Quaternions

Defining and interpolation rotations with Complex Numbers

Programming – Computer Graphics

SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Contents

- 3-D Rotations
  - Matrices
  - Euler Angles
  - Quaternions
- Quaternions
  - From Axis / Angle
  - Multiplication
  - Vector Rotations
  - To Matrices
  - From Matrices
  - Interpolation

3-D Rotations - Matrices

- In 3-D we can use 3x3, 4x3 and 4x4 matrices to define rotations
  - A 3x3 matrix can store rotation and scale
    - Contains X, Y and Z axis, each with xyz elements
    - Axis length defines scale along an axis, while direction specifies rotation
  - A 4x3 matrix stores translation as well as rotation and scale
    - Contains X, Y, Z and Translation axis
    - Can't be used for homogeneous multiplications without custom functions!
  - A 4x4 matrix stores rotation, translation and scale
    - Each axis has a W element
    - X, Y and Z axis have W of 0 while Translation axis has W of 1

$$\begin{bmatrix} Xx & Yx & Zx \\ Xy & Yy & Zy \\ Xz & Yz & Zz \end{bmatrix}$$

$$\begin{bmatrix} Xx & Yx & Zx & Tx \\ Xy & Yy & Zy & Ty \\ Xz & Yz & Zz & Tz \end{bmatrix}$$

$$\begin{bmatrix} Xx & Yx & Zx & Tx \\ Xy & Yy & Zy & Ty \\ Xz & Yz & Zz & Tz \\ Xw & Yw & Zw & Tw \end{bmatrix}$$

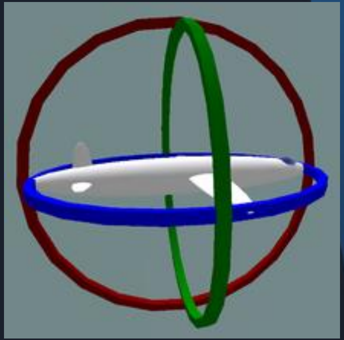
3-D Rotations – Euler Angles

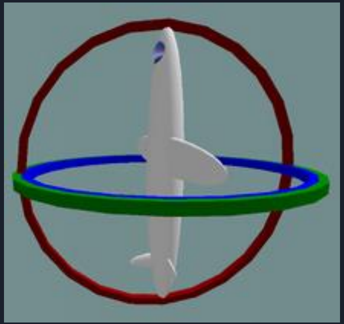
- Euler Angles (pronounced 'Oil-er') define a rotation in 3 parts:
  - Pitch, Yaw and Roll (sometimes X Y Z)
  - Treated as 3 numbers expressing rotation around each of the axis
  - The rotations are applied one after the other
  - You should always apply the rotations in the same order else you will encounter problems!
    - Pitch, Yaw then Roll does not equal the same as Yaw, Pitch then Roll

- We can implement Euler Angles using 3 concatenated matrix rotation transforms
  - There are ways to implement without needing to define then multiply the matrices, with the same results

Gimbal Lock – Combined Transform Problem

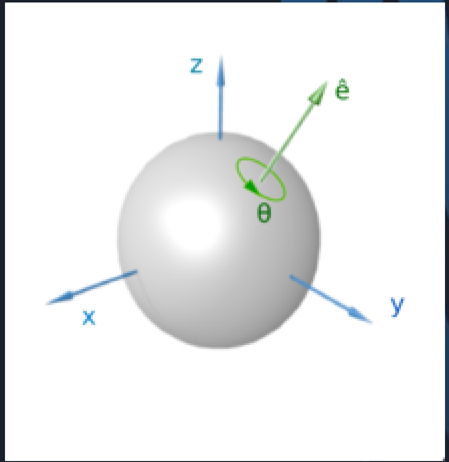
- Euler angles evaluate each axis independently in a set order
  - As each axis is processed it is not carried along to the next rotation
  - Thus if X is processed, then Y, then Z, there is a chance Y or Z end up facing in the same direction as X!
- This problem is called Gimbal Lock
  - It prevents us from being able to properly combine transforms with Euler Angles or Matrices, as over time the transform can end up with axis locked to each other
- For an animated example of the problem: http://www.anticz.com/eularqua.htm

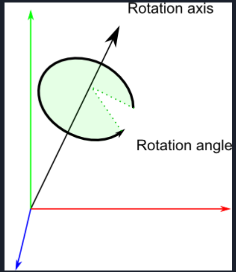




3-D Rotations - Quaternions

- Quaternions are a 3rd way to represent 3-D rotations
  - They are a form of complex number and can be hard to understand
  - For our purposes they represent a rotation around the surface of a unit sphere
- 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 think of it as such

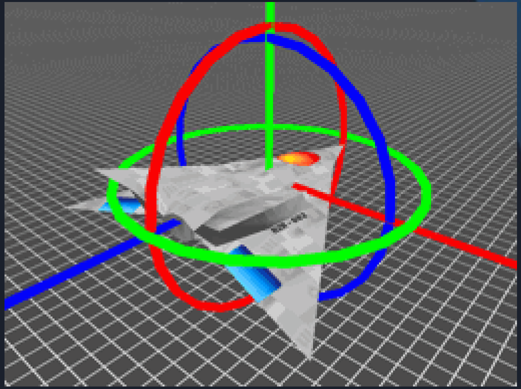




aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Quaternions

- Quaternions can be used to represent rotations only
  - No scale or translation
- They have benefits over Euler Angles and Matrices
  - Don’t suffer from Gimbal Lock
  - Use 4 elements (4x float usually) rather than 9 (3x3 matrix) or 16 (4x4 matrix)
  - Can be concatenated via multiplication like matrices, but faster to calculate
  - Can be interpolated from one Quaternion to another
  - Because they can be interpolated they are great for animation!



Quaternion Notation

- Mathematically quaternions seem confusing
- A quaternion has the mathematical form:

$$q = w + ix + jy + kz$$

  - 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 around 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

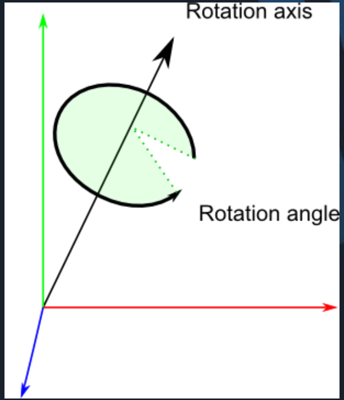
Quaternions from Axis / Angle

- Creating and using a Quaternion is much easier
  - A quaternion is like a vector, typically with W, X, Y and Z values

$$q = w + ix + jy + kz$$
$$q = [w, x, y, z]$$

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

$$V = axis(x, y, z)$$
$$\theta = angle\ in\ radians$$
$$w = \cos(\frac{\theta}{2})$$
$$s = \sin(\frac{\theta}{2})$$
$$q = w + i(V_x s) + j(V_y s) + k(V_z s)$$
$$q = [w, V_x s, V_y s, V_z s]$$



Quaternion Vector Similarities

- Quaternions have some similar attributes to 4-D vectors
  - And not just because they also have X, Y, Z and W elements

- Quaternions can calculate their Dot Product like Vectors:

$$d = q^1 \cdot q^2 = q_w^1 \times q_w^2 + q_x^1 \times q_x^2 + q_y^1 \times q_y^2 + q_z^1 \times q_z^2$$

- 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
  - 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
  - Multiplication concatenates the rotations, and like a Matrix A x B != B x A
  - But A x B = C and B x A = -C!

$$q = q^1 \times q^2$$

$$\begin{aligned} q = {} & \left(q_w^1 \times q_w^2 - q_x^1 \times q_x^2 - q_y^1 \times q_y^2 - q_z^1 \times q_z^2\right) + \\ & i\left(q_w^1 \times q_x^2 + q_x^1 \times q_w^2 + q_y^1 \times q_z^2 + q_z^1 \times q_y^2\right) + \\ & j\left(q_w^1 \times q_y^2 - q_x^1 \times q_z^2 - q_y^1 \times q_w^2 - q_z^1 \times q_x^2\right) + \\ & k\left(q_w^1 \times q_z^2 + q_x^1 \times q_y^2 + q_y^1 \times q_x^2 + q_z^1 \times q_w^2\right) + \end{aligned}$$

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:
  - $q^t = w - xi - yj - zk$

- We can rotate a vector by treating it as a quaternion (with a W component of 0 since it is a vector and not a point) and pre-multiplying it with the quaternion, then post-multiplying by the conjugate of the same quaternion:
  - $v2 = q \times v \times q^t$
  - The result is the vector rotated by the quaternion, just like a rotation matrix

Quaternion Vector Rotation

- 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 Vector3 = 9 multiply, 6 add/subtract = 15 operations
  - 4x4 Matrix X Vector4 = 16 multiply, 12 add/subtract = 28 operations
  - Quaternion X Vector4 = 21 multiply, 18 add/subtract = 39 operations!

Quaternion To Matrix

- Computer graphics uses dozens / hundreds / thousands of matrices every update, 60 updates per second (or more!)
  - Matrices transforming matrices

- 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 4-D 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 4x4 Matrix
  - If we just need to define rotations (skeleton bone orientations for example) or want to define smooth camera rotations without Gimbal Lock issues

Quaternion To Matrix

- We can convert Quaternions to Matrices and vice versa
- 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 quaternion concatenations and then only convert to a matrix once, we still have a performance gain!
  - We can convert a quaternion to a 4x4 matrix with the following formula:

$$q = wxyz$$

$$\begin{bmatrix} 1-(2y^2-2z^2) & 2xy-2zw & 2xz+2yw & 0 \\ 2xy+2zw & 1-(2x^2-2z^2) & 2yz-2xw & 0 \\ 2xz-2yw & 2yz+2xw & 1-(2x^2-2y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrix To Quaternion

- You can also easily build a quaternion from a matrix with the following formula
  - The matrix needs to be orthogonal
  - The matrix must have unit length axis, so no scale
  - Using the following matrix we can fill in the quaternion values to the right
    - Works the same using a 3x3 matrix as we don’t use the 4th row or column

$$\begin{bmatrix} m00 & m01 & m02 & m01 \\ m10 & m11 & m12 & m13 \\ m20 & m21 & m22 & m23 \\ m30 & m31 & m32 & m33 \end{bmatrix}$$

$$w = \frac{\sqrt{1 + m00 + m11 + m22}}{2}$$

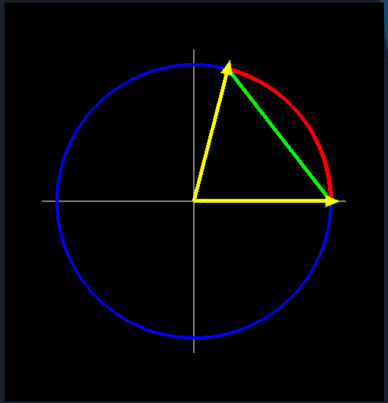
$$x = \frac{m21 - m12}{4w}$$

$$y = \frac{m02 - m20}{4w}$$

$$z = \frac{m10 - m01}{4w}$$

Quaternion Interpolation - SLERP

- One of the biggest advantages of a Quaternion over a Rotation Matrix or Euler Angles is the ability to interpolate one rotation to another
  - The rotation maintains its unit length as it rotates
  - Interpolating the axis of a matrix will skew the matrix
  - Interpolating Euler angles may introduce Gimbal Lock



- Quaternions rotate along the surface of a unit sphere
  - The method is called Spherical Linear Interpolation, aka SLERP
- There are a few other methods for interpolating Quaternions, but SLERP is the most common
  - Like a Linear Interpolation (LERP) it uses a start and an end quaternion and calculates a quaternion that lies between them using a value in the range [0,1]

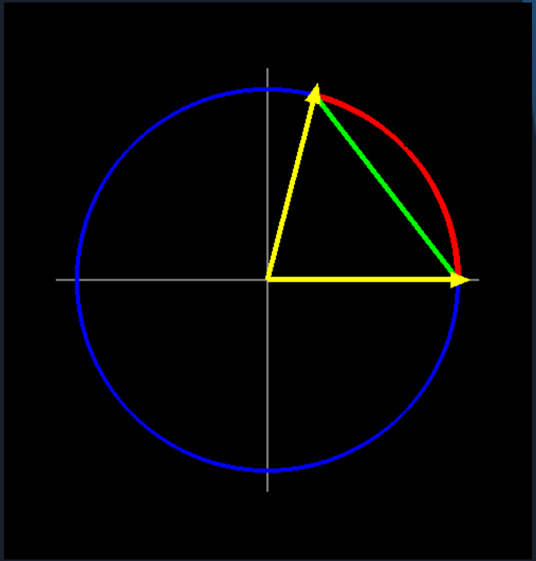
Quaternion Interpolation - SLERP

```
quaternion slerp(const quaternion& q1, const quaternion& q2, float t) {

    quaternion q3;
    float d = dot(q1, q2);

    /*  d = cos(theta)
        if (d < 0), q1 and q2 are more than 90 degrees apart,
        so we can invert one to reduce spinning   */
    if (d < 0) {
        d = -d;
        q3 = -q2;
    }
    else
        q3 = q2;

    if (d < 0.95f) {
        float angle = acosf(d);
        return (q1 * sinf(angle * (1-t)) + q3 * sinf(angle * t)) / sinf(angle);
    }
    else // if the angle is small, use linear interpolation
        return lerp(q1,q3,t);
}
```



Summary

- Quaternions are a mathematically complex way to represent a rotation
  - But practically, they are simple and efficient
- They don't suffer from Gimbal Lock
  - An important fact, as most art tools deal with Euler Angles
- Understanding how to use quaternions is more essential than understanding the complex theory behind them

Further Reading

- Dunn, F, Parberry, I, 2011, *3D Math Primer for Graphics and Game Development*, 2nd Edition, CRC Press
- Lengyel, E, 2011, *Mathematics for 3D Game Programming and Computer Graphics*, 3rd Edition, Cengage Learning
- Eberly, D, *Quaternion Algebra and Calculus*, http://www.geometrictools.com/Documentation/Quaternions.pdf , Last viewed 13/02/2015
- Blow, J, *Hacking Quaternions*, http://number-none.com/product/Hacking%20Quaternions/index.html , Last viewed 13/02/2015

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX