Quaternions

Warning – This lecture contains maths

Viewer discretion is advised

3D Rotations – Euler Angles

- Euler angles (pronounced 'Oil-er') define a rotation in 3 parts:
  - Pitch,Yaw and Roll (sometimes X Y Z)
  - Treated as 3 numbers expressing rotation around each of the axis
  - The rotations are applied one after the other
  - You should always apply the rotations in the same order else you will encounter problems!
- Gimbal Lock problems:
  - Euler angles evaluate each axis independently in a set order
  - As each axis is processed it is not carried along to the next rotation
  - Thus if X is processed, then Y, then Z, there is a chance Y or Z end up facing in the same direction as X!
  - For an animated example of the problem: http://www.anticz.com/eularqua.htm

Quaternions

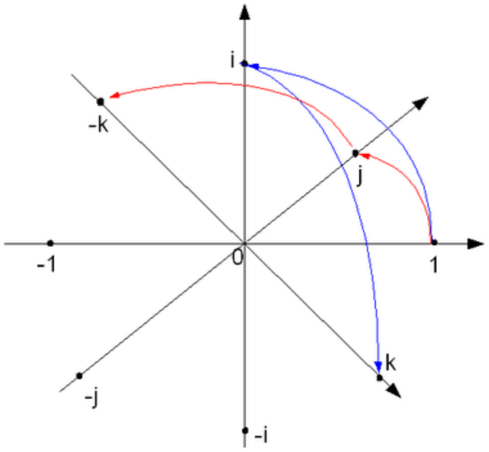
- Quaternions are a 3rd way to represent 3D rotations:
  - They are a form of complex number and can be hard to understand
  - Representations of rotations by quaternions are more compact and faster to compute than representations by matrices and unlike Euler angles are not susceptible to Gimbal Lock

- Consist of 1 scalar part and 1 vector part
  - The scalar part is known as a real dimension, while the vector part is 3 imaginary dimensions

- We can try to visualise a quaternion as a unit vector and a rotation around that vector:
  - Although that is not what a quaternion actually is, for our purposes in computer graphics it is easier to visualise it as such

Quaternions

- A quaternion has the form:
  - $q = w + xi + yj + zk$
    where:
    $i^2 = j^2 = k^2 = ijk = -1$
  - i, j, and k are the imaginary dimensions
  - w, x, y and z in our case all relate to the rotations about those imaginary dimensions
  - We only have to deal with the scalar w and the vector [x y z] when we use quaternions in computer graphics



Graphical representation of quaternion units product as 90°-rotation in 4D-space

ij = k
ji = -k
ij = -ji

Quaternions from Axis/Angle

- We can easily create a quaternion from an axis and a rotation around that axis

- The scalar w relates to the angle of rotation
  - w = cos(Ø / 2)

- The vector component [x y z] is related to the axis of rotation
  - [x y z] = axis * sin(Ø / 2)
  - [x y z] = [axisX * sin(Ø / 2), axisY * sin(Ø / 2), axisZ * sin(Ø / 2)]
  - q = cos(Ø / 2) +
i (x * sin(Ø / 2)) +
j (y * sin(Ø / 2)) +
k (z * sin(Ø / 2))

Quaternion Vector Similarities

- Quaternions have some similar attributes to 4D vectors
  - And not just because they also have x y z w elements

- Quaternions can calculate their Dot Product like Vectors:
  - Result = q0 ● q1 = q0.w * q1.w + q0.x * q1.x + q0.y * q1.y + q0.z * q1.z

- They can also calculate their magnitude, which for quaternions is called the Norm:
  - Notation is $||q||$
  - $||q|| = \sqrt{w^2 + x^2 + y^2 + z^2}$
  - A quaternion on the unit sphere has a norm of 1

Quaternion Multiplication

- There are 2 key advantages to using a quaternion to represent rotations rather than a matrix:
  - Less memory required (4 scalars rather that 9 for a 3x3 matrix)
  - Multiplication uses almost half the number of multiply and add operators

- Quaternion multiplication is tricky, but less operators is always a plus:
  - For the theory check the references
  - Like a matrix the resultant quaternion combines the initial two
  - Like matrices, A * B != B * A, but in fact, A * B = C and B * A = -C!
  - q3 = q1 * q2
  - q3 = (q1.w * q2.w – q1.x * q2.x – q1.y * q2.y – q1.z * q2.z) +
    i (q1.w * q2.x + q1.x * q2.w + q1.y * q2.z + q1.z * q2.y) +
    j (q1.w * q2.y – q1.x * q2.z – q1.y * q2.w – q1.z * q2.x) +
    k (q1.w * q2.z + q1.x * q2.y + q1.y * q2.x + q1.z * q2.w)

Quaternion Vector Rotation

- Quaternions can also be used to rotate vectors, but first we need to understand another part of quaternions:
  - Quaternion Conjugate
- Conjugate is simply a quaternion with the sign of the imaginary parts reversed:
  - Notation is q^* or q^t
  - If $q = w + xi + yj + zk$, then:
  - $q^* = w - xi - yj - zk$
- We can rotate a vector by treating it as a quaternion (with a w component of 0) and pre-multiplying it with the quaternion, then post-multiplying by the conjugate of the same quaternion:
  - $v2 = q \times v \times q^*$

Academy of Interactive Entertainment
www.aie.edu.au

Canberra Institute of Technology
Workplace • Online • Campus | cit.edu.au

Quaternion Vector Rotation

- Understanding how it works can be rather complex:
  - You should primarily understand how to use it rather than the complex math theory behind it
  - Check the references for the theory at your own peril!

- One thing to note is that although multiplying 2 quaternions is faster than multiplying 2 matrices, transforming a vector by a quaternion is slower than a matrix!
  - 3x3 Matrix X 3x3 Matrix = 27 multiply, 18 add/subtract = 45 operations
  - Quaternion X Quaternion = 16 multiply, 12 add/subtract = 28 operations
  - 3x3 Matrix X Vector = 9 multiply, 6 add/subtract = 15 operations
  - Quaternion X Vector = 21 multiply, 18 add/subtract = 39 operations!

Spherical Interpolation

- Spherical Interpolation (or Slerp) can be used to smoothly interpolate between two quaternions. Slerp has the following properties
  - *torque-minimal path*
  - *non-commutative*
  - *Expensive – requires the use of sin, cos and acos*

Spherical Interpolation

```
inline quaternion& quaternion::slerp(quaternion a_q1, quaternion a_q2, float a_fT)
{
        float angle = a_q1.dotProduct(a_q2);

        // make sure we use the short rotation
        if (angle < 0.0f)
        {
                q1 *= -1.0f;
                angle *= -1.0f;
        }

        if (angle <= 1.0 - 0.00001) //If rotation is really small, just lerp
        {
                const float theta = acosf(angle);
                const float invsintheta = reciprocal(sinf(theta));
                const float scale = sinf(theta * (1.0f- a_fT)) * invsintheta;
                const float invscale = sinf(theta *  a_fT) * invsintheta;
                return (*this = (a_q1*scale) + (a_q2*invscale));
        }
        else // linear interploation
                return lerp(q1,q2,time);
}
```

Quaternion To Matrix

- Computer graphics uses dozens / hundreds / thousands of matrices every update, 60 updates per second (ideal)
- If we switch them all to quaternions instead then we would gain a massive performance increase right?:
  - Yes, but no
  - Quaternions don’t specify scale or translation like a 4D matrix
  - GPU hardware deals with matrices, not quaternions
- We can still make use of quaternions and gain an advantage though:
  - Quaternion + Scale + Translation is still less scalars than a 4D matrix
  - If we just need to define rotations (skeleton bone orientations for example) or want to define fluid camera rotations without rotation issues

Quaternion To Matrix

- Concatenating quaternions and then converting the result to a matrix is slower than just a matrix multiplied by a matrix, but...
  - If we deal with thousands of concatenations and then only convert once we still have a performance gain
  - We can convert a quaternion to a matrix with the following formula:

$$\begin{bmatrix} 1-(2*y^2-2*z^2) & 2*x*y-2*z*w & 2*x*z+2*y*w \\ 2*x*y+2*z*w & 1-(2*x^2-2*z^2) & 2*y*z-2*x*w \\ 2*x*z-2*y*w & 2*y*z+2*x*w & 1-(2*x^2-2*y^2) \end{bmatrix}$$

Matrix To Quaternion

- You can also easily build a quaternion from a matrix with the following formula, but...
  - The matrix needs to be orthogonal
  - The matrix axis must be unit length (no scale)

$$
\begin{aligned}
w &= \sqrt{(1 + m00 + m11 + m22)} \;/\; 2 \\
x &= (m21 - m12) / (4 * w) \\
y &= (m02 - m20) / (4 * w) \\
z &= (m10 - m01) / (4 * w)
\end{aligned}
$$

Conclusion

- Quaternions are a mathematically complex way to represent a rotation, but practically they are simple and efficient
- They don't suffer from Gimbal Lock, which is important as most art tools deal with Euler Angles
- Understanding how to use quaternions is more essential that understanding the complex theory
- References:
  - http://en.wikipedia.org/wiki/Quaternion
  - http://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation
  - http://en.wikipedia.org/wiki/Euler_angles