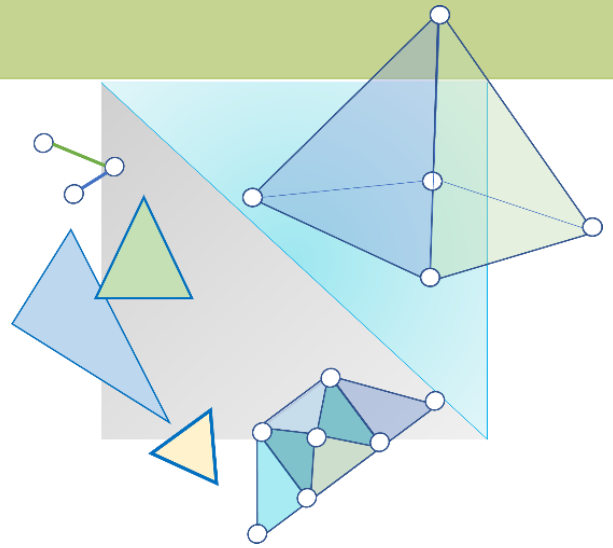
CITS3003 Graphics & Animation



Lecture 22:

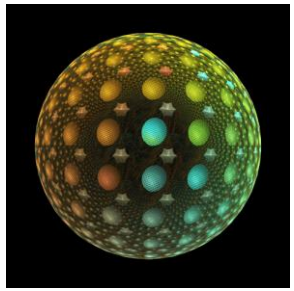
Animation Fundamentals & Unit Quaternions

Objectives

- To learn the concept of keyframes in animation
- The gimbal lock problem
- Quaternions (in place of rotation matrices)
- Rotations using quaternions
- Interpolating quaternions for smooth animation

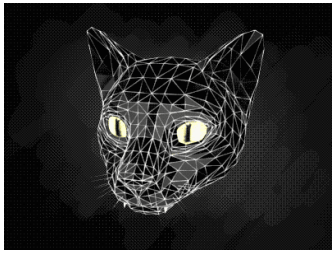
Static Objects are Boring

They just sit there. We can move them to other places and alter the rotation and colour, but they're still pretty boring



Link

- What about more complicated & interesting shapes?
  - living things
  - vehicles
  - etc.



Link

- These things are all made up of separate moving parts that are still part of the whole object
  - where I go, my arms go with me

Link

Tin Toy

The First Oscar Winning Computer Animated Film

Tin Toy (1988) - Pixar Short Film

https://www.youtube.com/watch?v=DWi2WTqD59A

Typical Role of Artist

- Create a mesh for a game shape/character/vehicle
  - geometry
  - texturing
  - materials
- Create animation sequences for different states for that entity

Keyframe (Cell) Animation

- In traditional hand-drawn animation:
  - the senior key artist would draw the **keyframes**, and
  - the junior artist would fill the **in-between frames** (“tweening”).

keyframes

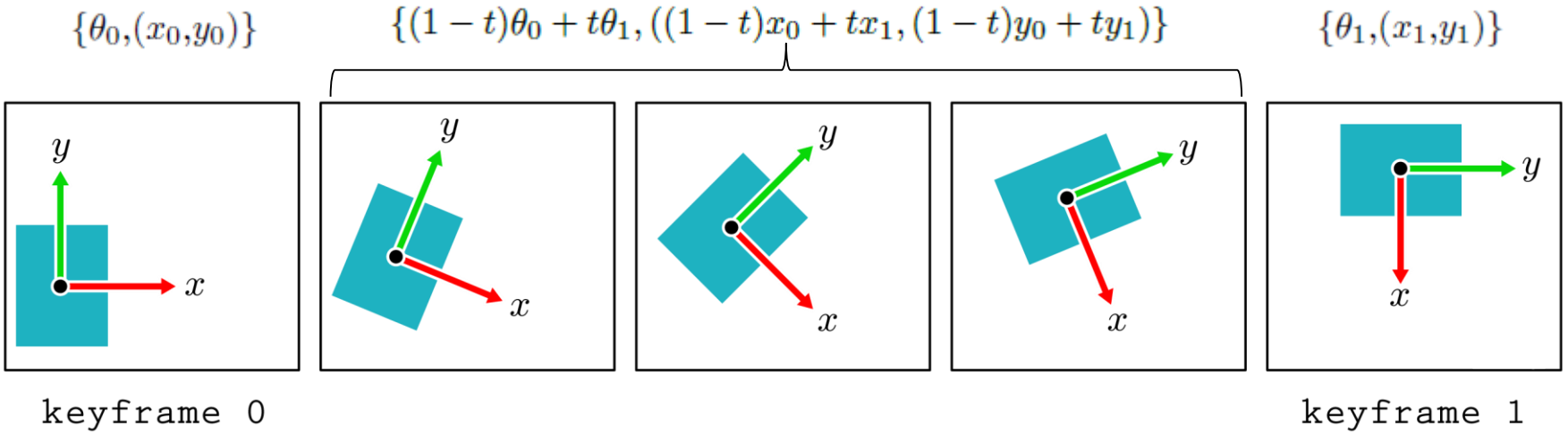
tweens

(c) Walt Disney Company, from "The Illusion of Life"

link

Keyframe (Cell) Animation

- In traditional hand-drawn animation:
  - the senior key artist would draw the **keyframes**, and
  - the junior artist would fill the **in-between frames**.
- We can use a similar approach for computer animation
- For a 30-fps animation, less than 30 keyframes need to be defined per second:
  1. The data (e.g., joint angles of an articulated figure) are assigned to the keyframes
  2. In-between frames are interpolated



Again, the Positions between Frames?

- The positions between frames must be interpolated. Linear interpolation is commonly used.

- Assume a time from 0.0 to 1.0
  - from one keyframe to the next

$$p_i = p_1 + t_i * (p_2 - p_1)$$

where p_1 and p_2 are positions in keyframes 1 and 2; i is the index of the frame (between the two keyframes) to be interpolated;

$i = 1 \cdots n$.

A Simple Animation Example: A Tank

- Gun Turret should be able to rotate
- Gun Barrel should be able to move up and down
- The wheels should be able to spin

Note: keyframes do not need to be equidistant.

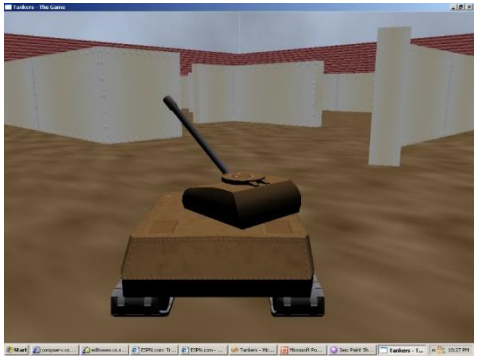
Keyframe 1: set the initial position of the gun turret

Keyframe 2: set the new desired position of the gun turret

Keyframe 3: set the new desired position of the gun barrel

What would be the Tank Hierarchy?

- Tank hull – this is the central parent section
  - Wheels – these are children of the hull and are siblings to each other
  - Gun Turret – this is another child of the hull
    - Gun Barrel – this is the only child of the Gun Turret



Thus,

- where the hull goes, the wheels go
- when the hull rotates, the wheels rotate

Note that the hull, wheels, etc., are represented as 3D meshes (i.e., triangles), so when the hull moves, it is actually the vertices of the triangles being transformed.

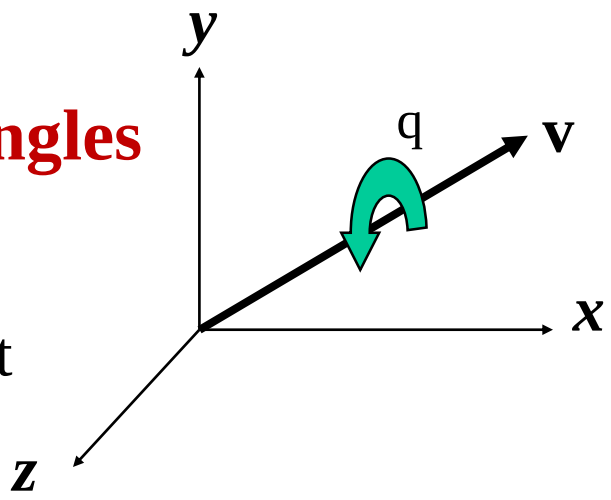
General Rotation About the Origin

A rotation angle of q about an arbitrary axis can be decomposed into the concatenation of rotations about the x, y, and z axes

$$\boldsymbol{R}(\theta) = \boldsymbol{R}_z(\theta_z)\,\boldsymbol{R}_y(\theta_y)\,\boldsymbol{R}_x(\theta_x)$$

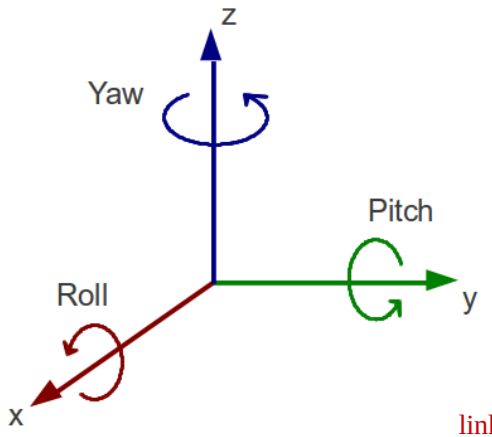
θ_x, θ_y, and θ_z are called the **Euler angles**

Note that **rotations do not commute**
We can use rotations in another order but with different angles



Problems with Interpolating Euler Angles

- The Euler angles (pitch, yaw, and roll) are often used (e.g., in blender) to represent the rotation angles of a body limb about a body joint, e.g.,
  - In keyframe 1 (frame number 11): we have α_1, β_1, and γ_1 representing the pitch, yaw, and roll angles of a rotation.
  - In keyframe 2 (frame number 21): we have α_2, β_2, and γ_2 representing the pitch, yaw, and roll angles of another rotation.
  - We need to compute these angles for frames 12 to 20.

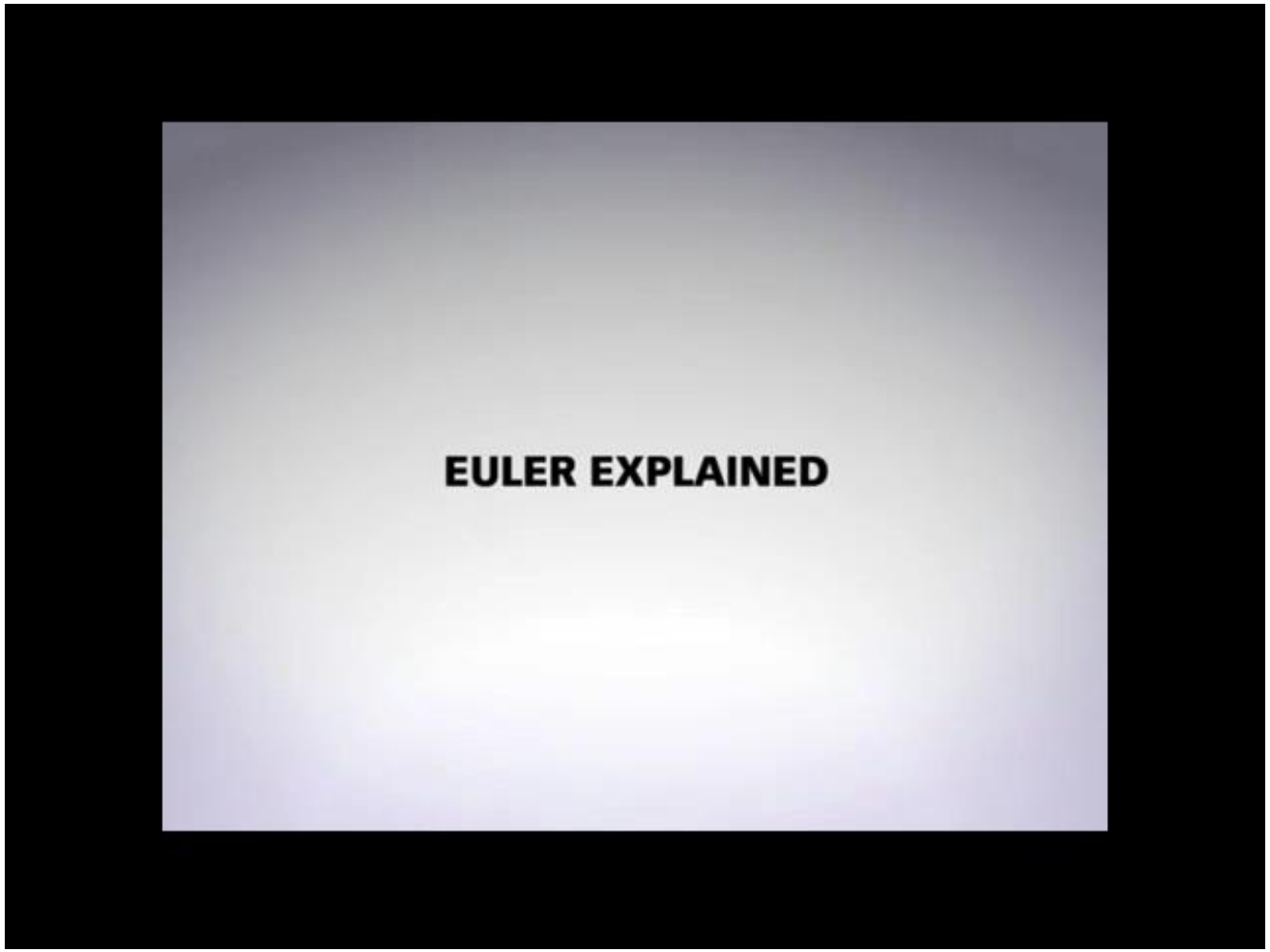


link

Problems with Interpolating Euler Angles

- The Euler angles (pitch, yaw, and roll) are often used (e.g., in blender) to represent the rotation angles of a body limb about a body joint, e.g.,
  - In keyframe 1 (frame number 11): we have α_1, β_1, and γ_1 representing the pitch, yaw, and roll angles of a rotation.
  - In keyframe 2 (frame number 21): we have α_2, β_2, and γ_2 representing the pitch, yaw, and roll angles of another rotation.
  - We need to compute these angles for frames 12 to 20.
- Unfortunately, the Euler angles are problematic when interpolating:
  - Unexpected rotations would result when we combine the three interpolated angles.
  - **Gimbal lock** – the second transform may align the first and third, leading to 2 degrees of freedom rather than 3

Euler Angles and Gimbal lock

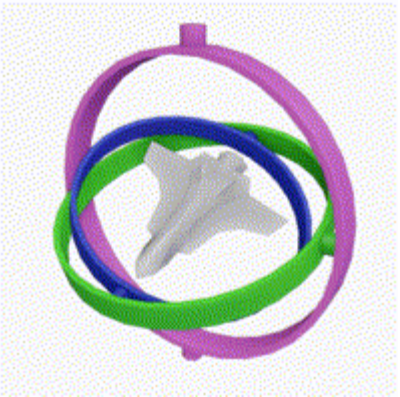


https://www.youtube.com/watch?v=zc8b2Jo7mno

Gimbal Lock

- Gimbal lock refers to the situation where a rotation accidentally brings two local coordinate axes aligned and thus losing 1 degree of freedom
  (i.e., we are left with two affective rotation axes rather than three).

- This is a serious problem as using the Euler angles to represent rotations we are not able to rotate about one of the 3 principal axes (x, y, and z) when gimbal lock occurs.



link

Quaternions

- An alternative solution is to use **unit quaternions**.
- Quaternions are an extension of complex numbers. They are often written as $a + bi + cj + dk$, where $a, b, c, d \in \mathbb{R}$, and i, j, k satisfy $i^2 = j^2 = k^2 = ijk = -1$.
- An alternative representation (and more suitable for representing rotations) is consider quaternions as 4D-vectors $\mathbf{q} = (a, b, c, d)$.

  For compactness, we often write this as $\mathbf{q} = (a, \mathbf{v})$, where $a \in \mathbb{R}$ and $\mathbf{v} \in \mathbb{R}^3$.
- When $a = 0$, we refer to the quaternion as a **pure quaternion**; when $\|\mathbf{q}\| = 1$, we refer to it as a **unit quaternion**.
- Rotations can be represented using unit quaternions.

Quaternion Advantages

- Interpolation is easy between 2 quaternions.
- Smoother animation can be achieved.
- Compared to matrices, quaternions take up less storage space.
- Quaternions can be easily converted to matrices for rendering.
- Quaternions do not suffer from gimbal lock.

Unit Quaternions are related to Angle-Axis Rotations

- We use **unit quaternions** to represent 3D rotations, i.e., if we use $\mathbf{q} = (a, \mathbf{v})$ to represent a rotation then we normalize $\mathbf{q}$ so that $\|\mathbf{q}\| = 1$.

- Any 3D rotation can be described by defining
  - A rotation axis $\mathbf{w} = (w_1, w_2, w_3)$ in the 3D space
  - A rotation angle θ about that axis

  The corresponding unit quaternion is $\left(\cos\left(\frac{\theta}{2}\right), \mathbf{w}\sin\left(\frac{\theta}{2}\right)\right)$

Unit Quaternions – A Few Examples

- Rotation about the x-axis an angle α: the unit quaternion is $\left(\cos\left(\frac{\alpha}{2}\right), \sin\left(\frac{\alpha}{2}\right), 0, 0\right)$
- Rotation about the y-axis an angle β: the unit quaternion is $\left(\cos\left(\frac{\beta}{2}\right), 0, \sin\left(\frac{\beta}{2}\right), 0\right)$
- Rotation about the z-axis an angle γ: the unit quaternion is $\left(\cos\left(\frac{\gamma}{2}\right), 0, 0, \sin\left(\frac{\gamma}{2}\right)\right)$
- We know that the inverse of a rotation matrix $\mathbf{R}$ is $\mathbf{R}^T$. Let $\mathbf{q} = (a, \mathbf{v})$ be the quaternion representing $\mathbf{R}$, then $\mathbf{q}^{-1} = (a, -\mathbf{v})$ is the inverse of $\mathbf{q}$ representing $\mathbf{R}^T$.
- The quaternion $(1, 0, 0, 0)$ represents the identity matrix (i.e., zero rotation angle).

Quaternion Multiplication

- Multiplying two rotation matrices together is equivalent to multiplying two quaternions.
- The multiplication of two quaternions is defined as follows:

  Let $\mathbf{q}_1 = (s_1, \mathbf{v}_1)$ and $\mathbf{q}_2 = (s_2, \mathbf{v}_2)$

  then

$$\mathbf{q}_1 \mathbf{q}_2 = (s_1 s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, s_1 \mathbf{v}_2 + s_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)$$

  where $\times$ is cross product, and $\cdot$ is dot product.

- Same as matrix multiplication, quaternion multiplication is **not commutative**.

Quaternion Multiplication (cont.)

An easy example that we can verify:

Let rotation R_1 be the rotation $\text{rotation}_x(\alpha)$ and let R_2 be $\text{rotation}_x(\beta)$. Then we know the resultant rotation R should be $\text{rotation}_x(\alpha+\beta)$.

The corresponding unit quaternions are: $\mathbf{q}_1 = (\cos(\frac{\alpha}{2}), \sin(\frac{\alpha}{2}), 0, 0)$ and $\mathbf{q}_2 = (\cos(\frac{\beta}{2}), \sin(\frac{\beta}{2}), 0, 0)$. Applying the multiplication formula, we get

$$\mathbf{q} = \begin{pmatrix} \cos(\alpha/2)\cos(\beta/2) - \sin(\alpha/2)\sin(\beta/2) \\ \cos(\alpha/2)\sin(\beta/2) + \cos(\beta/2)\sin(\alpha/2) \\ 0 \\ 0 \end{pmatrix}$$

Recall that $\mathbf{q}_1\mathbf{q}_2 = (s_1s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2,\ \ s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)$. Here we have $s_1 = \cos(\alpha/2)$, $s_2 = \cos(\beta/2)$, $\mathbf{v}_1 = (\sin(\alpha/2), 0, 0)$, and $\mathbf{v}_2 = (\sin(\beta/2), 0, 0)$.

Quaternion Multiplication (cont.)

Using the rules

$\cos(A+B) = \cos(A)\cos(B) - \sin(A)\sin(B)$ and

$\sin(A+B) = \sin(A)\cos(B) + \cos(A)\sin(B)$

we get the resultant quaternion

$$\mathbf{q} = \begin{pmatrix} \cos(\frac{\alpha+\beta}{2}) \\ \sin(\frac{\alpha+\beta}{2}) \\ 0 \\ 0 \end{pmatrix}$$

which represents the rotation about the x-axis an angle $\alpha + \beta$.

Recall that $\mathbf{q}_1\mathbf{q}_2 = (s_1s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2,\ \ s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)$. Here we have $s_1 = \cos(\alpha/2)$, $s_2 = \cos(\beta/2)$, $\mathbf{v}_1 = (\sin(\alpha/2), 0, 0)$, and $\mathbf{v}_2 = (\sin(\beta/2), 0, 0)$.

Rotating Points and Vectors using Quaternions

- Let $\mathbf{p} = (x, y, z)$ and $\mathbf{v} = (u, v, w)$ be a point and a vector in 3D. Let $\mathbf{q} = (a, \mathbf{w})$ be the unit quaternion representing the 3D rotation matrix $\mathbf{R}$. Suppose that we want to transform $\mathbf{p}$ and $\mathbf{v}$ by $\mathbf{R}$. i.e., we want to compute $\mathbf{p}' = \mathbf{R}\mathbf{p}$ and $\mathbf{v}' = \mathbf{R}\mathbf{v}$. How do we achieve the same result using the unit quaternion $\mathbf{q}$?

- We need to rewrite both $\mathbf{p}$ and $\mathbf{v}$ as 4D entities by letting $\mathbf{p} = (0, x, y, z)$ and $\mathbf{v} = (0, u, v, w)$. Then

$$\mathbf{p}' = \mathbf{q}\,\mathbf{p}\,\mathbf{q}^{-1}$$
$$\mathbf{v}' = \mathbf{q}\,\mathbf{v}\,\mathbf{q}^{-1}$$

where $\mathbf{q}^{-1}$ denotes the inverse of $\mathbf{q}$.

As unit quaternions cannot be used for translation, it doesn't matter whether we have a 0 or a 1 here.

Both the outputs $\mathbf{p}'$ and $\mathbf{v}'$ are 4D entities. Their 1st components would be 0. To convert them back to the 3D space, we can simply drop the 1st component.

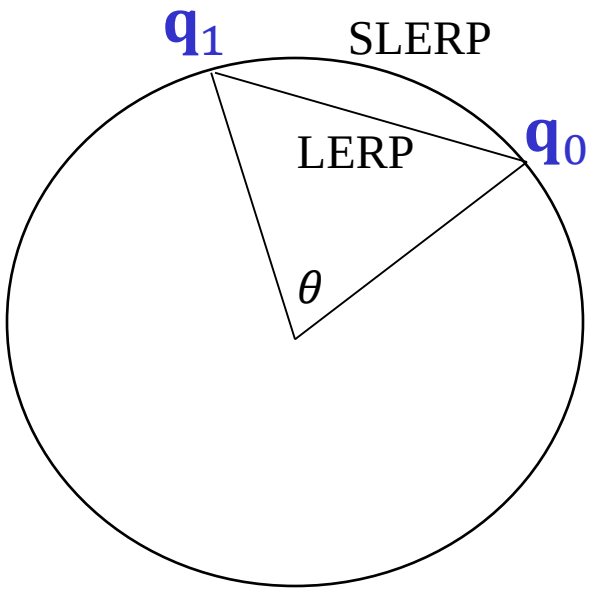
- Unit quaternions cannot be used for scaling and translation.

Interpolating Quaternions

- 2 ways of interpolating quaternions
  - LERP (Linear interpolation)

$$\mathrm{LERP}(\mathbf{q}_0, \mathbf{q}_1, t) = (1-t)\mathbf{q}_0 + t\mathbf{q}_1$$

The interpolated quaternion for a given *t* value.



  - SLERP (spherical linear interpolation)

$$\mathrm{SLERP}(\mathbf{q}_0, \mathbf{q}_1, t) = \frac{\sin((1-t)\theta)\mathbf{q}_0}{\sin(\theta)} + \frac{\sin(t\theta)\mathbf{q}_1}{\sin(\theta)}$$

where $\mathbf{q}_0$ and $\mathbf{q}_1$ are the quaternions that describe the initial and final orientations (= rotation), parameter $t = 0 \cdots 1$, and θ is the angle between $\mathbf{q}_0$ and $\mathbf{q}_1$, i.e.,

$$\theta = \arccos(\mathbf{q}_0 \cdot \mathbf{q}_1)$$

Converting Unit Quaternions to Rotation Matrices

- The easiest way to convert a unit quaternion $\mathbf{q} = (a, \mathbf{v})$ to a 3×3 rotation matrix $\mathbf{R}$ is using the Rodriques' rotation formula:

$$\mathbf{R} = \mathbf{I} + \sin(\theta)\,\text{skew}(\mathbf{v}) + (1 - \cos(\theta))\,\mathbf{v}\,\mathbf{v}^T$$
$$\mathbf{R}^T = \mathbf{I} - \sin(\theta)\,\text{skew}(\mathbf{v}) + (1 - \cos(\theta))\,\mathbf{v}\,\mathbf{v}^T$$

where

- $\mathbf{I}$ denotes the 3×3 identity matrix;
- θ can be obtained from the 1st component of $\mathbf{q}$ as follows: $\theta = 2\arccos(a)$;
- $\text{skew}(\mathbf{v})$ denotes the skew-symmetric matrix composed from the elements of $\mathbf{v} = (v_1, v_2, v_3)$:

$$\text{skew}(\mathbf{v}) = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix}$$

Further Learning Resources

Books:

- Computer graphics with OpenGL, Hearn baker
- Fundamentals of Computer Graphics, Peter Shirley and Steve Marschner

Useful references on the web:

https://eater.net/quaternions/video/intro

- Wikipedia:
  - http://en.wikipedia.org/wiki/Quaternion
    http://en.wikipedia.org/wiki/Gimbal_lock
- Youtube:
  - Quaternion and quaternion interpolation :
    https://www.youtube.com/watch?v=BXajpAy5-UI
  - Comparing LERP and SLERP:
    https://www.youtube.com/watch?v=kaGs4rGa97U