd Edition). New York: CRC Press.