Animation II

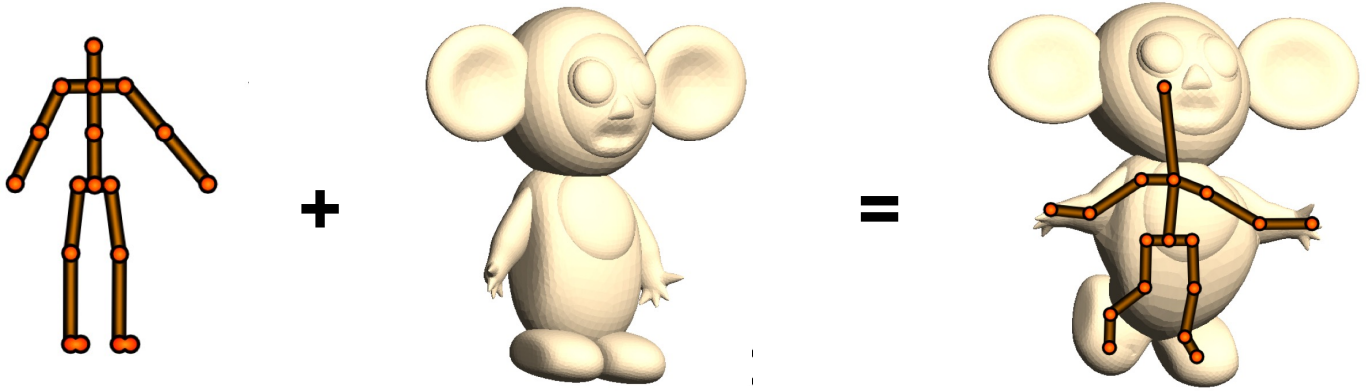
Prof Cengiz Öztireli

UNIVERSITY OF CAMBRIDGE

Character Animation

A rotation and a translation is stored on each bone or joint. We can assume they are stored on bones.
As the user rotates and translates bones via e.g. the mouse, these stored transformations are updated.
As an example, we can store the rotations as 3x3 matrices $\mathbf{R}_i$ and translations as vectors $\mathbf{t}_i$.
Note: matrices are always represented with boldface upper case letters.

- Rigging
  - Attaching a skeleton to a model
  - Skeleton is key-framed to animate the model

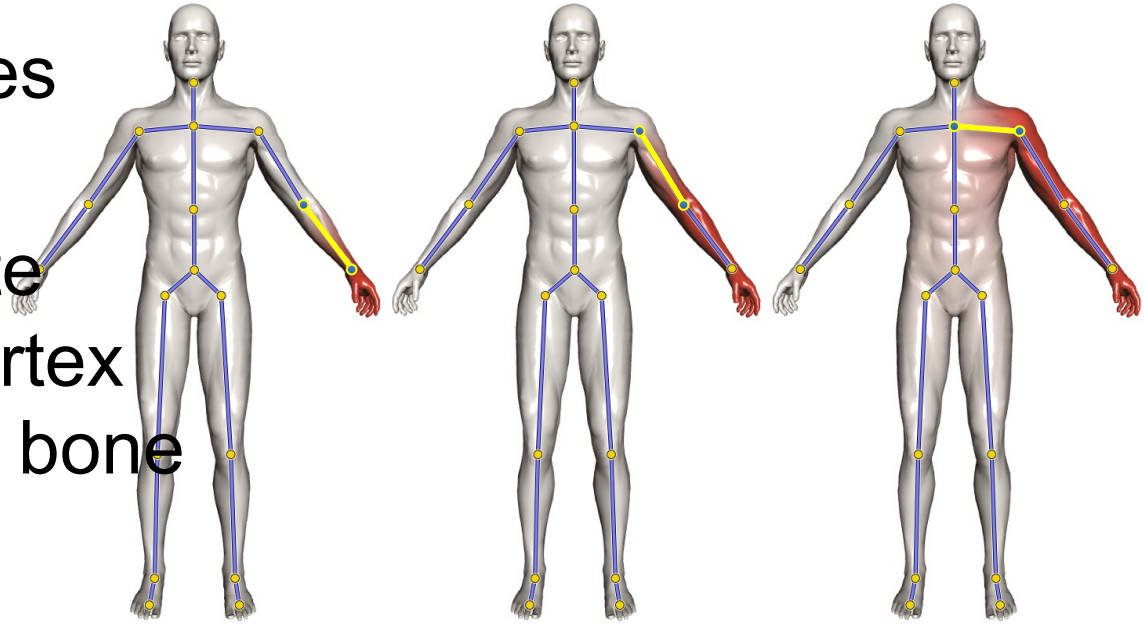


The second step of rigging is attaching each bone to a 3D model.

This means computing a weighting function for each bone that determines how much the transformation on a bone is affecting a given point on the model surface.

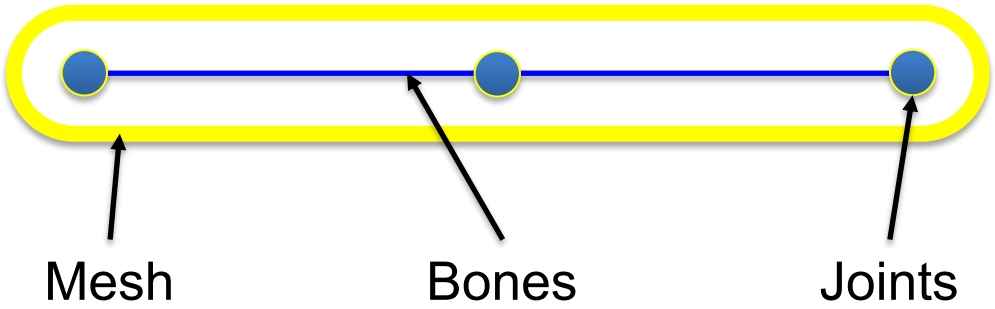
The function is plotted with color coding here with more red indicating higher values.

- Rigging
  - Attach the bones to the model
  - Weights indicate how much a vertex is affected by a bone



To see what is going on better, let's focus on a simple case: we have a simple mesh with two bones/ three joints.

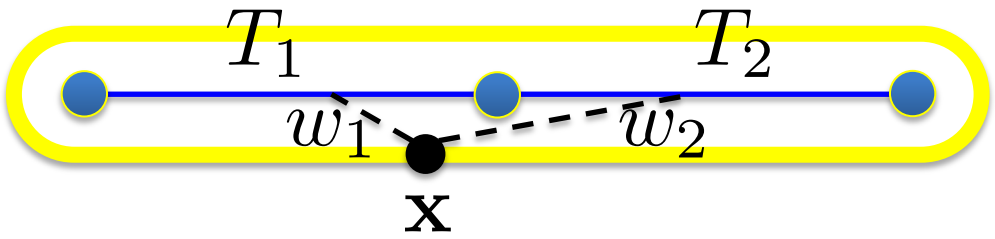
- Rigging
  - Attach the bones to the model



Character Animation

As we change the transformation T_i, i.e. rotations and translations, the mesh should deform accordingly.
Let's say we want to compute the new position of a given point $\mathbf{x}$ on the mesh.
We first blend the transformations with the stored weights w_i. This is denoted with the avg function here.
We then apply the transformation to the point $\mathbf{x}$ to get the new corresponding point on the deformed model.

- Rigging
  - Attach the bones to the model



$$T(\mathbf{x}) = \text{avg}(T_1, T_2, w_1, w_2)$$

The question is: how do we represent the transformations and compute the blending of transformations?
One way is representation via 4x4 transformation matrices $\mathbf{T}_i$, and blending via a linear combination of those.
Each matrix stores both the rotation and translation.
The final blended transformation is then applied to the point $\mathbf{x}$ in homogenous coordinates, i.e. $\mathbf{x} = [x,\ y,\ z,\ 1]^{\mathrm{T}}$

- Rigging
  - How to blend (average) transformations

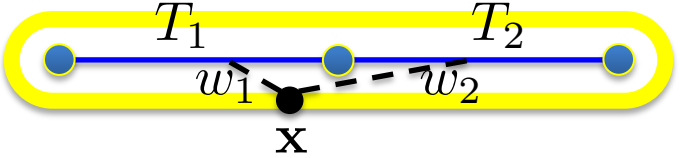
    Linear Blend Skinning

    Represent T_i with $\mathbf{T}_i$

    in homogenous coordinates

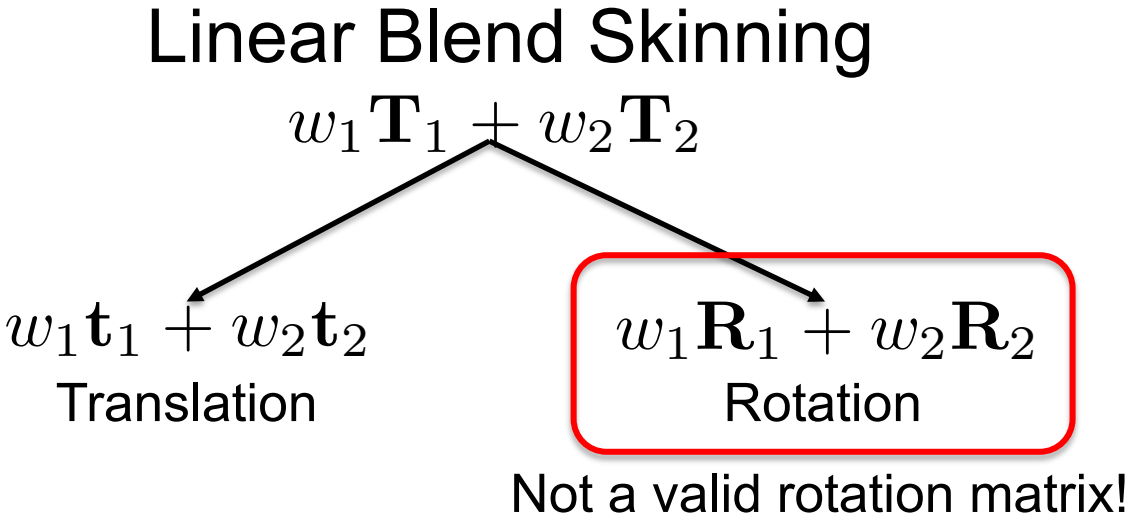
$$\mathbf{T}(\mathbf{x}) = w_1(\mathbf{x})\mathbf{T}_1 + w_2(\mathbf{x})\mathbf{T}_2$$

$$\mathbf{x}' = \mathbf{T}(\mathbf{x})\mathbf{x}$$



$$T(\mathbf{x}) = \mathrm{avg}(T_1, T_2, w_1, w_2)$$

- How to blend (average) transformations



A valid rotation matrix should satisfy two properties: its transpose is equal to its inverse and its determinant is 1. The first property ensures it is not scaling vectors and only rotating. The second avoids introducing flips.
It is easy to see that the first property is not satisfied for linear combinations of rotation matrices.

- How to blend (average) transformations

Valid rotation matrix

$$\mathbf{R}^T = \mathbf{R}^{-1}$$
$$\det(\mathbf{R}) = 1$$

Linear blending

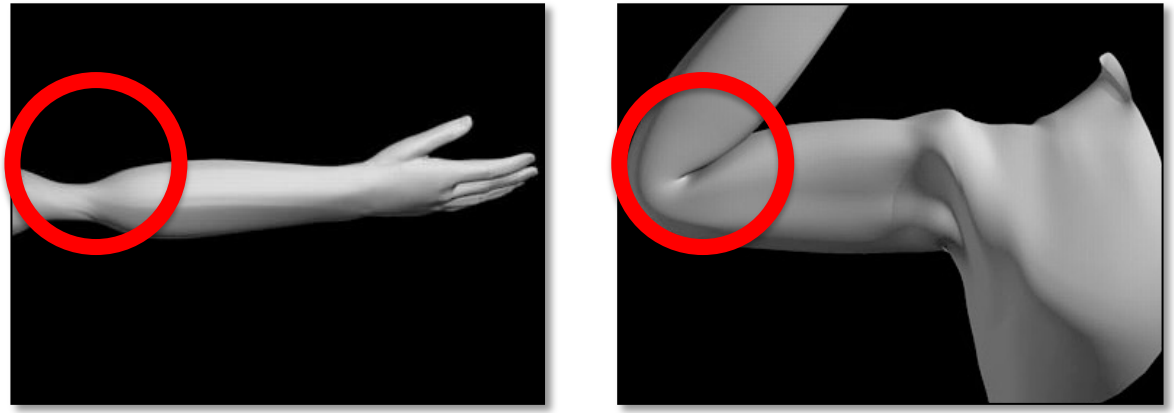
$$(w_1\mathbf{R}_1 + w_2\mathbf{R}_2)^T$$
$$= (w_1\mathbf{R}_1^T + w_2\mathbf{R}_2^T)$$
$$\neq (w_1\mathbf{R} + w_2\mathbf{R})^{-1}$$

Blended Rigid Transformations

In practice, invalid rotation matrices lead to volume loss, especially around the joints where the blending is strongest. Left: candy-wrapper artefact, right: elbow collapse artefact.

- How to blend (average) transformations

Linear Blend Skinning: problems



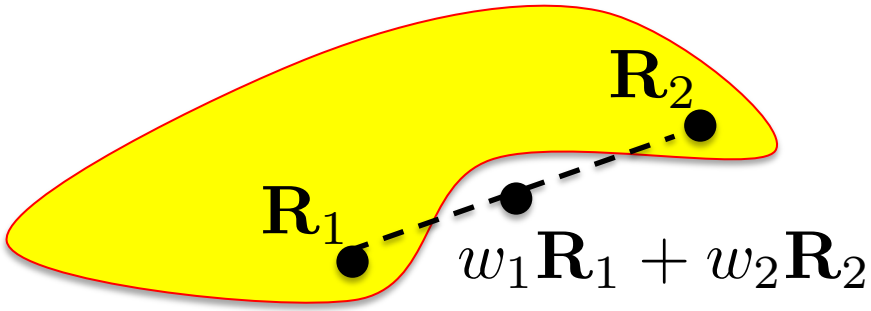
Blended Rigid Transformations

We can visualize what is going wrong by considering the subspace (it is a manifold) of rotation matrices in the 9 dimensional space of all 3x3 matrices.

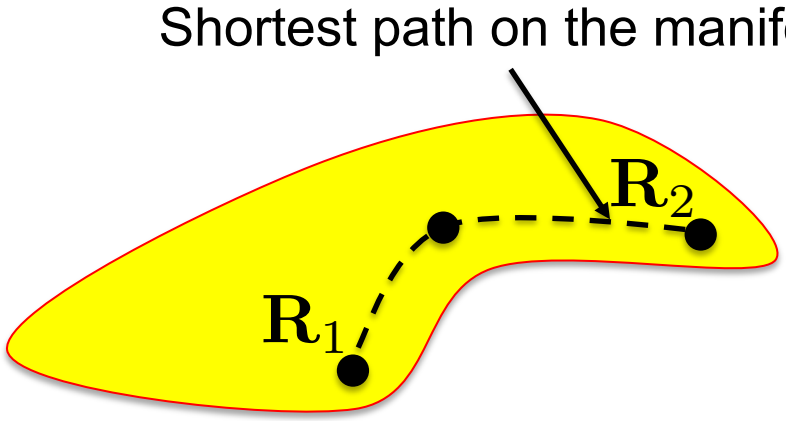
A linear combination of rotation matrices will not necessarily lie on this manifold.

Instead, what we want is the shortest path that respects the manifold of rotation matrices.

- How to blend transformations



Manifold of rigid transformations



Manifold of rigid transformations

This manifold is called SO(3) and is given by the two conditions on valid rotation matrices we have seen. Similarly, we can define SE(3), the manifold of valid rigid transformations.

- Manifold of rotations – SO (3)

Valid rotation matrix

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

$$\det(\mathbf{R}) = 1$$

- Manifold of rigid transformations – SE (3)

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

$$\det(\mathbf{R}) = 1$$

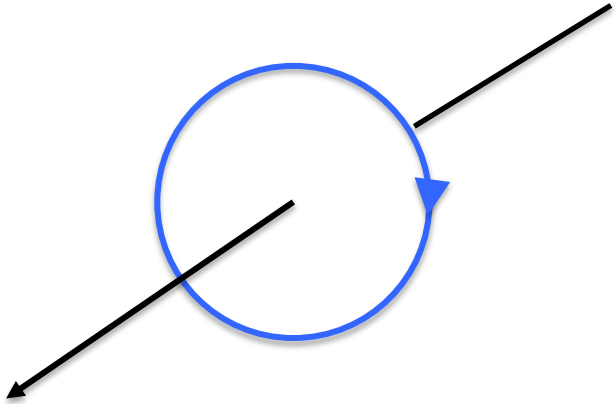
$$\mathbf{T} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0\ 0\ 0 & 1 \end{pmatrix}$$

In summary, it is hard to generate valid rigid transformations with linear combinations if we work with the matrix representation of transformations.
Instead, we will utilize an alternative representation: dual quaternions.

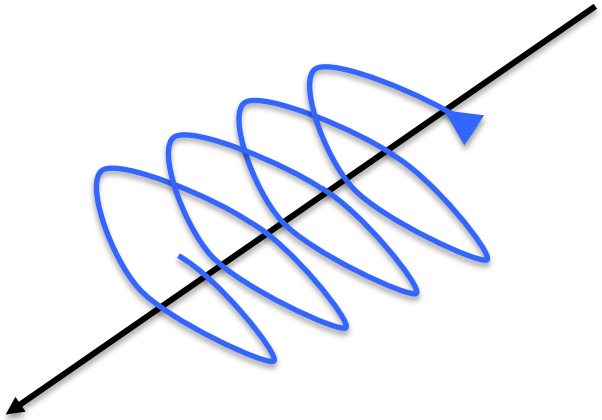
- Matrices not convenient for blending
- Alternative representation: dual quaternions

We will first learn about quaternions that represent rotations as a rotation of an angle around an axis.
We will then see how these generalize to rotations and translations with dual quaternions.
Intuitively, with dual quaternions, we have rotation around and translation along an axis.

- Representing rigid transformations



Rotations with quaternions



Rigid motions with dual quaternions

In a quaternion, we have the axis in 3D around which to rotate, and a rotation angle.

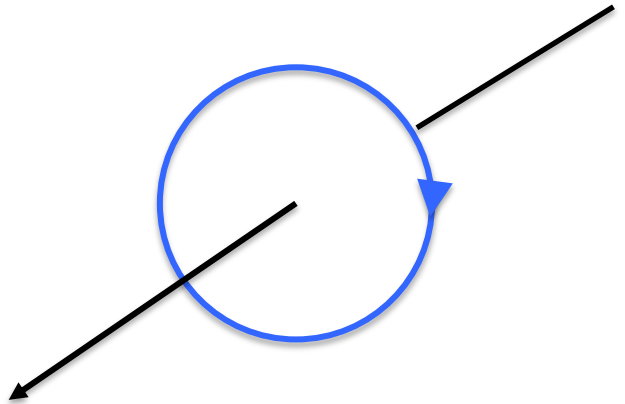
- Representing rotations with quaternions

$$\mathbf{q} = \cos\left(\frac{\theta}{2}\right) + \mathbf{s}\sin\left(\frac{\theta}{2}\right)$$

Quaternion

Rotation angle

Rotation axis



Rotations with quaternions

The rotation axis **s** is represented with an extension of imaginary numbers, denoted with i, j, k.
Rotation axis is a unit vector and hence the sum of squares of its components s_i, s_j, s_k equals 1.
The algebra of these numbers can be defined with $i^2 = j^2 = k^2 = ijk = -1$.

- Quaternions

$$\mathbf{q} = \cos\left(\frac{\theta}{2}\right) + \mathbf{s}\sin\left(\frac{\theta}{2}\right)$$

$$\mathbf{s} = s_i i + s_j j + s_k k$$

$$s_i^2 + s_j^2 + s_k^2 = 1$$

$$i^2 = j^2 = k^2 = ijk = -1$$

Rotations

There are a few important operations on quaternions that we will utilize.

The conjugate has the same meaning as for imaginary numbers, i.e. the non-real part, which is $\mathbf{s}$, is negated.

Note that this can either be written as having $-\mathbf{s}$, or the same $\mathbf{s}$ but $-\theta$.

Both mean we invert the rotation (recall how quaternions represent rotations). Inverse is thus equal to conjugate.

- # Operations on quaternions

$$\mathbf{q} = \cos\left(\frac{\theta}{2}\right) + \mathbf{s}\sin\left(\frac{\theta}{2}\right)$$

Conjugate

$$\mathbf{q}^* = \cos\left(\frac{\theta}{2}\right) - \mathbf{s}\sin\left(\frac{\theta}{2}\right) = \cos\left(-\frac{\theta}{2}\right) + \mathbf{s}\sin\left(-\frac{\theta}{2}\right)$$

Inverse (for unit quaternions) $\mathbf{q}^{-1} = \mathbf{q}^*$

Rotations

Composing rotations is done by multiplying quaternions.

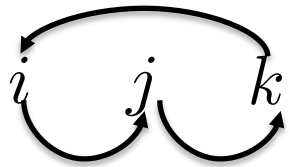
Multiplication is carried out with the rules: $ij = k$, $jk = i$, $ki = j$, $ji = -k$, $kj = -i$, $ik = -j$, $i^2 = j^2 = k^2 = -1$.

Multiplying a quaternion with its conjugate gives us the norm of it, which is **1 for the case of quaternions representing rotations**. This is easy to derive, please do.

- # Operations on quaternions

Multiplication

$$\mathbf{q}_1\mathbf{q}_2 = (a_1 + b_1 i + c_1 j + d_1 k)(a_2 + b_2 i + c_2 j + d_2 k)$$



Norm

$$||\mathbf{q}||^2 = \mathbf{q}\mathbf{q}^* = \cos^2\left(\frac{\theta}{2}\right) + ||s||^2 \sin^2\left(\frac{\theta}{2}\right) = 1$$

The power of a quaternion is defined with the definitions of the exponential and log.
For the case of quaternions representing rotations, q^t is the same as rotating with an angle $t\theta$.

- Operations on quaternions

$$\mathbf{q} = \cos\left(\frac{\theta}{2}\right) + \mathbf{s}\sin\left(\frac{\theta}{2}\right)$$

Power

$$\mathbf{q}^t = e^{t\log\mathbf{q}}$$

$$\log\mathbf{q} = \frac{\theta}{2}\mathbf{s}$$

for quaternions with zero scalar part:

$$e^{\mathbf{q}} = \cos||\mathbf{q}|| + \frac{\mathbf{q}}{||\mathbf{q}||}\sin||\mathbf{q}||$$

To apply a quaternion to a vector to rotate it, we first need to write the vector in the i, j, k space. The **v'** is then the rotated version of **v**.

- Operations on quaternions

$$\mathbf{q} = \cos\left(\frac{\theta}{2}\right) + \mathbf{s}\sin\left(\frac{\theta}{2}\right)$$

Applying to location vectors

$$\mathbf{v} = v_i i + v_j j + v_k k$$

$$\mathbf{v}' = \mathbf{q}\mathbf{v}\mathbf{q}^*$$

For the special case that the axes of rotation are the same for both quaternions, their blending, i.e interpolation, is the interpolation of the angle of rotation.

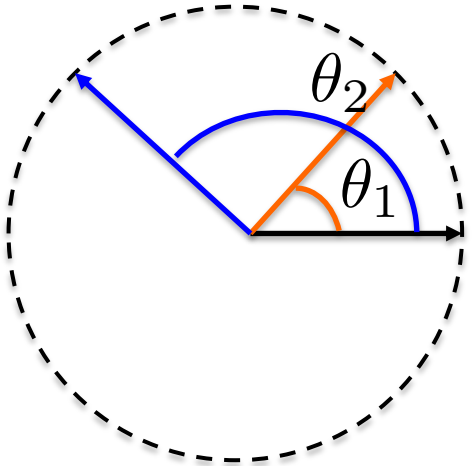
- Blending quaternions

$$\mathbf{s} = \mathbf{s}_1 = \mathbf{s}_2$$

$$\text{interpolate}(\mathbf{q}_1, \mathbf{q}_2, t)$$

$$\theta(t) = (1-t)\theta_1 + t\theta_2$$

$$\mathbf{q}(t) = \cos\left(\frac{\theta(t)}{2}\right) + \mathbf{s}\sin\left(\frac{\theta(t)}{2}\right)$$



For the general case, we use the so-called spherical blending for two quaternions.
For $t = 0$, we get $\mathbf{q}_1$, and for $t = 1$, we get $\mathbf{q}_2$, as expected.

- Blending quaternions
  - In general, $\mathbf{s}_1 \neq \mathbf{s}_2$
  - Spherical blending

$$(\mathbf{q}_2\mathbf{q}_1^*)^t\mathbf{q}_1$$

  - More than two rotations?

The case of more than two rotations is more challenging and we do not have a closed form solution.
A good approximation is a weighted sum of quaternions.
This equation is the reason why we care about quaternions. In contrast to matrix based representations, quaternion linear blending does not lead to volume loss and other artifacts!

- Blending quaternions

$$\mathbf{q}_1 \cdots \mathbf{q}_n \quad w_1 \cdots w_n$$

  - Good approximation:

$$\mathbf{b} = \sum_{i=1}^{n} w_i \mathbf{q}_i$$

Rigid Transformations

These ideas can easily be extended to the case of rigid transformations.
Instead of ordinary real numbers, we use dual numbers in this case in the definition of the quaternions.
Dual numbers have the real and dual part. The dual number ϵ has the property that its square is zero.
This leads to cancellation of higher order terms in multiplications.

- Rotation & translation
- Dual numbers

$$\hat{x} = x_0 + \epsilon x_\epsilon \quad \epsilon^2 = 0$$

E.g. multiplication

$$(a_0 + \epsilon a_\epsilon)(b_0 + \epsilon b_\epsilon)$$
$$= a_0 b_0 + \epsilon(a_0 b_\epsilon + a_\epsilon b_0)$$

In this case, the angle and each component of the axis $\mathbf{s}$ become dual numbers.
All operations and expressions we are interested in stay the same.
In particular, we still have the linear combination as a good approximation of proper shortest path blending.

- Dual quaternions
  - Replace numbers in quaternions with dual numbers

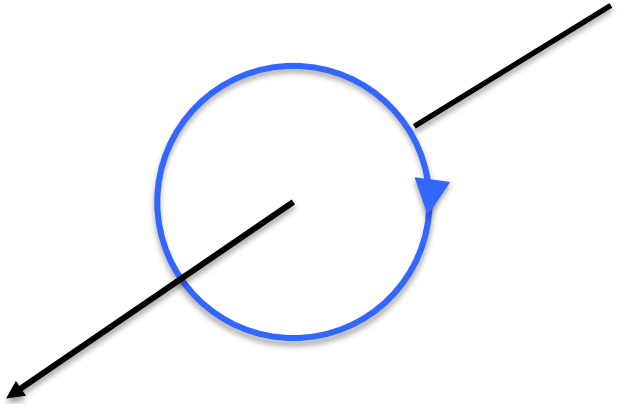
$$\hat{\mathbf{q}} = \cos\left(\frac{\hat{\theta}}{2}\right) + \hat{\mathbf{s}}\sin\left(\frac{\hat{\theta}}{2}\right)$$

  - Almost all operations & notations are the same
  - In particular: $\hat{\mathbf{b}} = \sum_{i=1}^{n} w_i \hat{\mathbf{q}}_i$

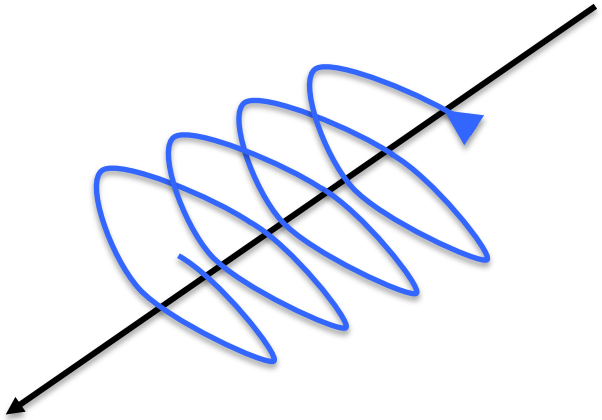
In practice, this means we represent a rotation and translation with a rotation around the axis **s** and a translation along the same axis.

We could prove that this covers all rigid transformations.

- Representing rigid transformations



Quaternions : 4 numbers



Dual quaternions : 8 numbers

Blended Rigid Transformations

There is one thing we ignored so far: recall that only normalized quaternions/ dual quaternions represent valid rotations/ rigid transformations.

With normalization, we always get dual quaternions representing valid rigid transformations.

- Properties

$$\hat{\mathbf{b}} = \sum_{i=1}^{n} w_i \hat{\mathbf{q}}_i$$

1. Generates valid transformations
   - Only if normalized!

$$\hat{\mathbf{b}} = \frac{\sum_{i=1}^{n} w_i \hat{\mathbf{q}}_i}{||\sum_{i=1}^{n} w_i \hat{\mathbf{q}}_i||}$$

Blended Rigid Transformations

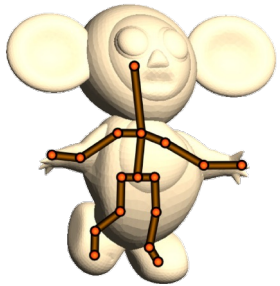
An important property of the approximate blending is that it is invariant to coordinate change: if we first transform all dual quaternions with the same transformation and then blend the resulting quaternions, we get the same result as first blending them and then applying the transformation.

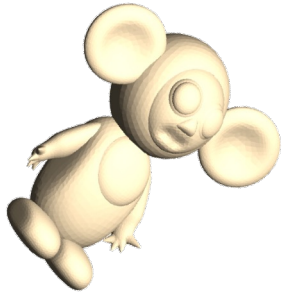
This follows from: $\sum w_i \hat{\mathbf{q}} \hat{\mathbf{q}}_i = \hat{\mathbf{q}} \sum w_i \hat{\mathbf{q}}_i$

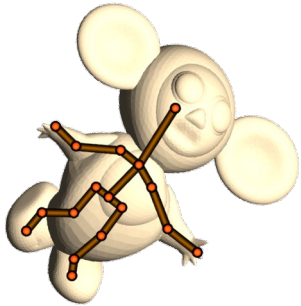
- Properties

$$\hat{\mathbf{b}} = \frac{\sum_{i=1}^{n} w_i \hat{\mathbf{q}}_i}{||\sum_{i=1}^{n} w_i \hat{\mathbf{q}}_i||}$$

2. Coordinate invariance







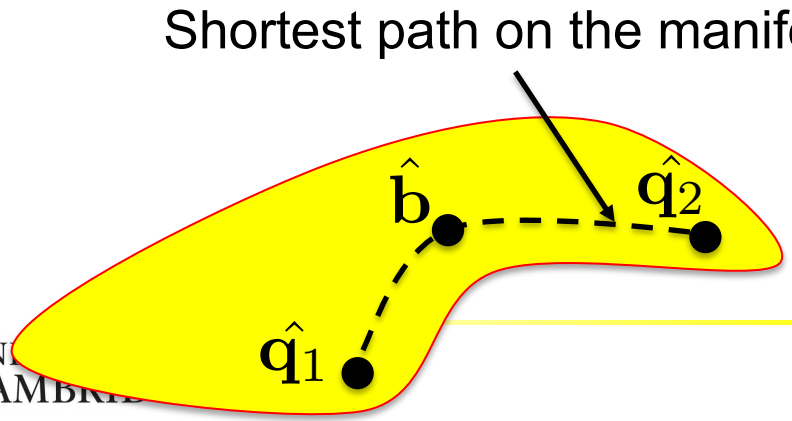
Blended Rigid Transformations

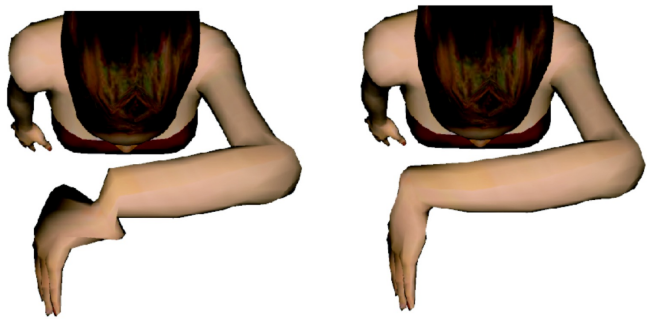
This simple linear blending stays very close to the shortest path blending on SE (3), the manifold of valid rigid transformations.

- Properties

$$\hat{\mathbf{b}} = \frac{\sum_{i=1}^{n} w_i \hat{\mathbf{q}}_i}{||\sum_{i=1}^{n} w_i \hat{\mathbf{q}}_i||}$$

3. Shortest path on SE (3)

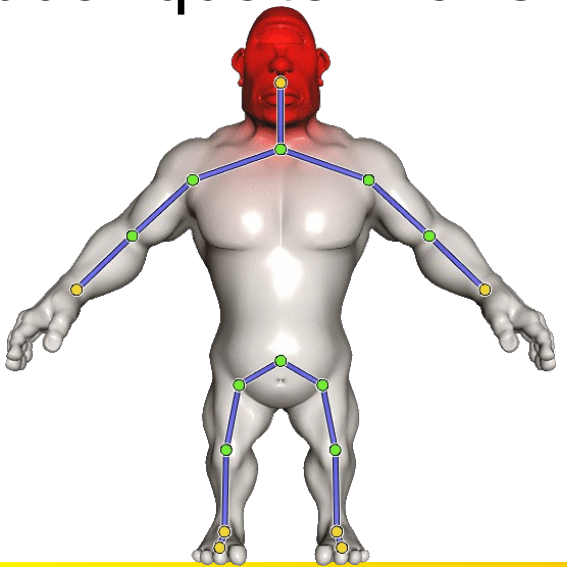




Blended Rigid Transformations

Apart from transformation representation, there are challenges with how we compute the weights for blending. There are several intuitive properties that we briefly summarize.

- Challenges
  - Blending transformations – dual quaternions
  - Weights $w_i(\mathbf{x})$
    - Shape adaptive
    - Intuitive deformations
    - Smooth deformations

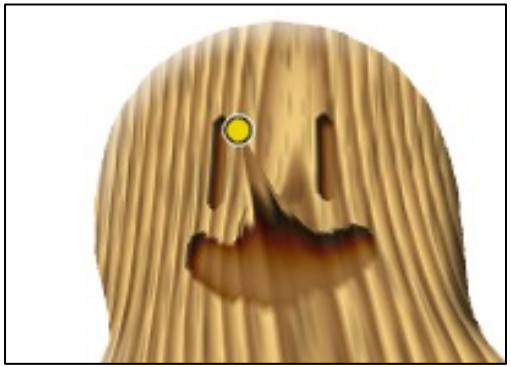


The first property is partition of unity. This is important for volume preservation.
Second property, smoothness of weights, ensures we get smooth deformations without artifacts.

- Weights – desired properties
  - Partition of unity

$$\sum_{i=1}^{n} w_i(\mathbf{x}) = 1$$

  - Smoothness

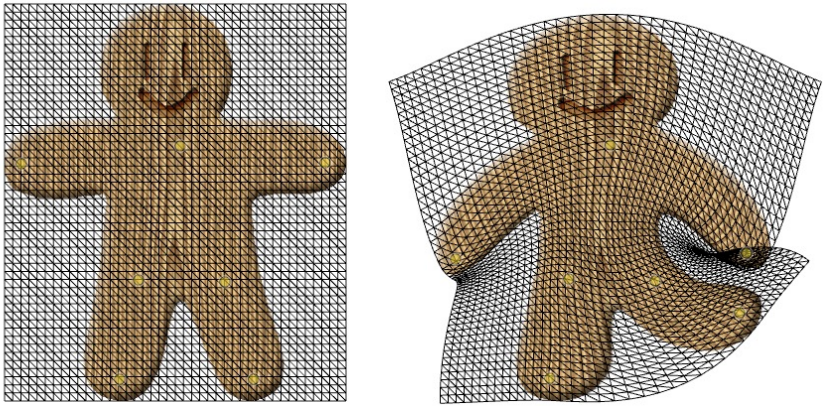


Blended Rigid Transformations

Finally, shape-awareness leads to local influence with respect to the deformed shape and is important for intuitive control.

- Weights – desired properties
  - Shape-awareness

Shape-aware weights



Shape-unaware weights