Geometry for Computer Vision

Lecture 7b

Rotations and Rigid body motion

Per-Erik Forssén

Overview

- Rotation group SO(3)
- Rotation averaging
- Rotation interpolation
- Rigid bodies, SE(3)
- SE(3) interpolation

The rotation group SO(3)

A **rotation**, **R**, is an action operating on points, **x**,**y**, in R^3:

$$\mathbf{y} = \mathbf{R}\mathbf{x}$$

1. It preserves distances:

$$||\mathbf{y}_k - \mathbf{y}_l|| = ||\mathbf{x}_k - \mathbf{x}_l||$$

2. It is orthogonal: $\mathbf{R}^T\mathbf{R} = \mathbf{I}$

3. It preserves handedness:

$$\mathbf{R}(\mathbf{x} \times \mathbf{y}) = (\mathbf{R}\mathbf{x}) \times (\mathbf{R}\mathbf{y})$$

The rotation group SO(3)

1. The **set** of all 3x3 matrices **R** that fulfill:

$$\Omega = \left\{ \mathbf{R} | \mathbf{R}^T \mathbf{R} = \mathbf{I}, \det \mathbf{R} = 1 \right\}$$

2. A **group operation**

$$\mathbf{R}_1, \mathbf{R}_2 \in \Omega \quad \Rightarrow \quad \mathbf{R}_1 \mathbf{R}_2 \in \Omega$$

3. An **identity element**

$$\mathbf{I}, \mathbf{R} \in \Omega \quad \Rightarrow \quad \mathbf{I}\mathbf{R} = \mathbf{R}\mathbf{I} = \mathbf{R}$$

4. An **inverse**

$$\mathbf{R}^{-1} = \mathbf{R}^T \in \Omega$$

The rotation group SO(3)

1. The **set** of all unit quaternions $\mathbf{q}$:

$$\mathbf{q} = (\cos \alpha/2, \hat{\mathbf{n}} \sin \alpha/2)$$

2. A **group operation**

$$\mathbf{q}_1, \mathbf{q}_2 \in \mathrm{Spin}(3) \quad \Rightarrow \quad \mathbf{q}_1 \mathbf{q}_2 \in \mathrm{Spin}(3)$$

3. An **identity element**

$$\mathbf{q}_I, \mathbf{q} \in \mathrm{Spin}(3) \quad \Rightarrow \quad \mathbf{q}_I \mathbf{q} = \mathbf{q}\mathbf{q}_I = \mathbf{q}$$

4. An **inverse**

$$\mathbf{q}^{-1} = \mathbf{q}^* \in \mathrm{Spin}(3)$$

The rotation group SO(3)

Intermediate rotations between two rotations $\mathbf{R}_1$ and $\mathbf{R}_2$ are obtained as:

$$\mathbf{R}(\mathbf{R}_1, \mathbf{R}_2, \lambda) = \mathbf{R}_1 \exp(\lambda \log(\mathbf{R}_1^T \mathbf{R}_2))$$

Spherical Linear Interpolation (SLeRP)

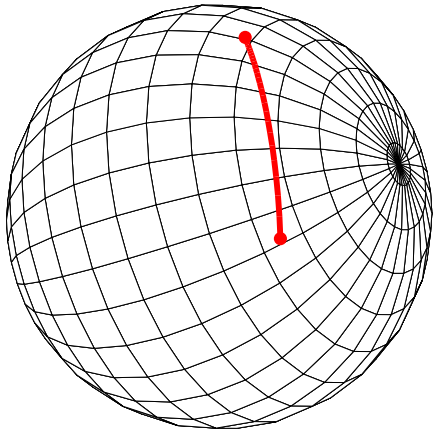
K. Shoemake, SIGGRAPH'85
(Paper for next week)

Uses group operations, and is thus **on SO(3)**

The rotation group SO(3)

The SLeRP construction is a **geodesic**:

The shortest trajectory between two points on a manifold.



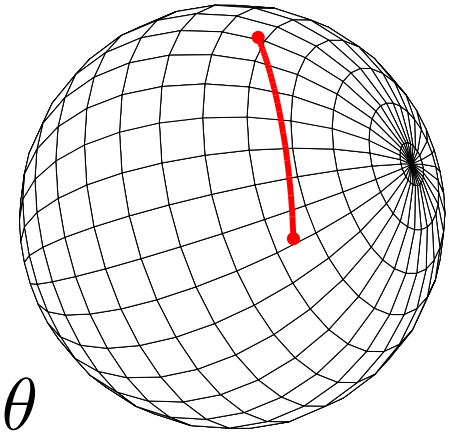
For quaternion SLeRP, the geodesic is a great arc on the unit ball in $\mathbf{R}^4$.

$$\mathbf{q}(\mathbf{q}_1, \mathbf{q}_2, \lambda) = \mathbf{q}_1 \exp(\lambda \log(\mathbf{q}_1^* \mathbf{q}_2))$$

The rotation group SO(3)

The SLeRP construction is a **geodesic**:

The shortest trajectory between two points on a manifold.



For quaternion SLeRP, the geodesic is a great arc on the unit ball in $\mathbf{R}^4$.

$$\mathbf{q}(\mathbf{q}_1, \mathbf{q}_2, \lambda) = \mathbf{q}_1 \exp(\lambda \log(\mathbf{q}_1^* \mathbf{q}_2))$$

$$= \frac{\sin(1-\lambda)\theta}{\sin\theta}\mathbf{q}_1 + \frac{\sin\lambda\theta}{\sin\theta}\mathbf{q}_2$$

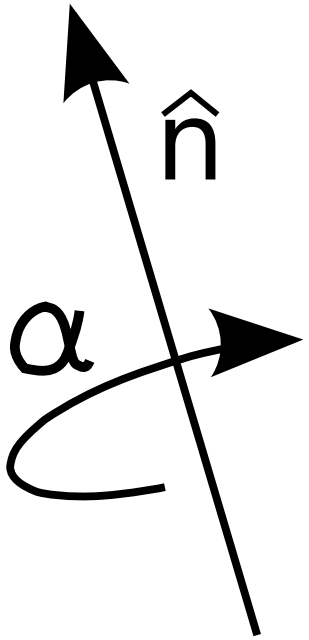
The rotation group SO(3)

Rotations may be more compactly represented using **axis-angle vectors**

This representation is closely related to the logarithm:

$$\mathbf{q} = (\cos \alpha/2, \hat{\mathbf{n}} \sin \alpha/2)$$

$$\log(\mathbf{q}) = (0, \alpha\hat{\mathbf{n}})$$



The rotation group SO(3)

The logarithm also induces a **natural metric** on SO(3):

$$\log(\mathbf{q}) = (0, \alpha\hat{\mathbf{n}})$$

$$\log(\mathbf{R}) = \alpha \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix}$$

The rotation group SO(3)

The logarithm also induces a **natural metric** on SO(3):

$$\log(\mathbf{q}) = (0, \alpha\hat{\mathbf{n}})$$

$$\log(\mathbf{R}) = \alpha \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix}$$

$$d(\mathbf{q}_1, \mathbf{q}_2) = ||\log(\mathbf{q}_1^* \mathbf{q}_2)||$$

$$d(\mathbf{R}_1, \mathbf{R}_2) = \frac{1}{\sqrt{2}} ||\log(\mathbf{R}_1^T \mathbf{R}_2)||$$

Rotation averaging

In Euclidean space an average vector is defined as:

$$\mathbf{x}_{\text{avg}} = \arg\min_{\mathbf{x}^*} \sum_{n=1}^{N} ||\mathbf{x}^* - \mathbf{x}_n||^2$$

with the well known solution:

$$\mathbf{x}_{\text{avg}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$$

Rotation averaging

In Euclidean space an average vector is defined as:

$$\mathbf{x}_{\text{avg}} = \arg\min_{\mathbf{x}^*} \sum_{n=1}^{N} ||\mathbf{x}^* - \mathbf{x}_n||^2$$

Averages are useful for:

- Fusion of several measurements
- Representative vectors in vector quantisation (e.g. K-means)
- etc.

Rotation averaging

For rotations, summing rotations or quaternions, and dividing by the number elements gives us a result outside SO(3), so this is not the way to average here.

$$\mathbf{R}_{\text{avg}} = \arg\min_{\mathbf{R}^*} \sum_{n=1}^{N} d(\mathbf{R}_n, \mathbf{R}^*)^2$$

Rotation averaging

For rotations, summing rotations or quaternions, and dividing by the number elements gives us a result outside SO(3), so this is not the way to average here.

$$\mathbf{R}_{\text{avg}} = \arg\min_{\mathbf{R}^*} \sum_{n=1}^{N} d(\mathbf{R}_n, \mathbf{R}^*)^2$$

instead, we should use the natural metric

$$\mathbf{R}_{\text{avg}} = \arg\min_{\mathbf{R}^*} \sum_{n=1}^{N} ||\log(\mathbf{R}_n^T \mathbf{R}^*)||^2$$

Rotation averaging

Computing the average rotation using the natural metric requires iterative non-linear optimization.

In [**Gramkow IJCV'01**] this is compared to averaging followed by orthogonalization, i.e.:

$$\mathbf{U}\mathbf{D}\mathbf{V}^T = \mathrm{svd}\left[\sum_{n=1}^{N} \mathbf{R}_n\right]$$

$$\mathbf{R}_{\mathrm{avg}} \approx \mathbf{U}\mathbf{S}\mathbf{V}^T\,, \quad \text{where} \quad \mathbf{S} = \mathrm{diag}(1, 1, \ldots, \det(\mathbf{U}\mathbf{V}^T)$$

Rotation averaging

Computing the average rotation using the natural metric requires iterative non-linear optimization.

In [**Gramkow IJCV'01**] this is compared to averaging followed by orthogonalization,

and to a re-normalised unit quaternion average:

$$\tilde{\mathbf{q}}_{\text{avg}} = \sum_{n=1}^{N} \mathbf{q}_n \qquad \mathbf{q}_{\text{avg}} \approx \tilde{\mathbf{q}}_{\text{avg}} / ||\mathbf{q}_{\text{avg}}||$$

Both turn out to be quite good approximations. Quaternions are slightly more accurate, and also faster.

Rotation averaging

Computing the average rotation using the natural metric requires iterative non-linear optimization.

In [**Gramkow IJCV'01**] this is compared to averaging followed by orthogonalization,

and to a re-normalised unit quaternion average:

$$\tilde{\mathbf{q}}_{\text{avg}} = \sum_{n=1}^{N} \mathbf{q}_n \qquad \mathbf{q}_{\text{avg}} \approx \tilde{\mathbf{q}}_{\text{avg}} / ||\mathbf{q}_{\text{avg}}||$$

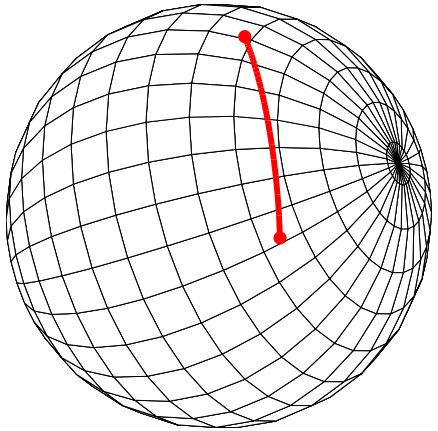
Both turn out to be quite good approximations. Quaternions are slightly more accurate, and also faster.
Also: Hartley suggests using the **L_1 norm** instead if the rotation set has outliers.

Rotation interpolation

For interpolation we have already mentioned the SLeRP construction.

$$\mathbf{q}(\mathbf{q}_1, \mathbf{q}_2, \lambda) = \mathbf{q}_1 \exp(\lambda \log(\mathbf{q}_1^* \mathbf{q}_2))$$

This is the **geodesic** between two rotations.



Rotation interpolation

Higher order curves were defined already by Shoemake in his SIGGRAPH85 paper, by applying SLeRP recursively.

- Not differentiable
- only C^1 continuous (1:st derivative)

Rotation interpolation

Kim, Kim, Shin SIGGRAPH'95 introduced a **closed form** expression for SO(3) interpolation with continuous higher order derivatives

- Using cumulative B-splines
- Using logarithms of relative rotations

Cumulative B-splines

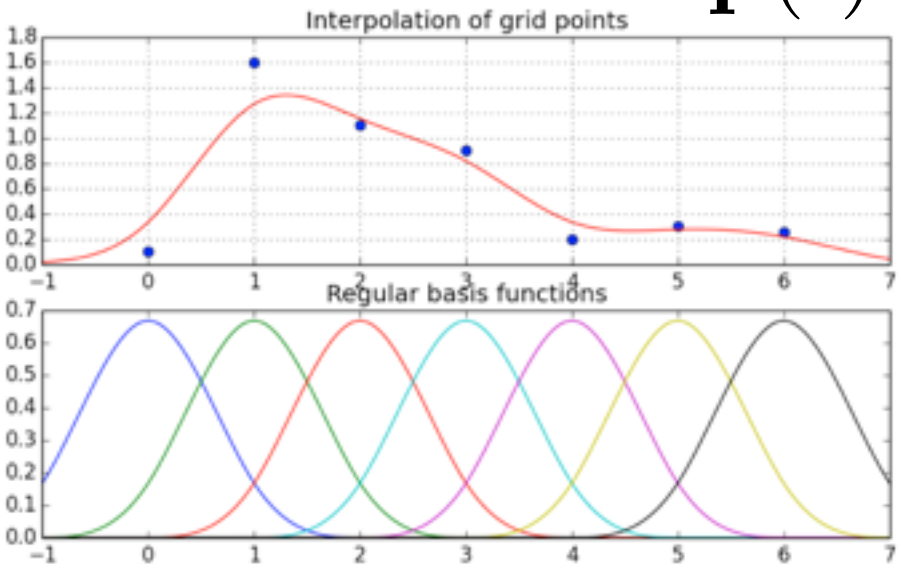
A regular B-spline curve can be written:

$$\mathbf{p}(t) = \sum_{k=1}^{K} \mathbf{p}_k B_k(t)$$

Cumulative B-splines

A regular B-spline curve can be written:

$$\mathbf{p}(t) = \sum_{k=1}^{K} \mathbf{p}_k B_k(t)$$



Cumulative B-splines

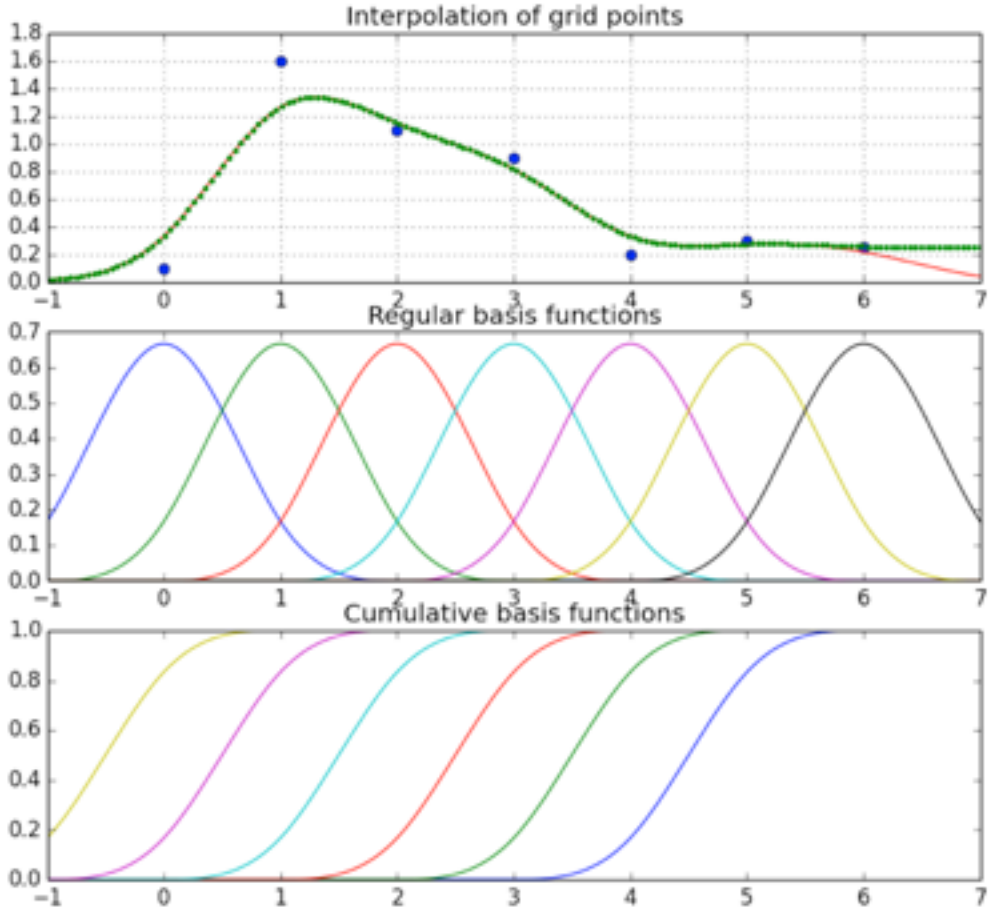
A regular B-spline curve can be written:

$$\mathbf{p}(t) = \sum_{k=1}^{K} \mathbf{p}_k B_k(t)$$

In cumulative form:

$$\mathbf{p}(t) = \mathbf{p}_1 \tilde{B}_1(t) + \sum_{k=2}^{K} (\mathbf{p}_k - \mathbf{p}_{k-1}) \tilde{B}_k(t)$$

Cumulative B-splines



$$\tilde{B}_k(t) = \sum_{l=k}^{K} B(t)$$

Interpolation

That was approximation, how about interpolation?

Interpolation

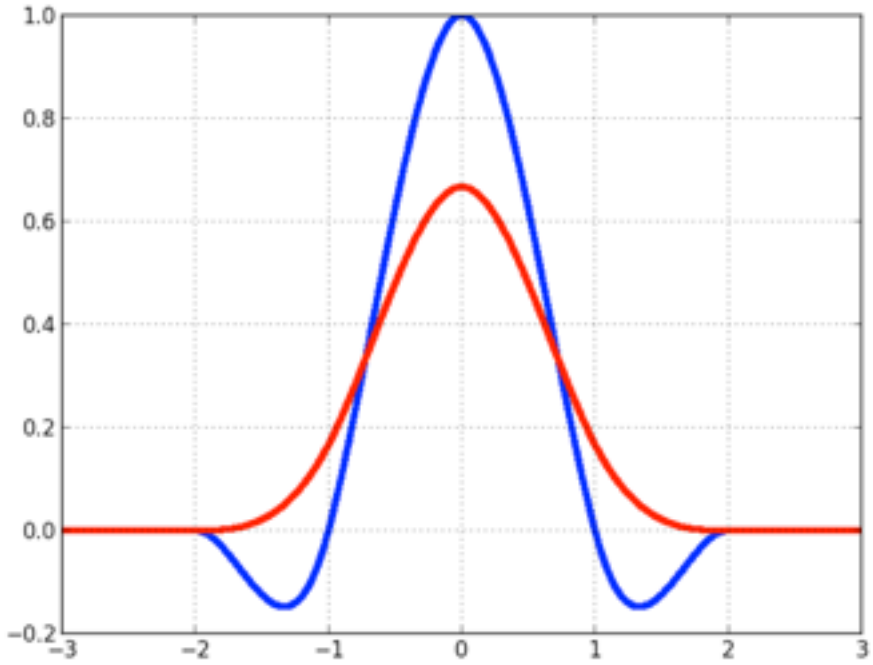
That was approximation, how about interpolation?

1. Solve a linear equation system to find dual basis or ”dual knots”, e.g. [**Unser, SP magazine’99**]

2. Replace the basis functions.

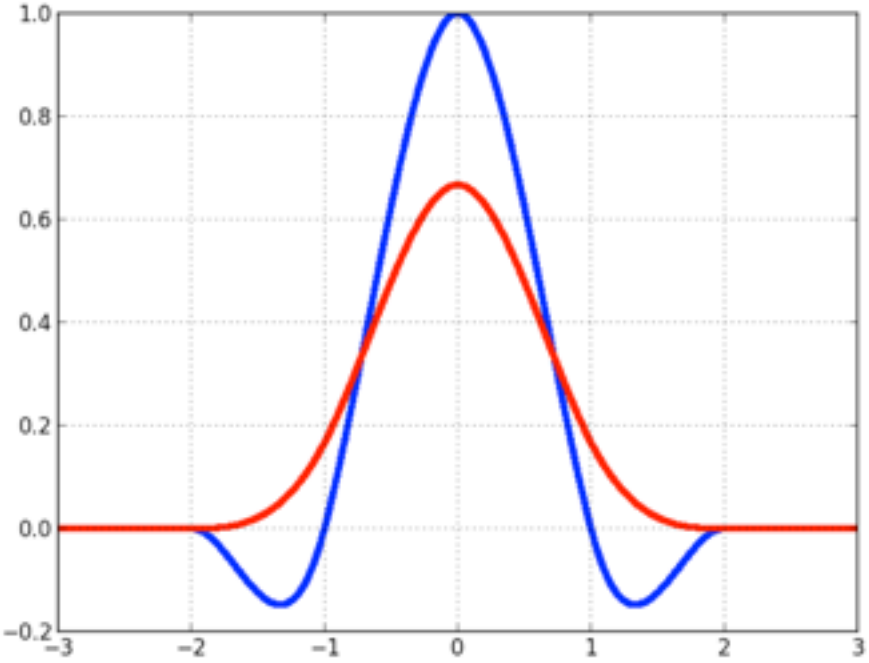
Interpolation

Replace kernel by an interpolating kernel:



Interpolation

Replace kernel by an interpolating kernel:

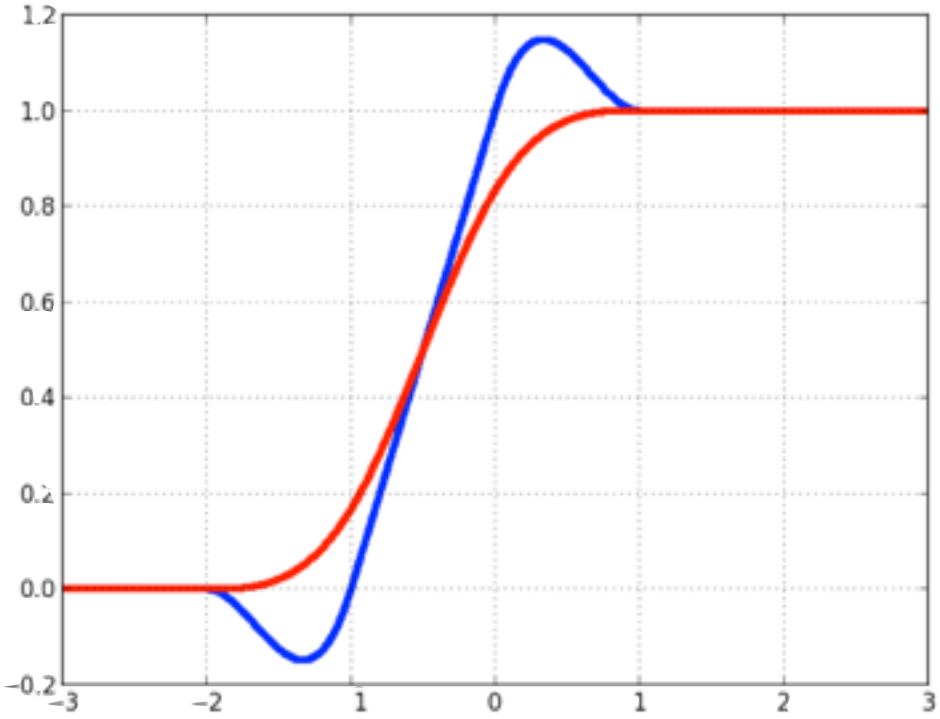


$$B_k(t) = \begin{cases} 1 & \text{if } t = k \\ 0 & \text{if } t \in \mathbb{Z} \setminus k \\ y & y \in \mathbb{R} \text{ otherwise.} \end{cases}$$

$$\sum_k B_k(t) = 1 \quad \forall t \in \Omega$$

Cumulative form for interpolation

The weights are no longer in [0,1]



$$\tilde{B}_k(t) = \sum_{l=k}^{K} B(t)$$

Cumulative form on SO(3)

$$\mathbf{q}(t) = \mathbf{q}_1^{\tilde{B}_1(t)} \prod_{k=2}^{K} \exp(\omega_k \tilde{B}_k(t))$$

where $\omega_k = \log(\mathbf{q}_{k-1}^{*}\mathbf{q}_k)$

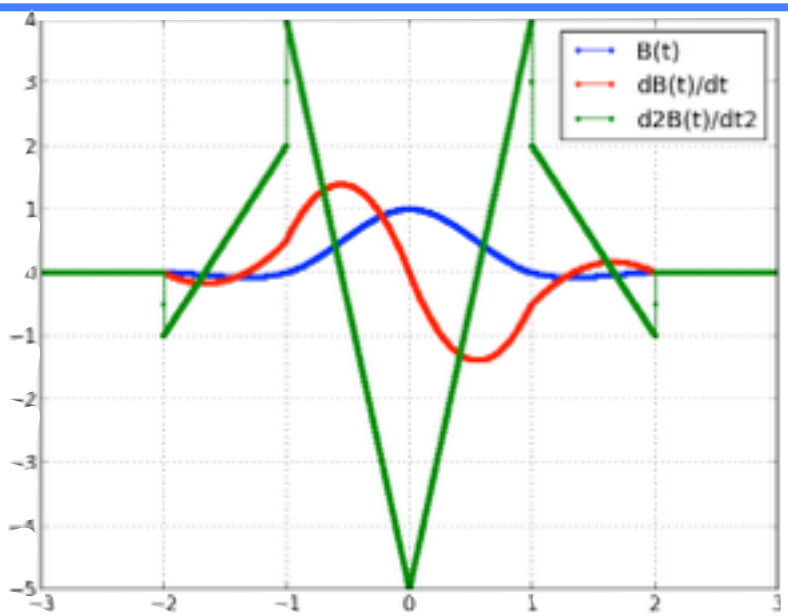
Cumulative form on SO(3)

$$\mathbf{q}(t) = \mathbf{q}_1^{\tilde{B}_1(t)} \prod_{k=2}^{K} \exp(\omega_k \tilde{B}_k(t))$$

where $\omega_k = \log(\mathbf{q}_{k-1}^{*}\mathbf{q}_k)$

- Derivatives are found using the chain rule
  - C^n continuous if B_k is C^n

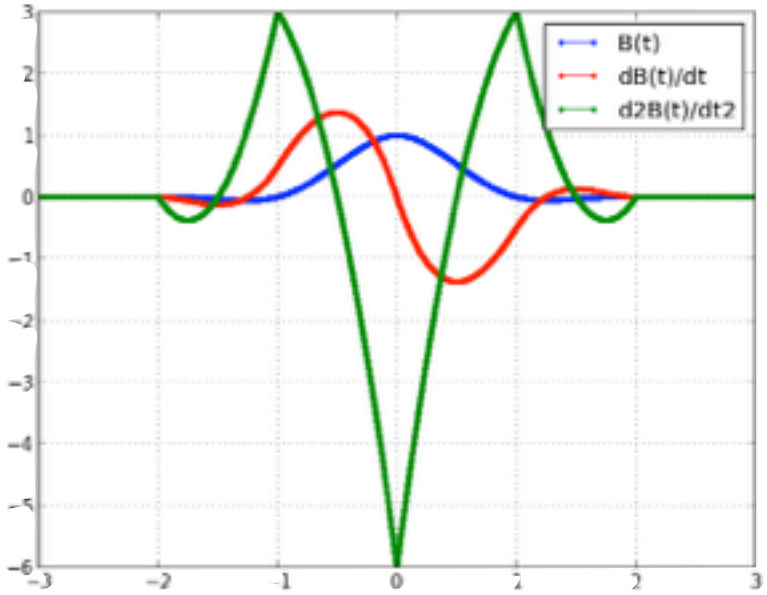
Cubic spline continuity



The standard cubic spline is only C^1

$$h(t) = \begin{cases} 1 - (a+3)t^2 + (a+2)|t|^3 & \text{if } |t| < 1 \\ a(|t|-1)(|t|-2)^2 & \text{if } 1 \leq |t| \leq 2 \\ 0 & \text{otherwise.} \end{cases}$$

Cubic spline continuity



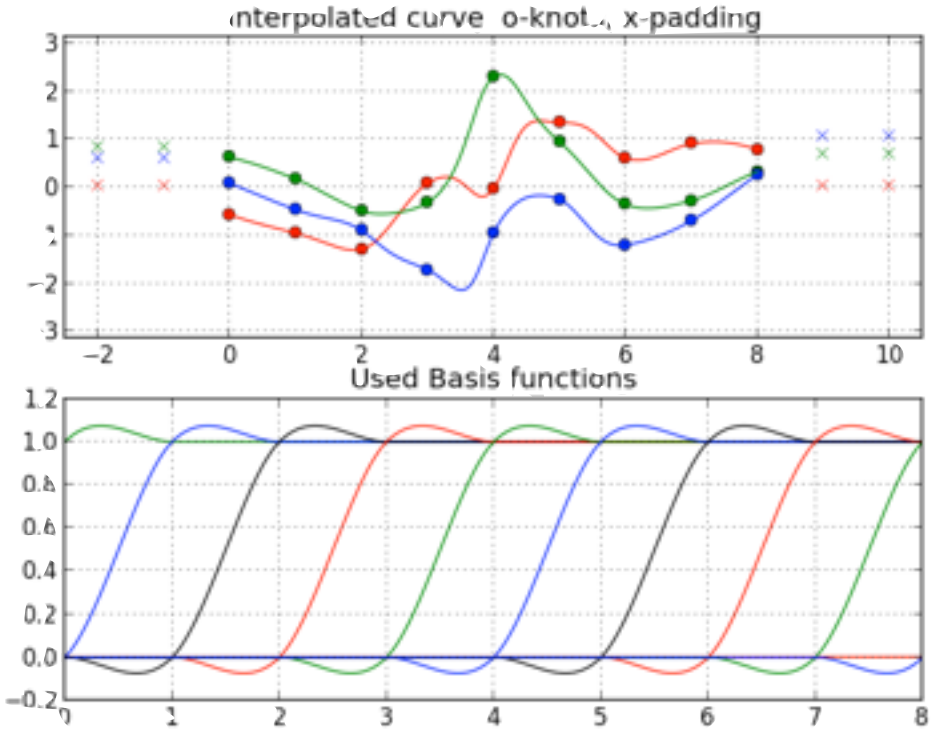
With a quartic spline we can obtain C^2 continuity with the same support

[**Ringaby&Forssén ICCP14**]

Unfortunately the derivatives are still not very smooth, thus a small improvement in practise :-(

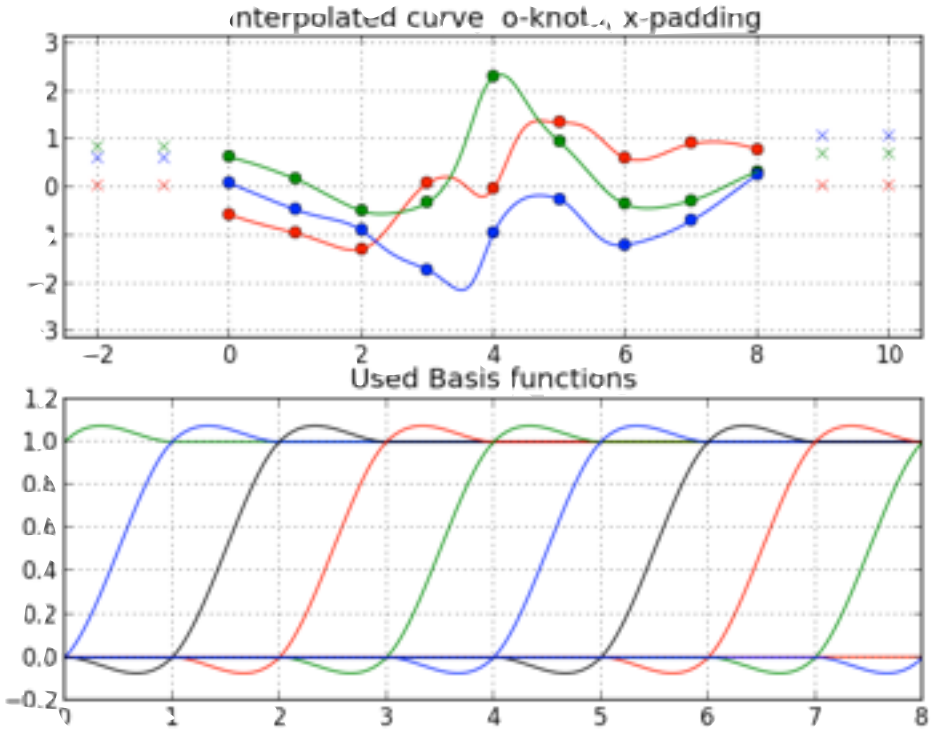
Rotation interpolation

Plotting R(t) in the log space

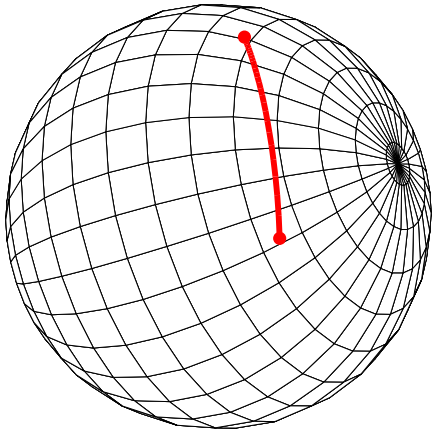


Rotation interpolation

Plotting R(t) in the log space



In SO(3)



$$\text{IM}(\log(\mathbf{q})) = \alpha\hat{\mathbf{n}}$$

Full rigid body motion

Both **R**(t) and **p**(t) should be interpolated.

From physics we know this:

A rigid body will continue to move according to its initial velocity and angular velocity, until affected by external forces.

Full rigid body motion

Both **R**(t) and **p**(t) should be interpolated.

In Computer Graphics Imaging (CGI) this is commonly done using:

$$\mathbf{R}_{\text{int}}(\mathbf{R}_1, \mathbf{R}_2, \lambda) = \mathbf{R}_1 \exp(\lambda \log(\mathbf{R}_1^T \mathbf{R}_2))$$

and

$$\mathbf{p}_{\text{int}}(\mathbf{p}_1, \mathbf{p}_2, \lambda) = \mathbf{p}_1 + \lambda(\mathbf{p}_2 - \mathbf{p}_1), \quad \lambda \in [0, 1]$$

Full rigid body motion

Both **R**(t) and **p**(t) should be interpolated.

The joint of **R** and **t** defines the special Euclidean group SE(3). An element **T** in SE(3) is an action on a 3D point **p**:

$$\begin{bmatrix} \mathbf{p}_2 \\ 1 \end{bmatrix} = \mathbf{T} \begin{bmatrix} \mathbf{p}_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ 1 \end{bmatrix}$$

Full rigid body motion

The recent paper:

S. Lovegrove, A. Patron-Perez, G. Sibely, *Spline Fusion: A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras*, **BMVC2013**

Proposes a SLeRP-like construction on SE(3):

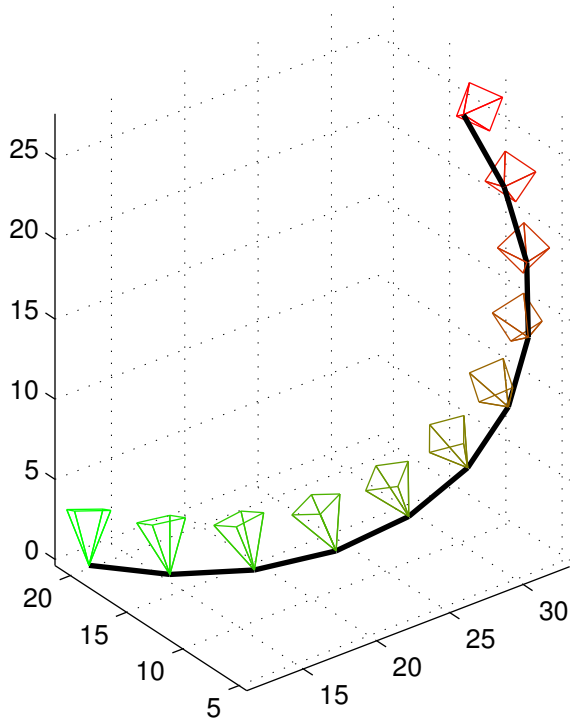
$$\mathbf{T}(\mathbf{T}_1, \mathbf{T}_2, \lambda) = \mathbf{T}_1 \exp(\lambda \log(\mathbf{T}_1^{-1}\mathbf{T}_2))$$

Used together with Shin,Shin,Kim style splines.

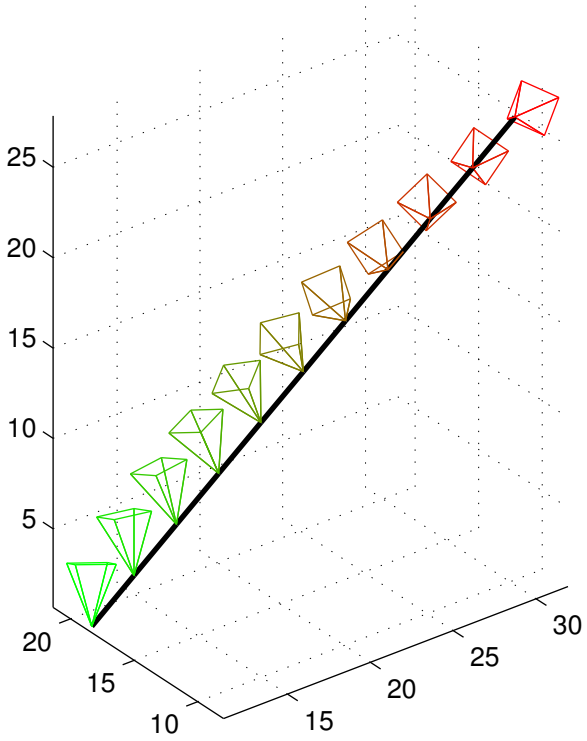
Full rigid body motion

Interesting idea, but here is what happens:

Trajectory from SE(3)



Trajectory from R^3 and SO(3)



Full rigid body motion

$$\mathbf{T}(\mathbf{T}_1, \mathbf{T}_2, \lambda) = \mathbf{T}_1 \exp(\lambda \log(\mathbf{T}_1^{-1}\mathbf{T}_2))$$

Expansion of the SE(3) tangent reveals why:

$$\log(\mathbf{T}_1^{-1}\mathbf{T}_2) = \log \begin{bmatrix} \mathbf{R}_1^T\mathbf{R}_2 & \mathbf{R}_1^T(\mathbf{t}_2 - \mathbf{t}_1) \\ \mathbf{0}^T & 1 \end{bmatrix}$$

Full rigid body motion

$$\mathbf{T}(\mathbf{T}_1, \mathbf{T}_2, \lambda) = \mathbf{T}_1 \exp(\lambda \log(\mathbf{T}_1^{-1}\mathbf{T}_2))$$

Expansion of the SE(3) tangent reveals why:

$$\log(\mathbf{T}_1^{-1}\mathbf{T}_2) = \log \begin{bmatrix} \mathbf{R}_1^T\mathbf{R}_2 & \mathbf{R}_1^T(\mathbf{t}_2 - \mathbf{t}_1) \\ \mathbf{0}^T & 1 \end{bmatrix}$$

Correct expression, used in e.g. CGI:

$$\mathbf{T}(\mathbf{T}_1, \mathbf{T}_2, \lambda) = \begin{bmatrix} \mathbf{R}_1 \exp(\lambda \log(\mathbf{R}_1^T\mathbf{R}_2)) & \mathbf{t}_1 + \lambda(\mathbf{t}_2 - \mathbf{t}_1) \\ \mathbf{0}^T & 1 \end{bmatrix}$$

Full rigid body motion

Compared to SE(3) interpolation, separate interpolation of **R**(t) and **p**(t) has the following advantages:

- Knot density may be set differently on **R**(t) and **p**(t). Used e.g. in [**Ringaby&Forssén ICCV'11**]
- Newton may rest in his grave.

Papers to discuss next week…

K. Shoemake, *Animating rotation with quaternion curves*, SIGGRAPH'85

and

Kim, Kim, Shin, *A General Construction Scheme for Unit Quaternion Curves with Simple High Order Derivatives*, SIGGRAPH'95