Game Engineering

CS420-2016S-08

Orientation & Quaternions

David Galles

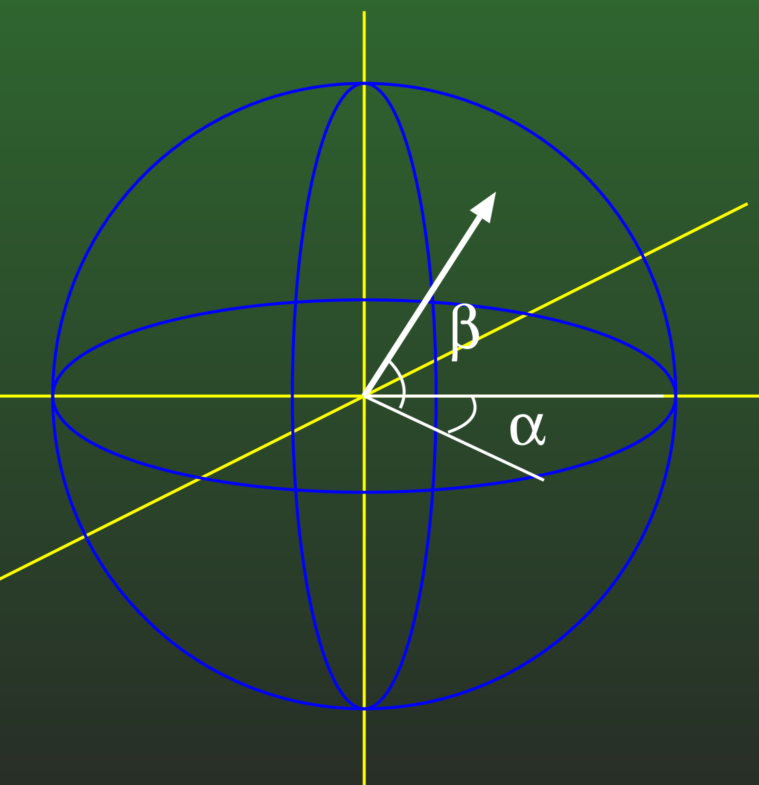
Department of Computer Science
University of San Francisco

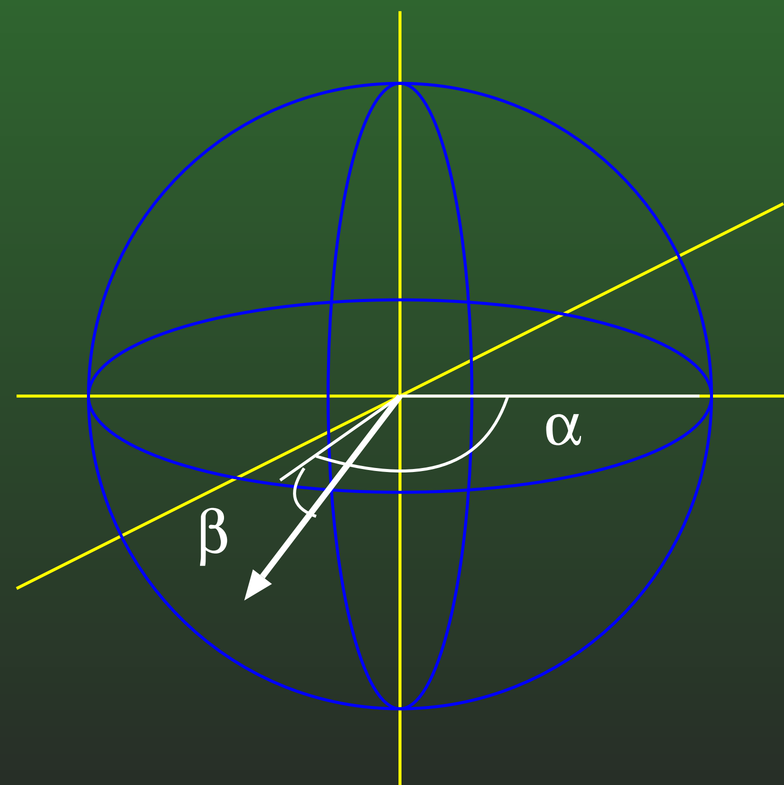
08-0: Orentation

- Orientation is *almost* the direction that the model is pointing.
- We can describe the *direction* that a model is pointing using two numbers, polar coordinattes

Direction in Polar Cooridnates

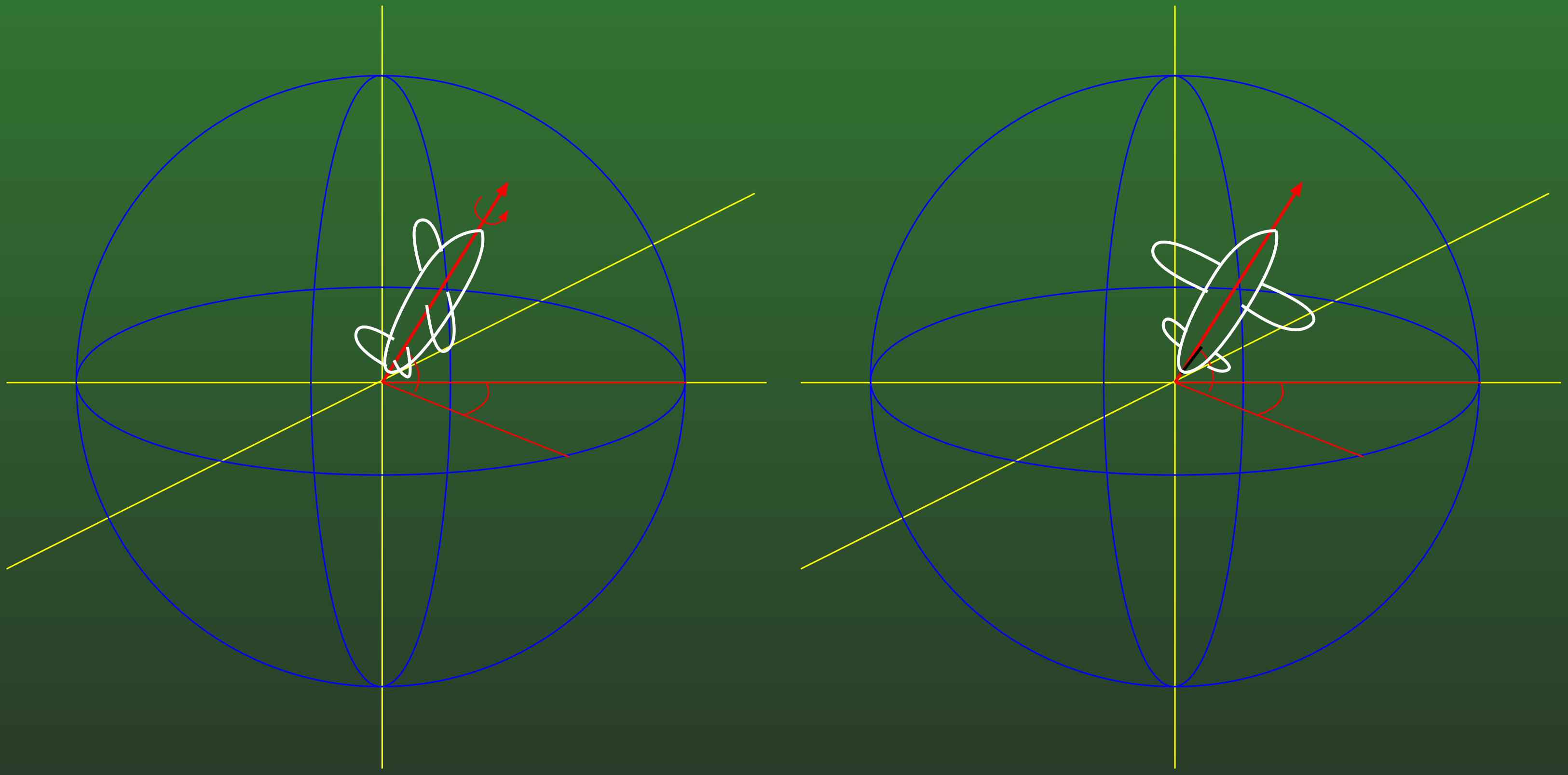
- We can describe *direction* using two values (α and β)
- What's missing for orientation?





08-2: Orientation

- Orientaion needs at least 3 numbers to describe



08-3: Absoulte Orientation?

- Recall vectors are actually a displacement, not a position
  - Need a reference point (origin) to get a position
  - If we think in terms of displacement instead of absoulte position, multiple reference frames easier to understand
- Orientation is the same way
  - Think of orentation as ΔOrientation instead of absoulte
  - Use Δ from a fixed reference frame (like origin) to get absoulte orientation

08-4: Absoulte Orientation?

- Of course, if our orientation is a Δ, we need to be careful about what kind of change
  - Change from Object space to Inertial Space?
  - Change from Inertial Space to Object Space?
- If we use Matrices, then one is the inverse of the other
- Rotational matrices are orthoginal, finding inverse is easy – Transpose

08-5: Matrices as Orientation

- We can represent orientation using 3x3 matrices
  - Delta from Object Space to Inertial Space

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

  - Delta from Inertial Space to Object Space

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{21} & m_{31} \\ m_{12} & m_{22} & m_{32} \\ m_{13} & m_{23} & m_{33} \end{bmatrix}$$

08-6: 4x4 Matricies

- We can completely describe the position and orientation of an object using a 4x4 matrix
  - Alternately, a 3x3 matrix and a 1x3 (or 3x1) position vector
- When we use matrices, we are using Δorientation and Δposition
- Easily combine two matrices – just a matrix multiplication

08-7: Matrix Problems

- Matricies are great for describing orientation and position
  - Easy to combine
  - How orientation is described within most graphics engines, and by OpenGL and DirectX
- What are some drawbacks to using matrices for orientation?

08-8: Matrix Problems

- Requires 9 numbers instead of 3
  - Uses more space
  - Not all matrices are valid rotational matrices
    - What happens when you use more values that you have degrees of freedom
    - Overconstraint problems

08-9: Matrix Problems

- Requires 9 numbers instead of 3
  - Consider a 3x3 matrix
  - The z basis vector (3 numbers) gives the direction
  - x basis vector needs to be parallel to z – we can describe the relative position of the x basis vector given the z basis vector with a single number (not 3!)
  - Once we have x and z, y is completely determined!

08-10: Matrix Problems

- Only orthoginal matrices are valid rotational matrices
  - Matrices can become non-orthoginal via matrix creep
  - Data can be a little off if not cleaned up properly (though the solution to that is to clean up your data!)
  - Can fix this problem by orthogonalizing matrices (as per last lecture)

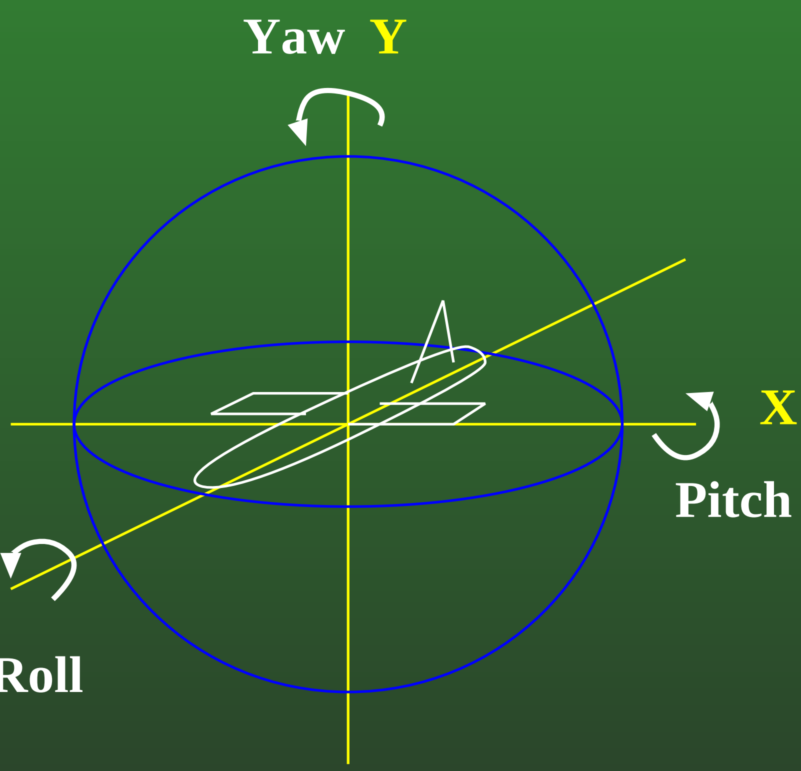
08-11: Matrix Problems

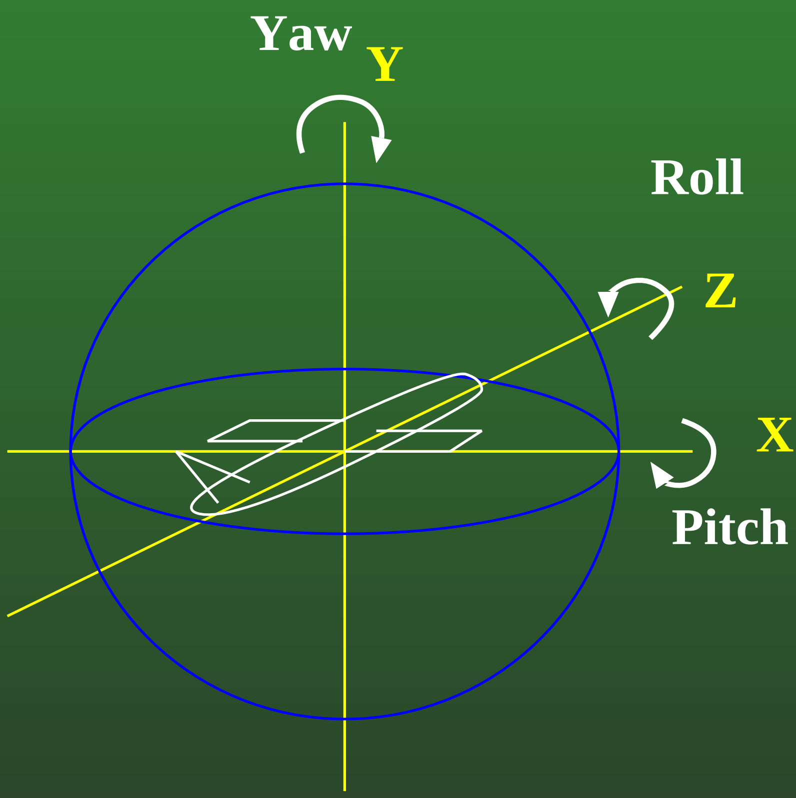
- Go to your artist / animator
- Tell him / her that all angles need to be described in terms of rotational matrices
- Duck as digitizing tablet is thrown at you
  - Matrices aren't exactly easily human readable

08-12: Euler Angles

- Describe rotation in terms of *roll*, *pich*, and *yaw*
  - Roll is rotation around the z axis
  - Pitch is rotation around the x axis
  - Yaw is rotation around the y axis

08-13: Euler Angles





08-14: Euler Angles

- We can describe any orientation using Euler Angles
  - Order is important!
  - 30 degree roll followed by 10 degree pitch $\neq$ 10 degree pitch followed by 30 degree roll!

08-15: Euler Angles

- Standard order is
  - Roll, pitch yaw
- Converting from object space to Inertial Space
- To convert from Inertial Space to Object space, go in reverse order

08-16: Euler Angles

- Object Space vs. World Space
  - We can define roll/ptich/yaw in object space
    - Rotate around the object’s z axis
    - Rotate around the object’s x axis
    - Rotate around the object’s y axis
  - Examples, using model

08-17: Euler Angles

- Object Space vs. World Space
  - We can define roll/ptich/yaw in world space
    - Rotate around the world's z axis
    - Rotate around the world's x axis
    - Rotate around the world's y axis
  - Examples, using model

08-18: Euler Angles

- So, what does Ogre use?
  - Both!
    - If we call roll/pitch/yaw functions with a single parameter, we rotate in object space (though we can do world space, too, using a second parameter)
    - If we ask for the euler angles, we get them in world space
      - RPY in word space is the same as YPR in object space

08-19: Euler Angle Problems

- Some issues with Euler angles
  - Any triple of angles describes a unique orientaion (given an order of application)
  - ... But the same orientation can be described with more than one set of Euler Angles!
    - Trivial example: Roll of 20 degrees is same as roll of 380 degrees
    - Can you think of a more complicated example?

08-20: Euler Angle Problems

- Aliasing Issues
  - (Same orientation with different angles, using object space *or* world space)
- Roll 90 degrees, Pitch 90 degrees, Yaw 90 degrees
- Pitch -90 degrees

08-21: Gimbal Lock

- When using Euler angles, we always rotate in a set order
  - Roll, pitch, yaw
- What happens when the 2nd parameter is 90 degrees?
  - Physical system
  - In game engine

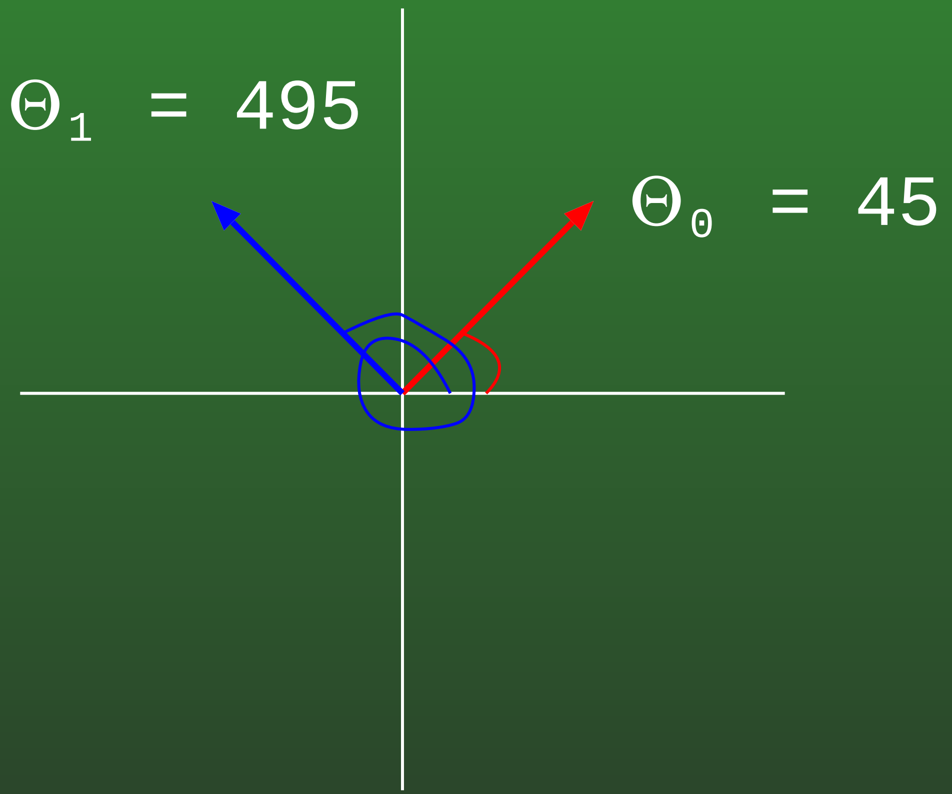
08-22: Angle Interpoloation

- Given two angles, we want to interpolate between them
  - Camera pointing at one object
  - Want to rotate camera to point to another object
  - Want to rotate a *little* bit each frame
- Find the “delta” between the angles, move along it a little bit

08-23: Angle Interpoloation

- Naive approach:
  - Initial Angle: Θ_0, Final angle Θ_1
  - Want to interpolate from Θ_0 to Θ_1: at time $t = 0$ be at angle Θ_0, at time $t = 1$ be at angle Θ_1
    - $\Delta\Theta = \Theta_1 - \Theta_0$
    - $\Theta_t = \Theta_0 + t\Delta\Theta$
  - When does this not “work” (that is, when does it do what we don’t expect?)

08-24: Angle Interpoloation

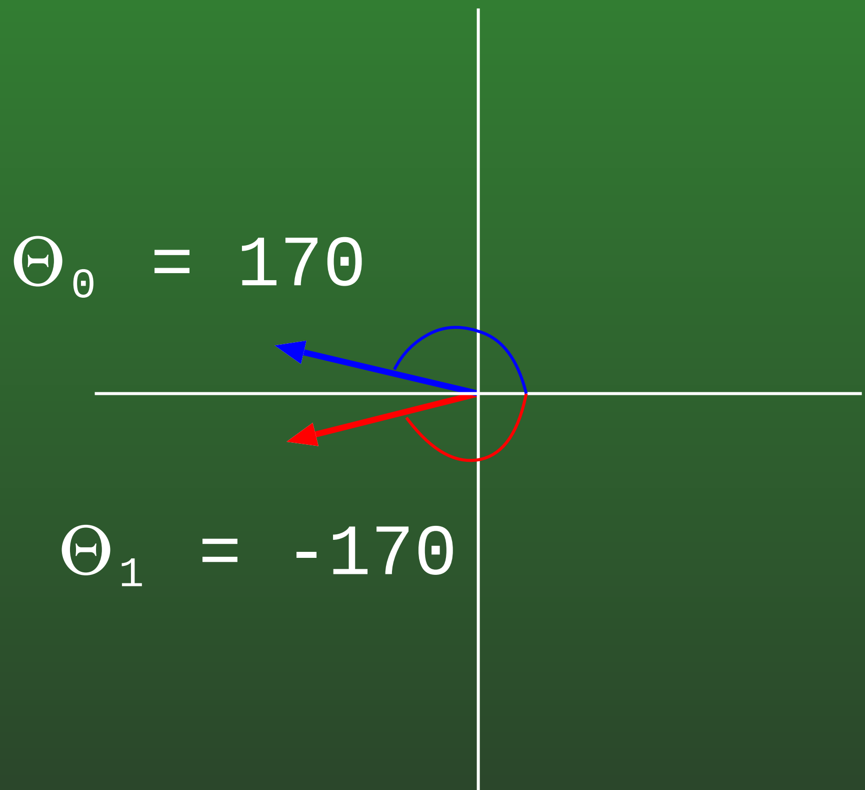


- $\Delta\Theta = 495 - 45$
- $\Theta_t = 45 + 450t$

08-25: Angle Interpoloation

- The naive approach spins all the way around (450 degrees), instead of just moving 45 degrees
- This is an aliasing problem
  - We can fix it by insisting on canonical angles
    - -180 $\leq$ roll $\leq$ 180
    - -90 $\leq$ pich $\leq$ 90
    - -180 $\leq$ yaw $\leq$ 180

08-26: Angle Interpoloation



- $\Delta\Theta = -170 - 170$
- $\Theta_t = 170 - 340t$

08-27: Angle Interpoloation

- We can fix this by forcing Δ, Θ to be in the range $-180 \ldots 180$
  - $wrap(x) = x - 360\lfloor(x + 180)/360\rfloor$
  - $\Delta\Theta = wrap(\Theta_1 - \Theta_0)$
  - $\Theta_t = \Theta_0 + t\Delta\Theta$
- Gimbal lock is still a problem, though
- Gimbal lock (or something analogous) will *always* be a problem if we use 3 numbers to represent angles (exactly why this is so is beyond the scope of this course, however)

08-28: Euler Angle Advantages

- Compact representation (3 numbers – matrices take 9, and Quaternians (up next!) take 4)
- Any set of 3 angles represents a valid orentation (not so with matricies – any 9 numbers are not a valid rotational matrix!)
- Conceptually easy to understand

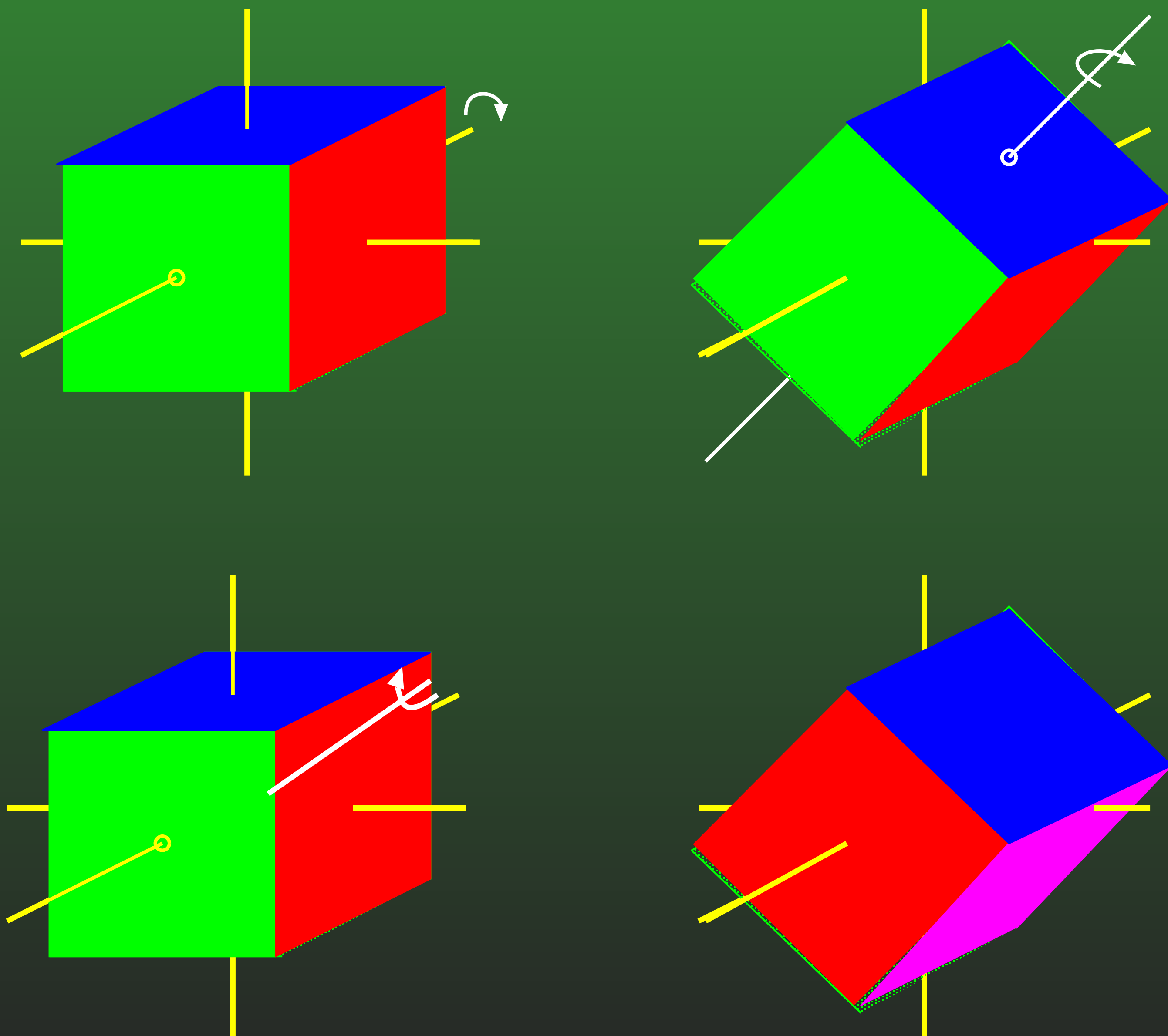
08-29: Euler Angle Disadvantages

- Can't combine rotations as easily as matrices
- Aliasing & Gimal Lock

08-30: Quaternians

- Roating about any axis can be duplicaed by rotations around the 3 cardinal axes
- Goes the other way as well –
  - Any set of roations around x, y, and z can be duplicaed by a single rotation around an arbitary axis

08-31: Rotational Equivalence



08-32: Quatenrions

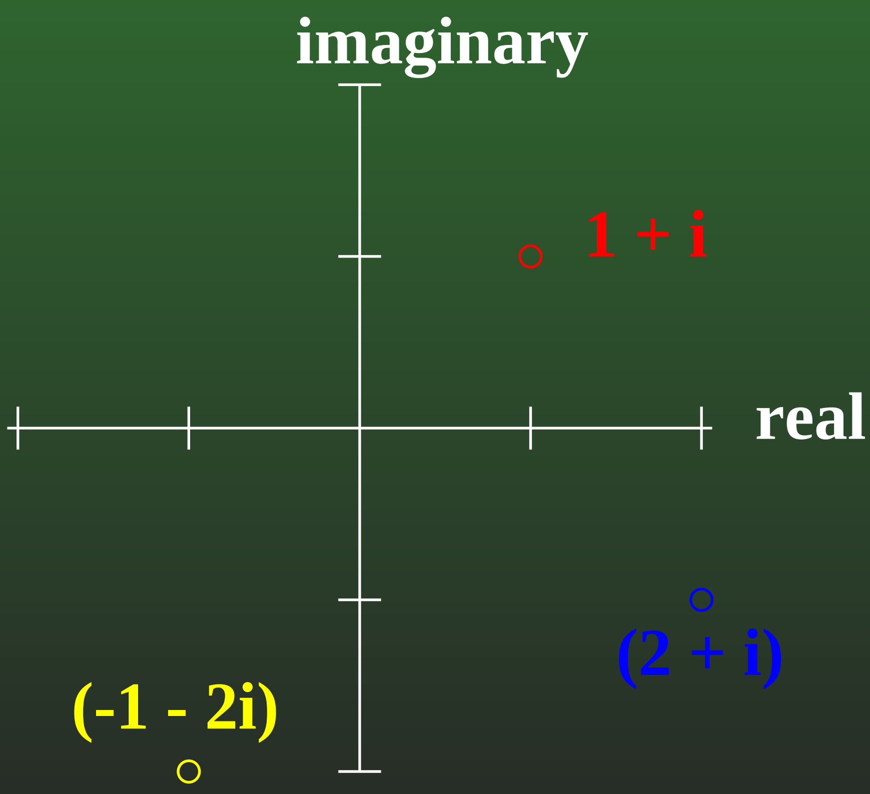
- When using Quaternions for rotation:
  - Quaternion encodes an axis and an angle
  - Represents rotation about that axis by an amount specified by the angle
  - Encoded in a slightly odd way – to understand it, we need to talk about complex numbers

08-33: Imaginary Numbers

- Define $i = \sqrt{-1}$
- Imaginary number is $k * i$ for some real number k
- Complex number has a real and an imaginary component
  - $c = a + bi$

08-34: Complex Plane

- A complex number can be used to represent a point (or a vector) on the complex plane
- “Real” axis and “Imaginary” axis



08-35: Complex Numbers

- Complex numbers can be added, subtracted and multiplied
  - $(a + bi) + (c + di) = (a + c) + (b + d)i$
  - $(a + bi) - (c + di) = (a - c) + (b - d)i$
  - $(a + bi)(c + di) = ac + adi + bci + bdi^2 = ac - bd + (ad + bc)i$
- (Dividing is a wee bit more tricky ...)

08-36: Complex Conjugate

- Complex number $p = a + bi$
- Conjugate of p, $p^* = a - bi$
- What happens when we multiply a number by its conjugate?
  - Think of the geometric interpretation ...

08-37: Complex Conjugate

- Complex number $p = a + bi$
- Conjugate of p, $p^* = a - bi$
- What happens when we multiply a number by its conjugate?

$$
\begin{aligned}
(a+bi)(a-bi) &= a^2 + abi - abi - b^2 i^2 \\
&= a^2 + b^2
\end{aligned}
$$

08-38: Complex Conjugate

- The magnitude of a complex number is the square root of the product of its conjugate
- $||p|| = \sqrt{pp^*}$
- What is the magnitude of a number with no imaginary part?

08-39: Complex Conjugate

- The conjugate of a complex number is also is also handy because the product of a number and it's conjuate has no imaginary part.
  - We can use this fact to do complex division

$$\begin{aligned}\frac{4+3i}{3-2i} &= \frac{(4+3i)(3+2i)}{(3-2i)(3+2i)} \\ &= \frac{12+12i-6}{9+4} \\ &= \frac{6+12i}{13} \\ &= \frac{6}{13}+\frac{12}{13}i\end{aligned}$$

08-40: Complex Conjugate

- The conjugate of a complex number is also is also handy because the product of a number and it's conjuate has no imaginary part.
  - We can use this fact to do complex division

$$
\begin{aligned}
\frac{a+bi}{c+di} &= \frac{(a+bi)(c-di)}{(c+di)(c-di)} \\
&= \frac{ac+bd+(bc-ad)i}{(c^2+d^2)} \\
&= \frac{ac+bd}{c^2+d^2} + \frac{bc-ad}{c^2+d^2}i
\end{aligned}
$$

08-41: Complex Rotations

- We can use complex numbers to represent rotations
  - We can create a “rotational” complex number r_Θ
  - Multiplying a complex number p by r_Θ rotates p Θ degrees counter-clockwise
  - Similar to a rotational matrix in “standard” 2D space

08-42: Complex Rotations

- We can use complex numbers to represent rotations
  - $r_\Theta = \cos\Theta + (\sin\Theta)i$
  - $p = (a + bi)$

$$\begin{aligned} pr_\Theta &= a\cos\Theta + (a\sin\Theta) + (b\cos\Theta)i - b\sin\Theta \\ &= (a\cos\Theta - b\sin\Theta) + (a\sin\Theta + b\cos\Theta)i \end{aligned}$$

- Does this look at all familiar?

08-43: Quaternions

- So, we can use complex numbers to represent points in 2D space, and rotations in 2D space
  - How can we extend this to 3D space?
  - Add an extra imaginary component for the 3rd dimension?

08-44: Quaternions

- So, we can use complex numbers to represent points in 2D space, and rotations in 2D space
  - How can we extend this to 3D space?
  - Add an extra imaginary component for the 3rd dimension?
    - Actually, we'll add *two* additional imaginary components

08-45: Quaternions

- A quaternion is a number with a real part and 4 imaginary parts:
  - $p = a + bi + cj + dk$
- Where i, j and k are all different imaginary numbers, with:
  - $i^2 = j^2 = k^2 = -1$
  - $i * j = k, j * i = -k$
  - $jk = i, kj = -i$
  - $ki = j, ik = -j$

08-46: Quaternions

- Quaternions are often divided into a scalar part (real part of the number) and a vector (complex part of the number)
  - $p = w + xi + yj + xk$
  - $p = [w, (x, y, z)]$
  - $p = [w, \mathbf{v}]$

08-47: Geometric Quaternions

- Complex numbers represent points/vectors in 2D space, and rotations in 2D space
- Quaternions only represent rotations in 3D space (Technically, you can use quaternions to represent scale as well, but we'll only do rotations in this class)
  - Can condier a quaternion to represent an orientation as an offset from some given orientation
  - Just like a vector can represent a point at an offset from the origin

08-48: Geometric Quaternions

- Quaternions represent rotation about an arbitrary axis
- Let $\mathbf{n}$ represent an arbitary unit vector
- Rotation of Θ degrees around $\mathbf{n}$ (using the appropriate handedness rule) is represented by the quaternion:

$$\begin{aligned} q &= [\cos(\Theta/2), \sin(\Theta/2)\mathbf{n}] \\ &= [\cos(\Theta/2), (\sin(\Theta/2)n_x, \sin(\Theta/2)n_y, \sin(\Theta/2)n_z)] \end{aligned}$$

- So, we can represent the position and orientation of a model as a vector and a quaternion (displacement from the origin, and rotation from initial orientation)

08-49: Quaternion Negation

- Negate quaternions by negating each component
  - $-\mathbf{q} = -[w, (x, y, z)] = [-w, (-x, -y, -z)]$
  - $-\mathbf{q} = -[w, \mathbf{v}] = [-w, -\mathbf{v}]$
- What is the geometric meaning of negating a quaternion?
- What happens to the orientation represented by a quaternion if it is negated?

08-50: Quaternion Negation

- Recall: Rotation of Θ degress around $\mathbf{n}$ is represented by
  - $\mathbf{q} = [\cos(\Theta/2) + \sin(\Theta/2)\mathbf{n}]$
- What happens if we add 360 degrees to Θ
  - How does it change the rotation represented by $\mathbf{q}$?
  - How does it change $\mathbf{q}$?

08-51: Quaternion Negation

- Each anglular displacement has *two* different quaternion representations q, q'
- $q = -q'$

08-52: Identity Quaternion

- Identity Quaternion represents no anglular displacement
  - $[1, \mathbf{0}] = [1, (0, 0, 0)]$
- Rotation of 0 degrees around a vector $\mathbf{n}$
  - $q = [\cos 0, \sin 0 * \mathbf{v}] = [1, \mathbf{0}]$
- What about $[-1, \mathbf{0}]$?

08-53: Identity Quaternion

- What about $[-1, \mathbf{0}]$?
  - Also represents no angular displacement (think rotation of 360 degrees)
  - Geometrically equivalent to identity quaternion
  - Not a *true* identity
    - q and $-q$ represent the same orientation, but are different quaternions.

08-54: Quaternion Magnitude

- Magnitude of a quaternion is defined as:
  - $||\mathbf{q}|| = ||[w, (x, y, z)]|| = \sqrt{w^2 + x^2 + y^2 + z^2}$
  - $||\mathbf{q}|| = ||[w, \mathbf{v}]|| = \sqrt{w^2 + ||\mathbf{v}||^2}$
- Let's take a look a geometric interpretation:

$$\begin{aligned} ||[w, \mathbf{v}|| &= \sqrt{w^2 + ||\mathbf{v}||^2} \\ &= \sqrt{\cos^2(\Theta/2) + (\sin(\Theta/2)||\mathbf{n}||)^2} \\ &= \sqrt{\cos^2(\Theta/2) + \sin^2(\Theta/2)||\mathbf{n}||^2} \end{aligned}$$

- If we restrict $\mathbf{n}$ to be a unit vector ...

08-55: Quaternion Magnitude

$$
\begin{aligned}
||[w, \mathbf{v}|| &= \sqrt{w^2 + ||\mathbf{v}||^2} \\
&= \sqrt{\cos^2(\Theta/2) + (\sin^2(\Theta/2)||\mathbf{n}||)^2} \\
&= \sqrt{\cos^2(\Theta/2) + \sin^2(\Theta/2)||\mathbf{n}||^2} \\
&= \sqrt{\cos^2(\Theta/2) + \sin^2(\Theta/2)} \\
&= \sqrt{1} \\
&= 1
\end{aligned}
$$

- All quaternions that represent orienation (using normalized $\mathbf{n}$) are *unit quaternions*

08-56: Conjugate & Inverse

- The conjugate of a quaternion is very similar to the complex conjugate
  - $\mathbf{q} = [w, \mathbf{v}] = [w, (x, y, z)]$
  - $\mathbf{q}^* = [w, -\mathbf{v}] = [w, (-x, -y, -z)]$
- The *inverse* of a quaternion is defined in terms of the conjugate
  - $\mathbf{q}^{-1} = \frac{\mathbf{q}^*}{||\mathbf{q}||} = q^*$ (for unit quaternions)

08-57: Quaternion Multiplication

- Quaternion Multiplication is just like complex multiplication:

$$
\begin{aligned}
\mathbf{q_1 q_2} &= (w_1 + x_1 i + y_1 j + z_1 k)(w_2 + x_2 i + y_2 j + z_2 k) \\
&= w_1 w_2 + w_1 x_2 i + w_1 y_2 j + w_1 z_2 k + \\
&\quad x_1 w_2 i + x_1 x_2 i^2 + x_1 y_2 ij + x_1 z_2 ik + \\
&\quad y_1 w_2 j + y_1 x_2 ji + y_1 y_2 j^2 + y_1 z_2 jk + \\
&\quad z_1 w_2 k + z_1 x_2 ki + z_1 y_2 kj + z_1 z_2 k^2 + \\
\\
&= w_1 w_2 + w_1 x_2 i + w_1 y_2 j + w_1 z_2 k + \\
&\quad x_1 w_2 i + x_1 x_2 (-1) + x_1 y_2 (k) + x_1 z_2 (-j) + \\
&\quad y_1 w_2 j + y_1 x_2 (-k) + y_1 y_2 (-1) + y_1 z_2 i + \\
&\quad z_1 w_2 k + z_1 x_2 j + z_1 y_2 (-i) + z_1 z_2 (-1) +
\end{aligned}
$$

08-58: Quaternion Multiplication

- Quaternion Multiplication is just like complex multiplication:

$$
\begin{aligned}
\mathbf{q_1 q_2} &= (w_1 + x_1 i + y_1 j + z_1 k)(w_2 + x_2 i + y_2 j + z_2 k) \\
&= \ldots \\
&= w_1 w_2 + w_1 x_2 i + w_1 y_2 j + w_1 z_2 k + \\
& \quad x_1 w_2 i + x_1 x_2 (-1) + x_1 y_2 (k) + x_1 z_2 (-j) + \\
& \quad y_1 w_2 j + y_1 x_2 (-k) + y_1 y_2 (-1) + y_1 z_2 i + \\
& \quad z_1 w_2 k + z_1 x_2 j + z_1 y_2 (-i) + z_1 z_2 (-1) + \\
\\
&= w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2 + \\
& \quad (w_1 x_2 + x_1 w_2 + y_1 z_2 - z_1 y_2) i + \\
& \quad (w_1 y_2 + y_1 w_2 + z_1 x_2 + x_1 z_2) j + \\
& \quad (w_1 z_2 + z_1 w_2 + x_1 y_2 + y_1 x_2) k
\end{aligned}
$$

08-59: Quaternion Multiplication

- Quaternion Multiplication is associative, but not commutative
  - $(\mathbf{q}_1\mathbf{q}_2)\mathbf{q}_3 = \mathbf{q}_1(\mathbf{q}_2\mathbf{q}_3)$
  - $\mathbf{q}_1\mathbf{q}_2 \neq \mathbf{q}_2\mathbf{q}_1$
- Mangnitude of product = product of magnitude
  - $||q_1 q_2|| = ||q_1|| \, ||q_2||$
    - Result of multiplying two unit quaternions is a unit quaternion

08-60: Quaternion Multiplication

- Given any two quaternions q_1 and q_2:
  - $(\mathbf{q}_1\mathbf{q}_2)^{-1} = \mathbf{q}_2^{-1}\mathbf{q}_1^{-1}$

08-61: Quaternion Rotation

- We can use quaternions to rotate a vector around an axis $\mathbf{n}$ by angle Θ
  - Let $\mathbf{q}$ be a quaternion $[w, (x, y, z)]$ that represents rotation about $\mathbf{n}$ by Θ
  - Let $\mathbf{v}$ be a “quaternion” version of the vector (same vector part, real part zero)
  - Rotated vector is: $\mathbf{q}\mathbf{v}\mathbf{q}^{-1}$

Quaternion Rotation

- How can we prove that the rotated version of $\mathbf{v}$ is $\mathbf{q}\mathbf{v}\mathbf{q}^{-1}$? Do the multiplication!
- Given $\mathbf{n}$, Θ, and $\mathbf{v} = [v_x, v_y, v_z]$:
- Create:
  - $\mathbf{q} = [\cos(\Theta/2), \sin(\Theta/2)(n_x, n_y, n_z)]$
  - $\mathbf{q}^{-1} = [\cos(\Theta/2), -\sin(\Theta/2)(n_x, n_y, n_z)]$
  - $v = [0, (v_x, v_y, v_z)]$
- Calculate $\mathbf{q}\mathbf{v}\mathbf{q}^{-1}$

08-63: Quaternion Rotation

- Calculate $v' = \mathbf{q}\mathbf{v}\mathbf{q}^{-1}$
  - ... Much ugly algebra later ...
  - Vector portion of v' is:

    $\mathbf{v}' = \cos\Theta(\mathbf{v} - (\mathbf{v}\cdot\mathbf{n})\mathbf{n}) + \sin\Theta(\mathbf{n}\times\mathbf{v}) + (\mathbf{v}\cdot\mathbf{n})\mathbf{n}$

    Which is what we calculated earlier for rotation of Θ degrees around an aritrary axis $\mathbf{n}$

08-64: Quaternion Rotation

- What it we wanted to do more than one rotation?
  - First rotate by q_1, and then rotate by q_2
- First, rotate by q_1: $q_1 v q_1^{-1}$
- Next, rotate that quantity by q_2: $q_2(q_1 v q_1^{-1})q_2^{-1}$
- $\mathbf{q}_2\mathbf{q}_1\mathbf{v}\mathbf{q}_1^{-1}\mathbf{q}_2^{-1} = (\mathbf{q}_2\mathbf{q}_1)\mathbf{v}(\mathbf{q}_2\mathbf{q}_1)^{-1}$

08-65: Quaternion "Difference"

- Given two quaternions $\mathbf{p}$ and $\mathbf{q}$, find the rotation required to get from $\mathbf{p}$ to $\mathbf{q}$
- That is, given p and q, find a d such that
  - $\mathbf{dp} = \mathbf{q}$
  - $\mathbf{d} = \mathbf{qp}^{-1}$
- Given two orientations $\mathbf{p}$ and $\mathbf{q}$, we can generated the angular displacement from one to another

08-66: Quaternion Log and Exp

- We’ll now define a few “helper” functions, that aren’t useful in and of themselves, but they will allow us to do a slerp, which is *very* useful
  - Quaternion Log
  - Quaternion Exp (“Anti-log”)

08-67: Quaternion Log and Exp

- Define $\alpha = \Theta/2$ (as a notational convenience)
  - $\mathbf{q} = [\cos\alpha, (\sin\alpha)\mathbf{n}]$
  - $\mathbf{q} = [\cos\alpha, (\sin\alpha n_x, \sin\alpha n_y, \sin\alpha n_z)]$
- $\log\mathbf{q} = \log([\cos\alpha, (\sin\alpha)\mathbf{n}] \equiv [0, \alpha\mathbf{n}]$

08-68: Quaternion Log and Exp

- Given a quaternion p of the form:
  - $\mathbf{q} = [0, \alpha\mathbf{n}] = [0, (\alpha n_x, \alpha n_y, \alpha n_z)]$
- $\exp(p) = \exp([0, \mathbf{n}]) \equiv [\cos\alpha, \sin\alpha\mathbf{n}]$
- Note that $\exp(\log(\mathbf{q})) = \mathbf{q}$

08-69: Scalar Multipication

- Given any quaternion $\mathbf{q} = [w, (x, y, z)]$ and scalar a
- $a\mathbf{q} = \mathbf{q}a = [aw, (ax, ay, az)]$

08-70: Quaternion Exponentiation

- q is a quaternion that represents a rotation about an axis
- Define q^t such that:
  - q^0 = identity quaternion
  - $q^1 = q$
  - $q^{1/2}$ = half the rotation around the axis defined by q
  - $q^{-1/2}$ = half the rotation around the axis defined by q, in the opposite direction

Quaternion Exponentiation

- q^0 = identity quaternion
- $q^1 = q$
- q^2 = twice half the rotation around the axis defined by q
  - Well, sort of.
  - Displacement using the shortest possible arc
  - Can't use exponentiation to represent multiple spins around the axis
  - Compare $(q^4)^{1/2}$ to q^2, when q represents 90 degrees ...

08-72: Quaternion Exponentiation

- We can define quaternion exponentiation mathematically:
  - $\mathbf{q}^t = \exp(t \log \mathbf{q})$
- Why does this work?
  - Log function extracts $\mathbf{n}$ and Θ from q
  - Multiply Θ by t
  - “Undo” log operation

08-73: Slerp

- Spherical Linear Interpolation
- Input: Two orientations (quaternions) q_1 and q_2, and a value $0 \leq t \leq 1$
- Output: An orientation that is between q_1 and q_2
  - If $t = 0$, result is q_1
  - If $t = 1$, result is q_2
  - if $t = 1/2$, result is 1/2 way between them

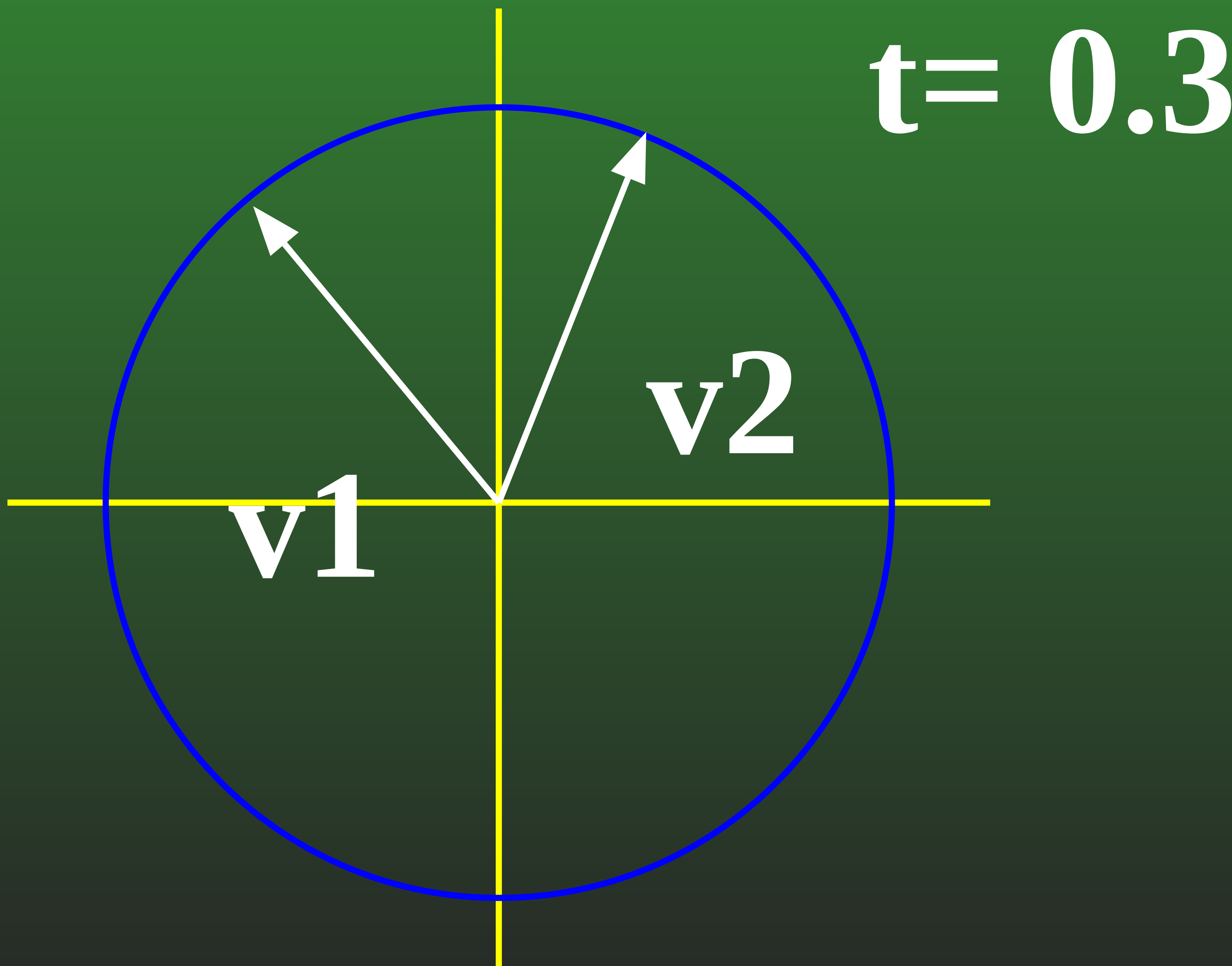
08-74: Slerp

- slerp(q_1,q_2,t):
  - Start with orientation q_1
  - Find the difference between q_1 and q_2
  - Calcualte portion t of the difference
- slerp(q_1,q_2,t) = $q_1(q_1^{-1}q_2)^t$

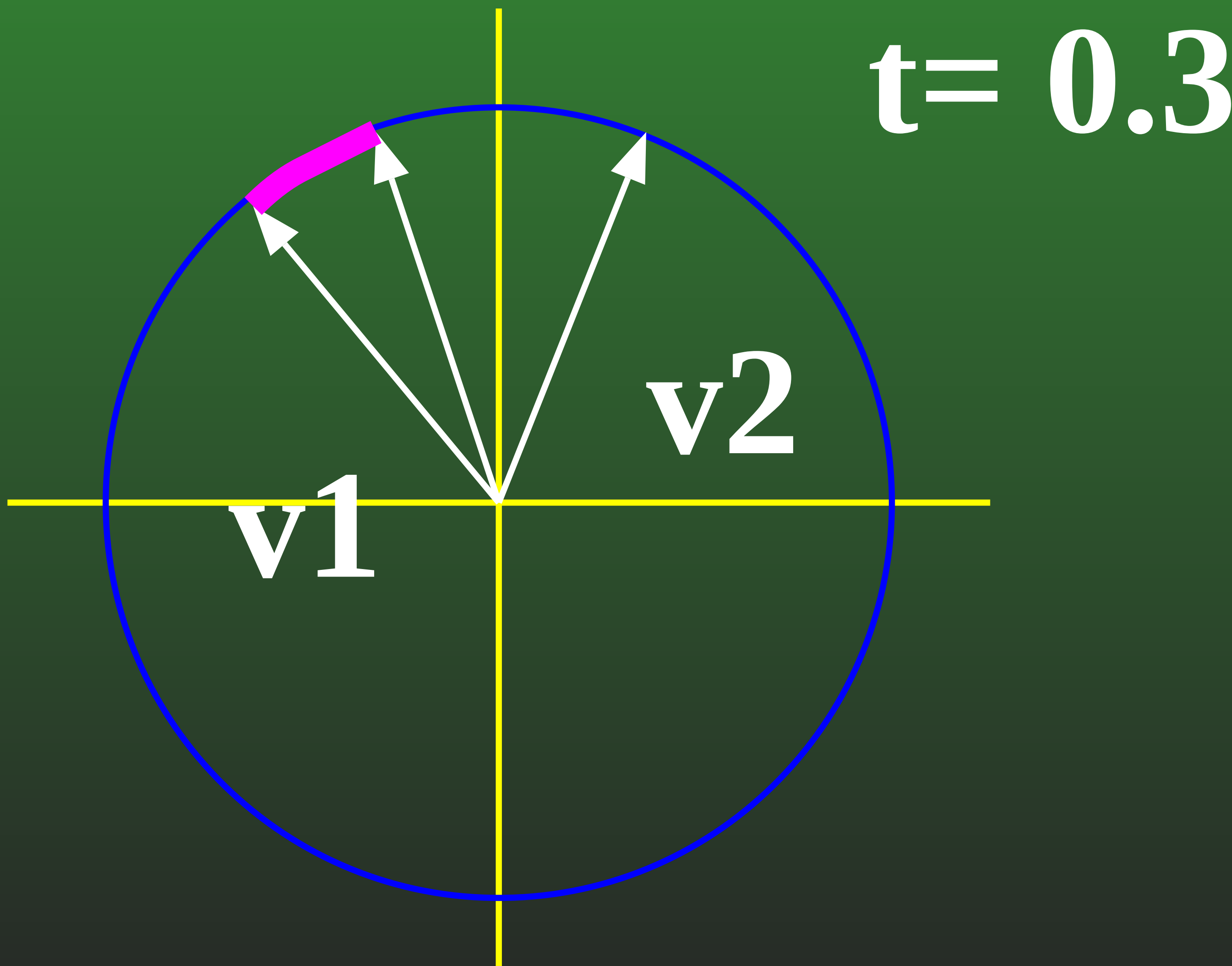
08-75: Slerp

- Finding Slerp, version II
  - Let's say we had two 2-dimensional unit vectors, and we wanted to interpolate between them.
  - All 2-dimensional unit vectors live on a circle
  - To interpolate 30% between $\mathbf{v}_1$ and $\mathbf{v}_2$, go 30% of the way along the arc between them

Slerp



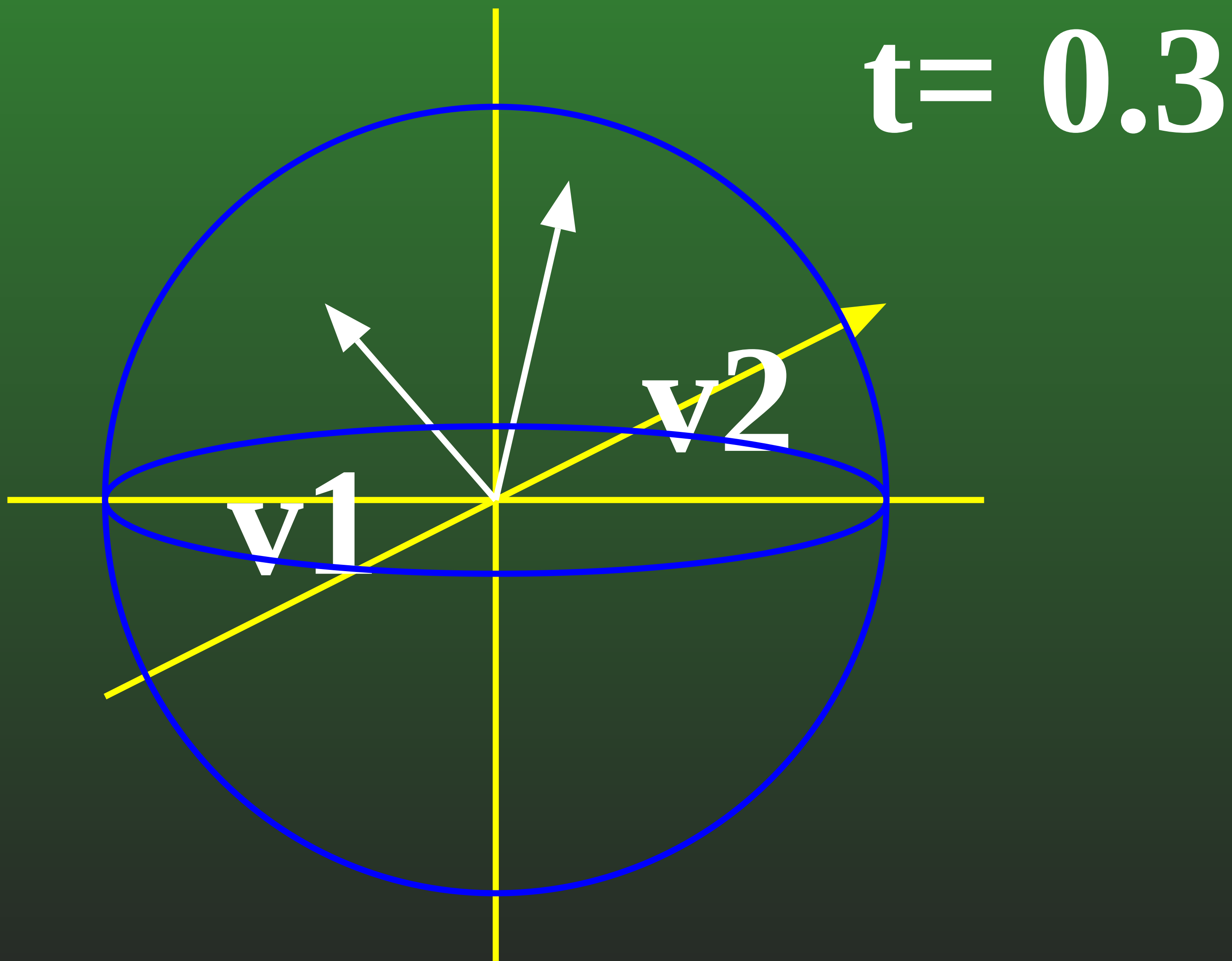
Slerp



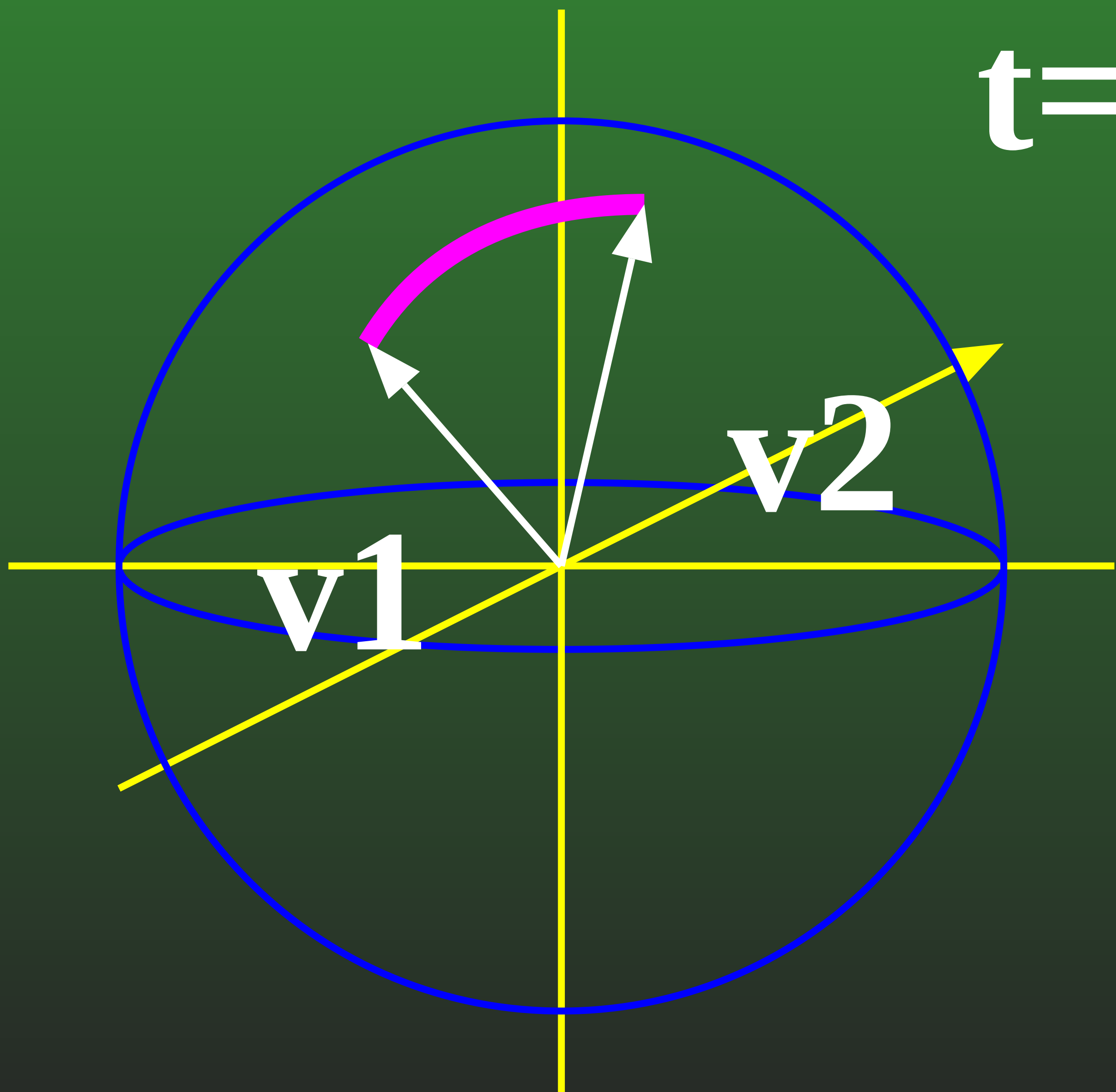
08-78: Slerp

- Finding Slerp, version II
  - Let’s say we had two 3-dimensional unit vectors, and we wanted to interpolate between them.
  - All 3-dimensional unit vectors live on a sphere
  - To interpolate 30% between $\mathbf{v}_1$ and $\mathbf{v}_2$, go 30% of the way along the arc between them

08-79: Slerp

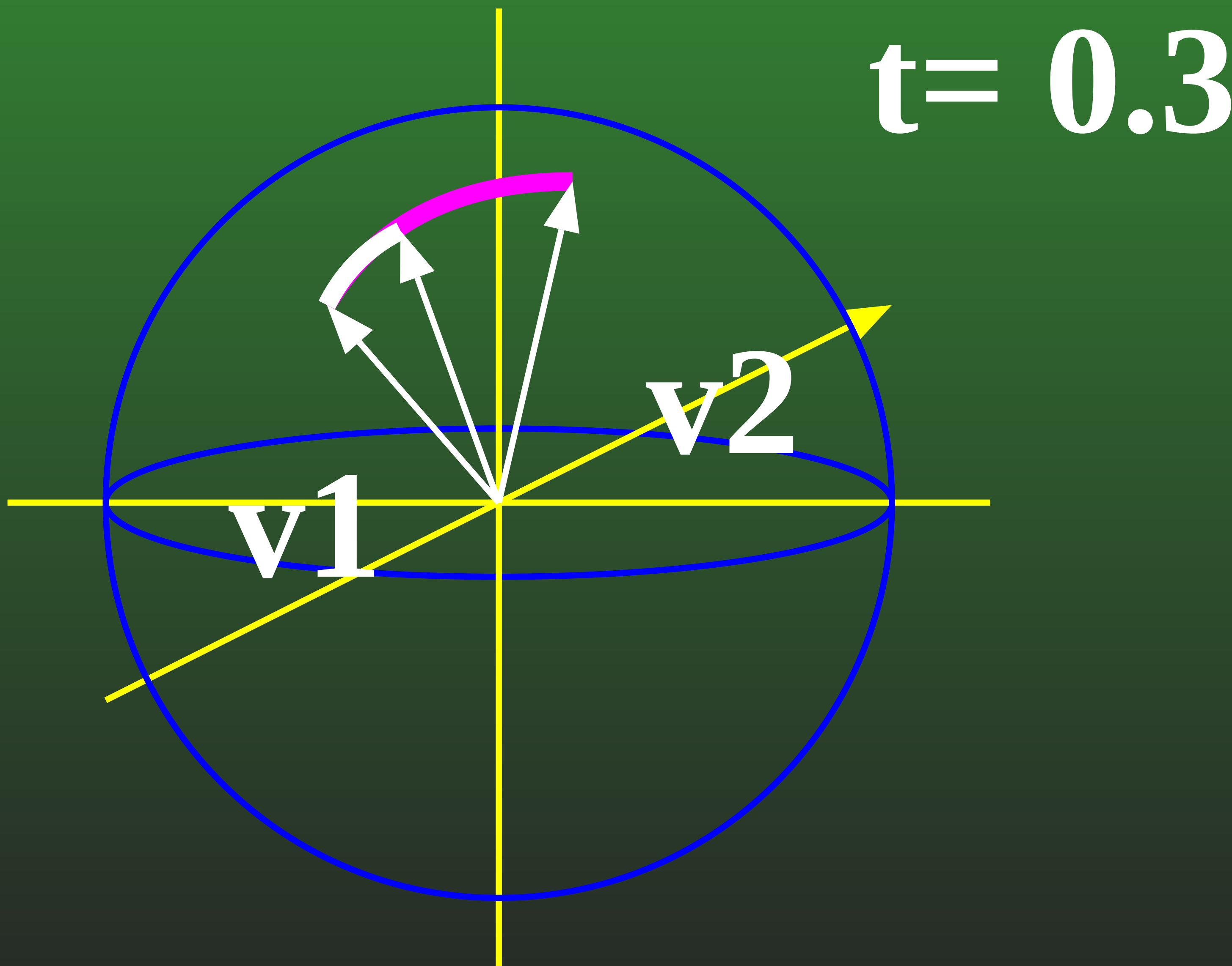


Slerp



t= 0.3

Slerp



08-82: Slerp

- Finding Slerp, version II
  - Let's say we had two 4-dimensional unit vectors, and we wanted to interpolate between them.
  - All 4-dimensional unit vectors live on a hypersphere
  - To interpolate 30% between $\mathbf{v}_1$ and $\mathbf{v}_2$, go 30% of the way along the arc between them

08-83: Slerp

(Sorry, no 4D diagram)

- slerp($\mathbf{q_1}, \mathbf{q_2}, t$) = $\frac{\sin(1-t)\omega}{\sin\omega} q_1 + \frac{\sin t\omega}{\sin\omega} q_1$
- ω is the angle between q_1 and q_2, can get it using a dot product
  - We can get $\cos\omega$ easily using the dot product, and can then get $\sin\omega$ from that

08-84: Using Quaternions

- Orientations in Ogre use quaternions
- Multiplication operator for multiplying a quaternion and a vector is overloaded to do the “right thing”
  - `Ogre::Quaternion q`
  - `Ogre::Vector v;`
  - `q*v` returns `v` rotated by `q`

08-85: Using Quaternions

- Tank example:
  - Quaternion & Position vector for tank
  - Quaternion & Position vector for barrel
  - End of barrel is 3 units down barrel's z axis
- Where is the end of the barrel in world space

08-86: Using Quaternions

- Tank: Orientation $\mathbf{q}_t$, Position $\mathbf{p}_t$
- Barrel: Orientation $\mathbf{q}_b$, Position $\mathbf{p}_b$
- End of barrel in world space:

$$\mathbf{q}_t(\mathbf{q}_b[0,0,3] + \mathbf{p}_b) + \mathbf{p}_t$$

08-87: Change Representations

- We are not restricted to using just matrices, or just euler angles, or just quaternions to represent orientation
  - We can go back and forth between representations
  - Given a set of Euler Angles, create a Rotational Marix
  - Given a Rotational Matrix, create a quaternion
  - ... etc

08-88: Euler Angles -> Matrix

- Given Euler angles in world space (as opposed to object space), it is easy to create an equivalent rotational matrix
- How?

08-89: Euler Angles -> Matrix

- Euler angles in world space represent a rotation around each axis
- We can create a matrix for each rotation, and combine them
  - Creating a rotational matrix for the cardinal axes is easy

08-90: Euler Angles -> Matrix

- For the euler angles r,p, y, the matrix would be:

$$\begin{bmatrix} \cos(r) & \sin(r) & 0 \\ -\sin(r) & \cos(r) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(p) & \sin(p) \\ 0 & -sin(p) & \cos(p) \end{bmatrix} \begin{bmatrix} \cos(y) & 0 & -\sin(y) \\ 0 & 1 & 0 \\ \sin(y) & 0 & \cos(y) \end{bmatrix} =$$

$$\begin{bmatrix} \cos r \cos y + \sin r \sin p \sin y & \sin r \cos p & \sin r \sin p \cos y - \cos r \sin y \\ \cos r \sin p \sin y - \sin r \cos y & \cos r \cos p & \cos r \sin p \cos y + \sin b \sin y \\ \cos p \sin y & -\sin p & \cos p \cos y \end{bmatrix}$$

08-91: Euler Angles -> Matrix

- What if your euler angles are in object space, and not world space?
- Then how do you create the appropriate matrix?

08-92: Euler Angles -> Matrix

- What if your euler angles are in object space, and not world space?
- Then how do you create the appropriate matrix?
  - Create the RPY matrices as before
  - Multiply them in the reverse order

Matrix -> Euler Angle

- What if we have a matrix, and we want to create a world-relative euler angle triple?
- Little more complicated than the other direction – recall the definition of a martrix from euler angles (we'll work backwards, kind of like a sudoku puzzle)

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} =$$

$$\begin{bmatrix} \cos r \cos y + \sin r \sin p \sin y & \sin r \cos p & \sin r \sin p \cos y - \cos r \sin y \\ \cos r \sin p \sin y - \sin r \cos y & \cos r \cos p & \cos r \sin p \cos y + \sin b \sin y \\ \cos p \sin y & -\sin p & \cos p \cos y \end{bmatrix}$$

08-94: Matrix -> Euler Angle

- From the previous equation:
  - $m_{32} = -\sin p$
  - $p = \arcsin(-m_{32})$
- So we have p – next up is y – once we have p, how can we get y?

$$\begin{bmatrix} \cos r \cos y + \sin r \sin p \sin y & \sin r \cos p & \sin r \sin p \cos y - \cos r \sin y \\ \cos r \sin p \sin y - \sin r \cos y & \cos r \cos p & \cos r \sin p \cos y + \sin b \sin y \\ \cos p \sin y & -\sin p & \cos p \cos y \end{bmatrix}$$

08-95: Matrix -> Euler Angle

- Assume that $\cos p \neq 0$ for the moment:
  - $m_{31} = \cos p \sin y$
  - $\sin y = m_{31} / \cos p$
  - $y = \arcsin(m_{31} / \cos p)$
  - (can do this a litle more efficiently with atan2)

08-96: Matrix -> Euler Angle

- Once we have p and y (again assuming $\cos p! = 0$) it is relatively easy to get r:
  - $m_{12} = \sin r \cos p$
  - $r = \arcsin(m_{12}/\cos p)$

08-97: Matrix -> Euler Angle

- What if $\cos p = 0$?
  - That means that p = 90 degrees
  - Gimbal lock case!
  - Yaw, roll do the same operation!
  - We need to make some assumptions about how much to roll and yaw

08-98: Matrix -> Euler Angle

- What if $\cos p = 0$?
  - p = 90 degrees
  - Assume no yaw (since roll does the same thing)
  - $\cos p = 0$, $\sin p = 1$, $y = 0$ $\sin y = 0$, $\cos y = 1$

$$\begin{bmatrix} \cos r \cos y + \sin r \sin p \sin y & \sin r \cos p & sin r \sin p \cos y - \cos r \sin y \\ \cos r \sin p \sin y - \sin r \sin y & \cos r \cos p & \cos r \sin p \cos y + \sin p \sin y \\ \cos p \sin y & -\sin p & \cos p \cos y \end{bmatrix}$$

$$= \begin{bmatrix} \cos r & 0 & \sin r \\ -1 \sin r & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

08-99: Matrix -> Euler Angle

- $m_{11} = \cos r$, and we're set!
  - (We can use $m_{12} = \sin r$ and atan2 for some more efficiency)

08-100: Quaternion -> Matrix

- Since we can use quaternions to rotate vectors, going from a quaternion to a matrix is easy.
- How?

08-101: Quaternion -> Matrix

- Rotational matrix == position of x, y, and z axes after rotation
- So, all we need to do is rotation basis vectors $[1, 0, 0]$, $[0, 1, 0]$ and $[0, 0, 1]$ by the quaternion!
  - $\mathbf{x}_{new} = q[0, (1, 0, 0)]q^{-1}$ (just $q[1, 0, 0]$ in ogre)
  - $\mathbf{y}_{new} = q[0, (0, 1, 0)]q^{-1}$ (just $q[0, 1, 0]$ in ogre)
  - $\mathbf{z}_{new} = q[0, (0, 0, 1)]q^{-1}$ (just $q[0, 0, 1]$ in ogre)
- Combine these 3 vectors into a matrix

08-102: Other conversions

- We can do other conversions as well
  - Matrix->Quaternion
  - Euler->Quaternion
  - Quaternion->Matrix
  - ... etc
- Basic approach is the same, some of the math is a little uglier