Quaternions and Exponentials

Michael Kazhdan

(601.457/657)
Announcements

• Do you care if the second midterm is Dec. 1 (Wednesday) instead of Dec. 3 (Friday)?
Recall

We saw two different methods for interpolating/approximating between rotations:

- **Normalization**: (SVD) Blend as $3 \times 3$ matrices and then map to the closest rotation.
  - Requires SVD
  - Works in a 9-dimensional space

- **Parameterization**: (Euler Angles) Compute the parameter values, blend those, and then evaluate at the blended values.
  - Parameterization is not uniform (e.g. dense sampling near poles)
Overview

• Math review
  ◦ Cross products
  ◦ Symmetric matrices
  ◦ Complex numbers
  ◦ The exponential map

• Quaternions

• The exponential map

• OpenGL and Texture-Mapping
Cross Product

Given 3D vectors \( \mathbf{u} = (u_1, u_2, u_3) \) and \( \mathbf{v} = (v_1, v_2, v_3) \) the cross product of \( \mathbf{u} \) and \( \mathbf{v} \) is:

\[
\mathbf{u} \times \mathbf{v} = 
\begin{pmatrix}
    u_2v_3 - u_3v_2 \\
    u_3v_1 - u_1v_3 \\
    u_1v_2 - u_2v_1
\end{pmatrix}
\]

Properties:
- The cross product is orthogonal to both \( \mathbf{u} \) and \( \mathbf{v} \).
- The vectors \( \mathbf{u}, \mathbf{v}, \mathbf{u} \times \mathbf{v} \) align with the right hand rule.
- The length of the cross product is equal to the area of the parallelogram defined by \( \mathbf{u} \) and \( \mathbf{v} \).
- \( \mathbf{u} \times \mathbf{v} = -\mathbf{v} \times \mathbf{u} \)
- \( \mathbf{u} \times (\mathbf{v} + \mathbf{w}) = \mathbf{u} \times \mathbf{v} + \mathbf{u} \times \mathbf{w} \)
- \( (t\mathbf{u}) \times \mathbf{v} = t(\mathbf{u} \times \mathbf{v}) \)
(Skew) Symmetric Matrices

A matrix $\mathbf{M}$ is symmetric if:

$$M_{ij} = M_{ji} \iff \mathbf{M} = \mathbf{M}^\top$$

A matrix $\mathbf{M}$ is skew-symmetric if:

$$M_{ij} = -M_{ji} \iff \mathbf{M} = -\mathbf{M}^\top$$

(Skew) Symmetric matrices are closed under addition and scaling:

- If $\mathbf{A} = \mathbf{A}^\top$ and $\mathbf{B} = \mathbf{B}^\top$, then $(\mathbf{A} + \mathbf{B}) = (\mathbf{A} + \mathbf{B})^\top$.
- If $\mathbf{A} = -\mathbf{A}^\top$ and $\mathbf{B} = -\mathbf{B}^\top$, then $(\mathbf{A} + \mathbf{B}) = -(\mathbf{A} + \mathbf{B})^\top$.
- If $\mathbf{A} = \mathbf{A}^\top$ then $(\alpha \mathbf{A}) = (\alpha \mathbf{A})^\top$.
- If $\mathbf{A} = -\mathbf{A}^\top$ then $(\alpha \mathbf{A}) = -(\alpha \mathbf{A})^\top$. 
Complex Numbers

Complex numbers are extensions of the real numbers, incorporating an imaginary value:

\[ a + ib \]

We add complex numbers together by summing the real and imaginary components:

\[ (a_1 + ib_1) + (a_2 + ib_2) = (a_1 + a_2) + i(b_1 + b_2) \]

Squaring the imaginary component gives:

\[ i^2 = -1 \]

The product of two complex numbers is:

\[
(a_1 + ib_1) \times (a_2 + ib_2) = (a_1a_2 - b_1b_2) + i(a_1b_2 + a_2b_1)
\]
Complex Numbers

• Given a complex number \( c = a + ib \)
  
  ◦ The **conjugate** of \( c \) is:
    \[
    \bar{c} = a - ib
    \]

  ◦ The **norm** of \( c \) is the real value:
    \[
    \|c\| = \sqrt{a^2 + b^2} = \sqrt{c\bar{c}}
    \]

  ◦ The **reciprocal** of \( c \) (assuming \( c \neq 0 \)) is defined by dividing the conjugate of \( c \) by the square norm:
    \[
    \frac{1}{c} = \frac{1}{c} \cdot \frac{\bar{c}}{\|c\|^2} = \frac{\bar{c}}{\|c\|^2}
    \]
The Exponential Map

The exponential map is a map from real values to non-negative real values:

\[ \exp : \mathbb{R} \to \mathbb{R}^{>0} \]

The inverse is the logarithm map, taking positive real values to real values:

\[ \ln : \mathbb{R}^{>0} \to \mathbb{R} \]
The Exponential Map

Properties:

• $\exp(0) = 1$

• $\frac{\partial \exp(t\alpha)}{\partial t} \bigg|_{t=0} = \alpha$

• $\exp(\ln(t)) = t$
The Exponential Map

Taylor Expansion:

We can approximate the exponential map by its Taylor Expansion around \( t = 0 \):

\[
\exp(s) = 1 + s + \frac{1}{2!} s^2 + \cdots + \frac{1}{n!} s^n + \cdots
\]

We can approximate the logarithm map by its Taylor Expansion around \( t = 1 \):

\[
\ln(s) = (s - 1) - \frac{(s - 1)^2}{2} + \cdots + (-1)^{n+1} \frac{(s - 1)^n}{n} + \cdots
\]
Overview

• Math review
• Quaternions
• The exponential map
• OpenGL and Texture-Mapping
Quaternions

Normalization:

• Find a representation of rotations that makes it easy to map the blend of rotations to the closest rotation
Quaternions

Quaternions are extensions of complex numbers, with three imaginary values instead of one:

\[ a + ib + jc + kd \]

Like the complex numbers, we can add quaternions together by summing the individual components:

\[
(a_1 + ib_1 + jc_1 + kd_1) \\
+ (a_2 + ib_2 + jc_1 + kd_2) \\
= (a_1 + a_2) + i(b_1 + b_2) + j(c_1 + c_2) + k(d_1 + d_2)
\]
Quaternions

Quaternions are extensions of complex numbers, with three imaginary values instead of one:

\[ a + ib + jc + kd \]

Like the imaginary component of complex numbers, squaring the components gives:

\[ i^2 = j^2 = k^2 = -1 \]

The multiplication rules are more complex:

\[ ij = k \quad ik = -j \quad jk = i \]
\[ ji = -k \quad ki = j \quad kj = -i \]

Note:
Multiplication of quaternions is not commutative – the result of the multiplication depends on the order in which it is done
More generally, the product of two quaternions is:

\[
(a_1 + ib_1 + jc_1 + kd_1) \times (a_2 + ib_2 + jc_2 + kd_2)
\]

\[
= (a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2) + i(a_1 b_2 + a_2 b_1 + c_1 d_2 - c_2 d_1) + j(a_1 c_2 + a_2 c_1 - b_1 d_2 + b_2 d_1) + k(a_1 d_2 + a_2 d_1 + b_1 c_2 - b_2 c_1)
\]

\[
i^2 = j^2 = k^2 = -1
\]

\[
ij = k \quad ik = -j \quad jk = i
\]

\[
ji = -k \quad ki = j \quad kj = -i
\]
Quaternions

As with complex numbers, the **conjugate** of a quaternion \( q = a + ib + jc + kd \) is:

\[
\bar{q} = a - ib - jc - kd
\]

As with complex numbers, the **norm** of a quaternion \( q = a + ib + jc + kd \) is:

\[
\|q\| = \sqrt{a^2 + b^2 + c^2 + d^2} = \sqrt{q \bar{q}}
\]

As with complex numbers, the **reciprocal** is defined by dividing the conjugate by the square norm:

\[
\frac{1}{q} = \frac{1}{q} \cdot \frac{\bar{q}}{\bar{q}} = \frac{\bar{q}}{\|q\|^2}
\]
Quaternions

One way to express a quaternion is as a pair consisting of the real value and the 3D vector consisting of the imaginary components:

\[ q = (\alpha, \mathbf{w}) \quad \text{with} \quad \alpha = a \quad \text{and} \quad \mathbf{w} = (b, c, d) \]

In this representation, multiplication becomes:

\[
q_1 \cdot q_2 = (\alpha_1, \mathbf{w}_1) \cdot (\alpha_2, \mathbf{w}_2) = (\alpha_1 \cdot \alpha_2 - \langle \mathbf{w}_1, \mathbf{w}_2 \rangle, \alpha_1 \cdot \mathbf{w}_2 + \alpha_2 \cdot \mathbf{w}_1 + \mathbf{w}_1 \times \mathbf{w}_2)
\]

\[
q_1 \cdot q_2 = (a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2) + i (a_1 b_2 + a_2 b_1 + c_1 d_2 - c_2 d_1) + j (a_1 c_2 + a_2 c_1 - b_1 d_2 + b_2 d_1) + k(a_1 d_2 + a_2 d_1 + b_1 c_2 - b_2 c_1)
\]
Quaternions

We can also think of points in 3D as (purely imaginary) quaternions:

$$(x, y, z) \rightarrow ix + jy + kz = (0, \mathbf{w})$$

Given a unit quaternion $q$ and an imaginary quaternion (3D point) $p$, consider the map:

$q(p) = qp\bar{q}$
Quaternions

\[ q(p) = qp\bar{q} \]

\[ q_1 \cdot q_2 = (\alpha_1 \cdot \alpha_2 - \langle w_1, w_2 \rangle, \alpha_1 \cdot w_2 + \alpha_2 \cdot w_1 + w_1 \times w_2) \]

Claim:

1. The map takes 3D points to 3D points

\[ q(p) = (\alpha_q, w_q)(0, w_p)(\alpha_q, -w_q) \]

\[ = (-\langle w_q, w_p \rangle, \alpha_q w_p + w_q \times w_p)(\alpha_q, -w_q) \]

\[ = (-\alpha_q \langle w_q, w_p \rangle + \alpha_q \langle w_p, w_q \rangle + \langle w_q \times w_p, w_q \rangle, ...) \]

\[ = (0, ...) \]
Quaternions

\[ q(p) = qp\bar{q} \]
\[ q_1 \cdot q_2 = (\alpha_1 \cdot \alpha_2 - \langle \mathbf{w}_1, \mathbf{w}_2 \rangle, \alpha_1 \cdot \mathbf{w}_2 + \alpha_2 \cdot \mathbf{w}_1 + \mathbf{w}_1 \times \mathbf{w}_2) \]

Claim:

1. The map takes 3D points to 3D points
2. The map is linear

\[ q(a \cdot p_1 + b \cdot p_2) = q(a \cdot p_1 + b \cdot p_2)\bar{q} \]
\[ = a \cdot qp_q\bar{q} + b \cdot qp_2\bar{q} \]
\[ = a \cdot q(p_1) + b \cdot q(p_2) \]
Quaternions

\[ q(p) = qp\bar{q} \]

\[ q_1 \cdot q_2 = (\alpha_1 \cdot \alpha_2 - \langle \mathbf{w}_1, \mathbf{w}_2 \rangle, \alpha_1 \cdot \mathbf{w}_2 + \alpha_2 \cdot \mathbf{w}_1 + \mathbf{w}_1 \times \mathbf{w}_2) \]

Claim:

1. The map takes 3D points to 3D points
2. The map is linear
3. The map is norm-preserving

\[ |q(p)| = |qp\bar{q}| = |q||p||\bar{q}| = |p| \]

since \( q \) is a unit quaternion
Quaternions

\[ q(p) = qp\bar{q} \]

\[ q_1 \cdot q_2 = (\alpha_1 \cdot \alpha_2 - \langle w_1, w_2 \rangle, \alpha_1 \cdot w_2 + \alpha_2 \cdot w_1 + w_1 \times w_2) \]

Claim:

1. The map takes 3D points to 3D points
2. The map is linear
3. The map is norm-preserving

\[\Downarrow\]

When \( q \) is a unit quaternion, the map \( p \rightarrow qp\bar{q} \) is an orthogonal transformation (specifically, a rotation).
Quaternions and Rotations

If \( q = a + ib + jc + kd \) is a unit quaternion (\( \|q\| = 1 \)), we can associate \( q \) with the rotation:

\[
R(q) = \begin{pmatrix}
1 - 2c^2 - 2d^2 & 2bc - 2ad & 2bd + 2ac \\
2bc + 2ad & 1 - 2b^2 - 2d^2 & 2cd - 2ab \\
2bd - 2ac & 2cd + 2ab & 1 - 2b^2 - 2c^2
\end{pmatrix}
\]

Note that because all of the terms are quadratic, the rotation associated with \( q \) is the same as the rotation associated with \(-q\).
Quaternions and Rotations

If \( q = a + ib + jc + kd \) is a unit quaternion (\( \|q\| = 1 \)), we can associate \( q \) with the rotation:

\[
R(q) = \begin{pmatrix}
1 - 2c^2 - 2d^2 & 2bc - 2ad & 2bd + 2ac \\
2bc + 2ad & 1 - 2b^2 - 2d^2 & 2cd - 2ab \\
2bd - 2ac & 2cd + 2ab & 1 - 2b^2 - 2c^2
\end{pmatrix}
\]

Because \( q \) is a unit quaternion, we have:

\[
\|q\|^2 = \|(\alpha, w)\|^2 = \alpha^2 + \|w\|^2 = 1
\]

Or equivalently, if we set \( v = w/\|w\| \), there exists \( \theta \) such that:

\[
q = \left( \cos \frac{\theta}{2}, \sin \frac{\theta}{2} v \right)
\]
Quaternions and Rotations

If \( q = a + ib + jc + kd \) is a unit quaternion (\( \|q\| = 1 \)), we can associate \( q \) with the rotation:

\[
R(q) = \begin{pmatrix}
1 - 2c^2 - 2d^2 & 2bc - 2ad & 2bd + 2ac \\
2bc + 2ad & 1 - 2b^2 - 2d^2 & 2cd - 2ab \\
2bd - 2ac & 2cd + 2ab & 1 - 2b^2 - 2c^2
\end{pmatrix}
\]

Because \( q \) is a unit quaternion, we have:

\[
q = \left( \cos \frac{\theta}{2}, \sin \frac{\theta}{2} \mathbf{v} \right)
\]

It turns out that \( q \) corresponds to the rotation whose:

- axis of rotation is \( \mathbf{v} \), and
- angle of rotation is \( \theta \).
Quaternions and Rotations

If \( q = a + ib + jc + kd \) is a unit quaternion (\( \|q\| = 1 \)), we can associate \( q \) with the rotation:

\[
R(q) = \begin{pmatrix}
1 - 2c^2 - 2d^2 & 2bc - 2ad & 2bd + 2ac \\
2bc + 2ad & 1 - 2b^2 - 2d^2 & 2cd - 2ab \\
2bd - 2ac & 2cd + 2ab & 1 - 2b^2 - 2c^2
\end{pmatrix}
\]

Because \( q \) is a unit quaternion, we have:

In particular, if we express rotations in the axis-angle representation, we can compute the composition by multiplying quaternions.

It turns out that \( q \) corresponds to the rotation whose:

- axis of rotation is \( \mathbf{v} \), and
- angle of rotation is \( \theta \).
Quaternions

Instead of blending rotations \( \{ \mathbf{R}_0, \ldots, \mathbf{R}_{n-1} \} \) and then normalizing using SVD, we can blend the quaternions and then normalize them:

- For each \( \mathbf{R}_i \), compute the quaternion rep. \( (\alpha_i, \mathbf{w}_i) \)
Quaternions

Instead of blending rotations \( \{R_0, \ldots, R_{n-1} \} \) and then normalizing using SVD, we can blend the quaternions and then normalize them:

- For each \( R_i \), compute the quaternion rep. \((\alpha_i, w_i)\)
- Interpolate/Approximate the quaternions:
Quaternions

Instead of blending rotations \( \{ \mathbf{R}_0, \ldots, \mathbf{R}_{n-1} \} \) and then normalizing using SVD, we can blend the quaternions and then normalize them:

- For each \( \mathbf{R}_i \), compute the quaternion rep. \((\alpha_i, \mathbf{w}_i)\)
- Interpolate/Approximate the quaternions:
  - Linear Interpolation:
    \[
    \alpha_k(t) = (1 - t)\alpha_k + t\alpha_{k+1}
    \]
    \[
    \mathbf{w}_k(t) = (1 - t)\mathbf{w}_k + t\mathbf{w}_{k+1}
    \]
Quaternions

Instead of blending rotations \(\{\mathbf{R}_0, \ldots, \mathbf{R}_{n-1}\}\) and then normalizing using SVD, we can blend the quaternions and then normalize them:

- For each \(\mathbf{R}_i\), compute the quaternion rep. \((\alpha_i, \mathbf{w}_i)\)
- Interpolate/Approximate the quaternions:
  - Linear Interpolation
  - Catmull-Rom Interpolation:
    \[
    \alpha_k(t) = CR_0(t)\alpha_{k-1} + CR_1(t)\alpha_k + CR_2(t)\alpha_{k+1} + CR_3(t)\alpha_{k+2}
    \]
    \[
    \mathbf{w}_k(t) = CR_0(t)\mathbf{w}_{k-1} + CR_1(t)\mathbf{w}_k + CR_2(t)\mathbf{w}_{k+1} + CR_3(t)\mathbf{w}_{k+2}
    \]
Quaternions

Instead of blending rotations \{\textbf{R}_0, \ldots, \textbf{R}_{n-1}\} and then normalizing using SVD, we can blend the quaternions and then normalize them:

- For each \textbf{R}_i, compute the quaternion rep. \((\alpha_i, \textbf{w}_i)\)
- Interpolate/Approximate the quaternions:
  - Linear Interpolation
  - Catmull-Rom Interpolation
  - Uniform Cubic B-Spline Approximation:
    \[
    \alpha_k(t) = B_{0,3}(t)\alpha_{k-1} + B_{1,3}(t)\alpha_k + B_{2,3}(t)\alpha_{k+1} + B_{3,3}(t)\alpha_{k+2}
    \]
    \[
    \textbf{w}_k(t) = B_{0,3}(t)\textbf{w}_{k-1} + B_{1,3}(t)\textbf{w}_{k} + B_{2,3}(t)\textbf{w}_{k+1} + B_{3,3}(t)\textbf{w}_{k+2}
    \]
Quaternions

Instead of blending rotations \{\mathbf{R}_0, \ldots, \mathbf{R}_{n-1}\} and then normalizing using SVD, we can blend the quaternions and then normalize them:

- For each \(\mathbf{R}_i\), compute the quaternion rep. \((\alpha_i, \mathbf{w}_i)\)
- Interpolate/Approximate the quaternions:
  - Linear Interpolation
  - Catmull-Rom Interpolation
  - Uniform Cubic B-Spline Approximation
- Set the value of the in-between rotation to be the normalized quaternion:

\[
q_k(t) = \frac{(\alpha_k(t), \mathbf{w}_k(t))}{\big\| (\alpha_k(t), \mathbf{w}_k(t)) \big\|}
\]
Instead of blending rotations \(\{\mathbf{R}_0, ..., \mathbf{R}_{n-1}\}\) and then normalizing using SVD, we can blend the quaternions and then normalize them:

- For each \(\mathbf{R}_i\), compute the quaternion rep. \((\alpha_i, \mathbf{w}_i)\)
- Interpolate/Approximate the quaternions:

Note:
- Using SVD, we interpolated in the \((9 = 3 \times 3)\)-dimensional space of matrices and then normalized.
- With quaternions we interpolate in the 4-dimensional space of quaternions and normalize.

\[
q_k(t) = \frac{(\alpha_k(t), \mathbf{w}_k(t))}{\| (\alpha_k(t), \mathbf{w}_k(t)) \|}
\]
Quaternions

As with points on the circle/sphere, this type of interpolation/approximation has the limitation:

• Uniform sampling in quaternion space does not result in uniform sampling in rotation space.

\[ \Phi(t) = (1 - t)p_0 + tp_1 \]

\[ \tilde{\Phi}(t) = \frac{\Phi(t)}{||\Phi(t)||} \]
Quaternions

As with points on the circle/sphere, this type of interpolation/approximation has the limitation:

- Uniform sampling in quaternion space does not result in uniform sampling in rotation space.

Additionally, since $\mathbf{R}(-q) = \mathbf{R}(q)$ there are two different quaternions we can associate with a rotation, so the mapping is not well-defined.
Overview

- Math review
- Quaternions
- The exponential map
- OpenGL and Texture-Mapping
The Exponential Map

Parametrization:

• Find a canonical way to parametrize rotations so that there is little distortion
Geodesics

Given a surface $S(u, v)$ a geodesic is a curve that is (locally) the shortest path between two points.

$S(u, v) = (\cos u \cos v, \sin u, \cos u \sin v)$
Geodesics

Given a manifold (a $d$-dimensional surface) a geodesic is a curve that is (locally) the shortest path between two points.
Tangent Spaces

Given a curve $C(t)$, the tangent line to the curve at a point $p_0 = C(t_0)$ is the line that most closely approximates the curve $C(t)$ at the point $p_0$. 
Tangent Spaces

Given a curve $\mathbf{C}(t)$, the **tangent line** to the curve at a point $\mathbf{p}_0 = \mathbf{C}(t_0)$ is the line that most closely approximates the curve $\mathbf{C}(t)$ at the point $\mathbf{p}_0$.

This is the line through $\mathbf{p}_0$ with direction $\mathbf{C}'(t_0)$. 
Tangent Spaces

Given a surface $S(u, v)$ the tangent plane to the curve at a point $p_0 = S(u_0, v_0)$ is the plane that most closely approximates $S(u, v)$ at the point $p_0$. 

$S(u, v) = (\cos u \cos v, \sin u, \cos u \sin v)$
Tangent Spaces

Given a surface $S(u, v)$ the tangent plane to the curve at a point $p_0 = S(u_0, v_0)$ is the plane that most closely approximates $S(u, v)$ at the point $p_0$.

This is the plane through $p_0$, spanned by:

$$\frac{\partial S(u,v)}{\partial u} \bigg|_{(u_0,v_0)} \quad \text{and} \quad \frac{\partial S(u,v)}{\partial v} \bigg|_{(u_0,v_0)}$$

$$S(u,v) = (\cos u \cos v, \sin u, \cos u \sin v)$$
Tangent Spaces

Given a manifold (a $d$-dimensional surface) the tangent space to the manifold at a point $p_0$ on the manifold is the $d$-dimensional plane that most closely approximates the manifold at the point $p_0$. 
The Exponential Map

Given a curve $\mathbf{C}(t)$, the exponential at $\mathbf{p}_0 = \mathbf{C}(t_0)$ is a map that sends points in the tangent space of $\mathbf{p}_0$ to the curve $\mathbf{C}(t)$. 

$\mathbf{C}(t) \xrightarrow{\text{exp}_{\mathbf{p}_0}(\mathbf{v})} \mathbf{C}(t_0)$

Tangent Line
The Exponential Map

Given a curve \( \mathbf{C}(t) \), the exponential at \( \mathbf{p}_0 = \mathbf{C}(t_0) \) is a map that sends points in the tangent space of \( \mathbf{p}_0 \) to the curve \( \mathbf{C}(t) \).

The distance along the curve from \( \mathbf{p}_0 \) to point \( \exp_{\mathbf{p}_0}(\mathbf{v}) \) is equal to \( ||\mathbf{v}|| \).

Note: This is not the distance from the tangent line to the closest point on the curve.
The Exponential Map

Given a surface $S(u, v)$, the exponential at the point $p_0 = S(u_0, v_0)$ is a map that sends points in the tangent plane of $p_0$ to the surface $S(u, v)$.
The Exponential Map

Given a surface $S(u, v)$, the exponential at the point $p_0 = S(u_0, v_0)$ is a map that sends points in the tangent plane of $p_0$ to the surface $S(u, v)$.

If we fix a vector $\vec{w}$ in the tangent space of $p_0$, then the curve $\exp_{p_0}(t\vec{w})$ will be a geodesic, leaving $p_0$ in direction $\vec{w}$ and will have length equal to $\|t\vec{w}\|$.
The Exponential Map

Given a surface $S(u, v)$, the exponential at the point $p_0 = S(u_0, v_0)$ is a map that sends points in the tangent plane of $p_0$ to the surface $S(u, v)$.

If we fix a vector $\vec{w}$ in the tangent space of $p_0$, then the curve $\exp_{p_0}(t\vec{w})$ will be a geodesic, leaving $p_0$ in direction $\vec{w}$ and will have length equal to $\|t\vec{w}\|$.
The Exponential Map

Given a surface $S(u, v)$, the exponential at the point $p_0 = S(u_0, v_0)$ is a map that sends points in the tangent plane of $p_0$ to the surface $S(u, v)$.

If we fix a vector $\vec{w}$ in the tangent space of $p_0$, then the curve $\exp_{p_0}(t\vec{w})$ will be a geodesic, leaving $p_0$ in direction $\vec{w}$ and will have length equal to $\|t\vec{w}\|$. 
The Exponential Map

Given a surface $S(u, v)$, the exponential at the point $p_0 = S(u_0, v_0)$ is a map that sends points in the tangent plane of $p_0$ to the surface $S(u, v)$.

If we fix a vector $\vec{w}$ in the tangent space of $p_0$, then the curve $\exp_{p_0}(t\vec{w})$ will be a geodesic, leaving $p_0$ in direction $\vec{w}$ and will have length equal to $\|t\vec{w}\|$. 
Given a surface $S(u, v)$, the exponential at the point $p_0 = S(u_0, v_0)$ is a map that sends points in the tangent plane of $p_0$ to the surface $S(u, v)$.

If we fix a vector $\vec{w}$ in the tangent space of $p_0$, then the curve $\exp_{p_0}(t\vec{w})$ will be a geodesic, leaving $p_0$ in direction $\vec{w}$ and will have length equal to $\|t\vec{w}\|$.
The Exponential Map

Given a surface $S(u, v)$, the **exponential** at the point $p_0 = S(u_0, v_0)$ is a map that sends points in the tangent plane of $p_0$ to the surface $S(u, v)$.

If we fix a vector $\vec{w}$ in the tangent space of $p_0$, then the curve $\exp_{p_0}(t\vec{w})$ will be a geodesic, leaving $p_0$ in direction $\vec{w}$ and will have length equal to $\|t\vec{w}\|$.
The Exponential Map

Given a manifold (a $d$-dimensional surface), the exponential at point $p_0$ on the manifold is a map that sends points in the tangent plane of $p_0$ to the manifold.

If we fix a vector $\vec{w}$ in the tangent space of $p_0$, then the curve $\exp_{p_0}(t\vec{w})$ will be a geodesic, leaving $p_0$ in direction $\vec{w}$ and will have length equal to $\|t\vec{w}\|$.
The Logarithm Map

For a point $p_0$ on a curve/surface/manifold, the logarithm is the inverse of the exponential, sending points on the curve/surface/manifold back into the tangent space of $p_0$. 
The Exponential Map

Example:

Let $\mathbf{C}$ be the unit circle, the exponential map $\exp_{(1,0)}(t)$ is the map sending the point $t$ to the point $(\cos t, \sin t)$.
The Exponential Map

Example:

Let $\mathbb{C}$ be the unit circle, the exponential map $\exp_{(1,0)}(t)$ is the map sending the point $t$ to the point $(\cos t, \sin t)$.

Note: The exponential map is many-to-one: $\exp_{(1,0)}(t) = \exp_{(1,0)}(t + 2k\pi)$ so the logarithm is not unique.
The Exponential Map

Fact:

• Let $SO(n)$ be the space of $n \times n$ rotation matrices, $\exp_{\text{id.}}(S)$ is the map sending a skew-symmetric matrix $S$ to a rotation.
The Exponential Map

How do we compute the exponential map?
The Exponential Map

How do we compute the exponential map?

It is difficult to find a closed form solution, but for matrices we can use a Taylor series approximation:

$$\exp_{\text{id.}}(S) = \text{id.} + S + \frac{1}{2!} S^2 + \cdots + \frac{1}{n!} S^n + \cdots$$

In a similar manner, we can define the logarithm:

$$\ln_{\text{id.}}(R) = (R - \text{id.} ) - \frac{(R-\text{id.})^2}{2} + \cdots + (-1)^{n+1} \frac{(R-\text{id.})^n}{n} + \cdots$$
The Exponential Map

Properties:

- \( \exp_{\text{id.}}(0) = \text{id.} \)
- \( \frac{\partial \exp_{\text{id.}}(tS)}{\partial t} \bigg|_{t=0} = S \)
- \( \exp_{\text{id.}}(\ln_{\text{id.}} R) = R \)
Rotation Interpolation/Approximation

Given a collection of rotations \( \{R_0, ..., R_{n-1}\} \) we can generate a curve passing through/near the matrices:

- For each \( R_i \), compute the logarithm \( S_i = \ln_{id}(R_i) \)
Rotation Interpolation/Approximation

Given a collection of rotations \( \{R_0, \ldots, R_{n-1}\} \) we can generate a curve passing through/near the matrices:

- For each \( R_i \), compute the logarithm \( S_i = \ln_{\text{id}}(R_i) \)
- Interpolate/Approximate the logarithms:
Rotation Interpolation/Approximation

Given a collection of rotations \( \{\mathbf{R}_0, \ldots, \mathbf{R}_{n-1}\} \) we can generate a curve passing through/near the matrices:

- For each \( \mathbf{R}_i \), compute the logarithm \( \mathbf{S}_i = \ln_{\text{id}}(\mathbf{R}_i) \)
- Interpolate/Approximate the logarithms:
  - Linear Interpolation:
    \[
    \mathbf{S}_k(t) = (1 - t)\mathbf{S}_k + t\mathbf{S}_{k+1}
    \]
Rotation Interpolation/Approximation

Given a collection of rotations \( \{ \mathbf{R}_0, \ldots, \mathbf{R}_{n-1} \} \) we can generate a curve passing through/near the matrices:

- For each \( \mathbf{R}_i \), compute the logarithm \( \mathbf{S}_i = \ln_{\text{id.}}(\mathbf{R}_i) \)
- Interpolate/Approximate the logarithms:
  - Linear Interpolation:
  - Catmull-Rom Interpolation:

\[
\mathbf{S}_k(t) = C\mathbf{R}_0(t)\mathbf{S}_{k-1} + C\mathbf{R}_1(t)\mathbf{S}_k + C\mathbf{R}_1(t)\mathbf{S}_{k+1} + C\mathbf{R}_1(t)\mathbf{S}_{k+2}
\]
Rotation Interpolation/Approximation

Given a collection of rotations \( \{ \mathbf{R}_0, \ldots, \mathbf{R}_{n-1} \} \) we can generate a curve passing through/near the matrices:

- For each \( \mathbf{R}_i \), compute the logarithm \( \mathbf{S}_i = \ln_{\text{id}}(\mathbf{R}_i) \)
- Interpolate/Approximate the logarithms:
  - Linear Interpolation:
  - Catmull-Rom Interpolation:
  - Uniform Cubic B-Spline Approximation:

\[
\mathbf{S}_k(t) = B_{0,3}(t)\mathbf{S}_{k-1} + B_{1,3}(t)\mathbf{S}_k + B_{2,3}(t)\mathbf{S}_{k+1} + B_{3,3}(t)\mathbf{S}_{k+2}
\]
Rotation Interpolation/Approximation

Given a collection of rotations \( \{ \mathbf{R}_0, \ldots, \mathbf{R}_{n-1} \} \) we can generate a curve passing through/near the matrices:

- For each \( \mathbf{R}_i \), compute the logarithm \( \mathbf{S}_i = \ln_{\mathbf{I}}(\mathbf{R}_i) \)
- Interpolate/Approximate the logarithms:
  - Linear Interpolation:
  - Catmull-Rom Interpolation:
  - Uniform Cubic B-Spline Approximation:
- Set the value of the in-between rotation to be the exponent of the blended logarithms:
  \[ \mathbf{R}_k(t) = \exp_{\mathbf{I}}(\mathbf{S}_k(t)) \]
Rotation Interpolation/Approximation

Given a collection of rotations \( \{ \mathbf{R}_0, \ldots, \mathbf{R}_{n-1} \} \) we can generate a curve passing through/near the matrices:

- For each \( \mathbf{R}_i \), compute the logarithm \( \mathbf{S}_i = \ln_{\text{id}}(\mathbf{R}_i) \)
- Interpolate/Approximate the logarithms:

\[
\mathbf{R}_k(t) = \exp_{\text{id}}(t \mathbf{S}_k(t))
\]

Note:
Since the logarithm of rotations is a skew-symmetric matrix, and since skew-symmetric matrices are closed under addition and scaling, the weighted average \( \mathbf{S}_k(t) \) is also skew-symmetric, so its exponent has to be a rotation.

Warning:
Is taking the exponential/logarithm with respect to the identity the right thing to do? (e.g. Maybe we should take it with respect to some other point.)
Summary

In order to define in-between frames for an animation, we need to interpolate/approximate the transformations specified in the key-frames.

• For translation, we can just use splines

• For rotations, we need to ensure that the in-between transformations are also rotations:
  - Euler angles
  - Exponential map
  - SVD
  - Quaternions

  \( \text{In-between transformations are}\)
  \( \text{guaranteed to be rotations}\)

  \( \text{Normalize in-between transformations to}\)
  \( \text{turn them into the nearest rotations}\)
Overview

• Math review
• Quaternions
• The Exponential Map
• OpenGL and Texture-Mapping
OpenGL Textures

To use a texture you need to:

- Have a description of the texture and its properties on the GPU
- Have a handle (like a pointer, but represented as an unsigned integer) to the texture information on the GPU
- Enable texture mapping
- Specify which texture you will be using
OpenGL Textures

To get a texture handle, you have to ask OpenGL to generate a texture handle for you:

```c
glGenTextures( GLsizei, GLuint * );
```

- The first argument gives the number of handles you want OpenGL to generate.
- The second argument is the memory address of the array where OpenGL should write the handles.

Typically, you will generate textures one at a time.
OpenGL Textures

To specify that you will be using a texture handle for (2D) texture mapping:

```c
glBindTexture( GLenum , GLuint);
```

- The first argument describes the texture type.
- The second is the handle itself.

Note that you need to this both for rendering the texture and for specifying the texture properties (so OpenGL know which texture’s properties you’re setting).
OpenGL Textures

To specify the texture information you need to specify how the texture gets mapped and what the texture values are:

```c
glTexParameteri( GLenum , GLenum , Glint );
glTexImage2D( ... );
glTexEnvi( GLenum , GLenum , Glint );
```

You do not have to copy the texture values into a separate array. You can directly use the memory address of the first image pixel:

```c
&_image(0,0)
```
OpenGL Textures

To enable texture mapping you need to tell OpenGL that you will be using (2D) texture mapping, and which handle you will be using:

```c
    glEnable( GLenum );
    glBindTexture( GLenum , GLuint )
```

If the material does not have a texture associated with it, you should disable texture mapping, so that OpenGL doesn’t try using the texture from the previously specified material.

Recall that OpenGL is a state machine.