Computer Animation

Shih-Chin Weng
shihchih.weng@gmail.com

Animation

$$shape = f(time)$$

[TED-Ed: Animation basics: The art of timing and spacing]

12 PRINCIPLES OF ANIMATION

[12 principles of animation by aCreativeAgency]

Animation Principles

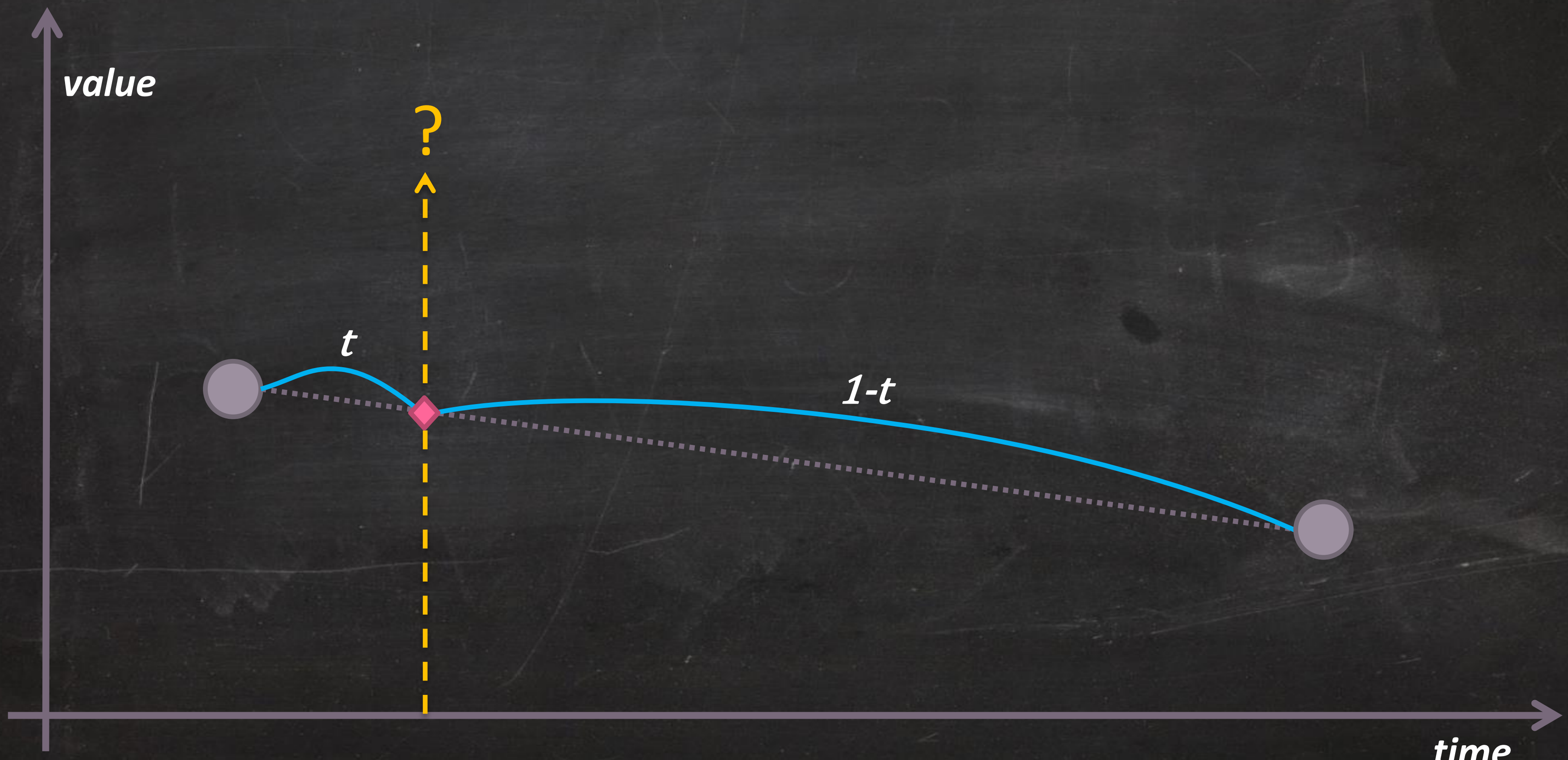
1. *Squash & Stretch*
2. *Anticipation*
3. *Arcs*
4. *Ease In & Ease Out*
5. *Appeal*
6. *Timing*
7. *Solid Drawing*
8. *Exaggeration*
9. *Pose To Pose*
10. *Staging*
11. *Secondary Motion*
12. *Following Through*

https://en.wikipedia.org/wiki/12_basic_principles_of_animation

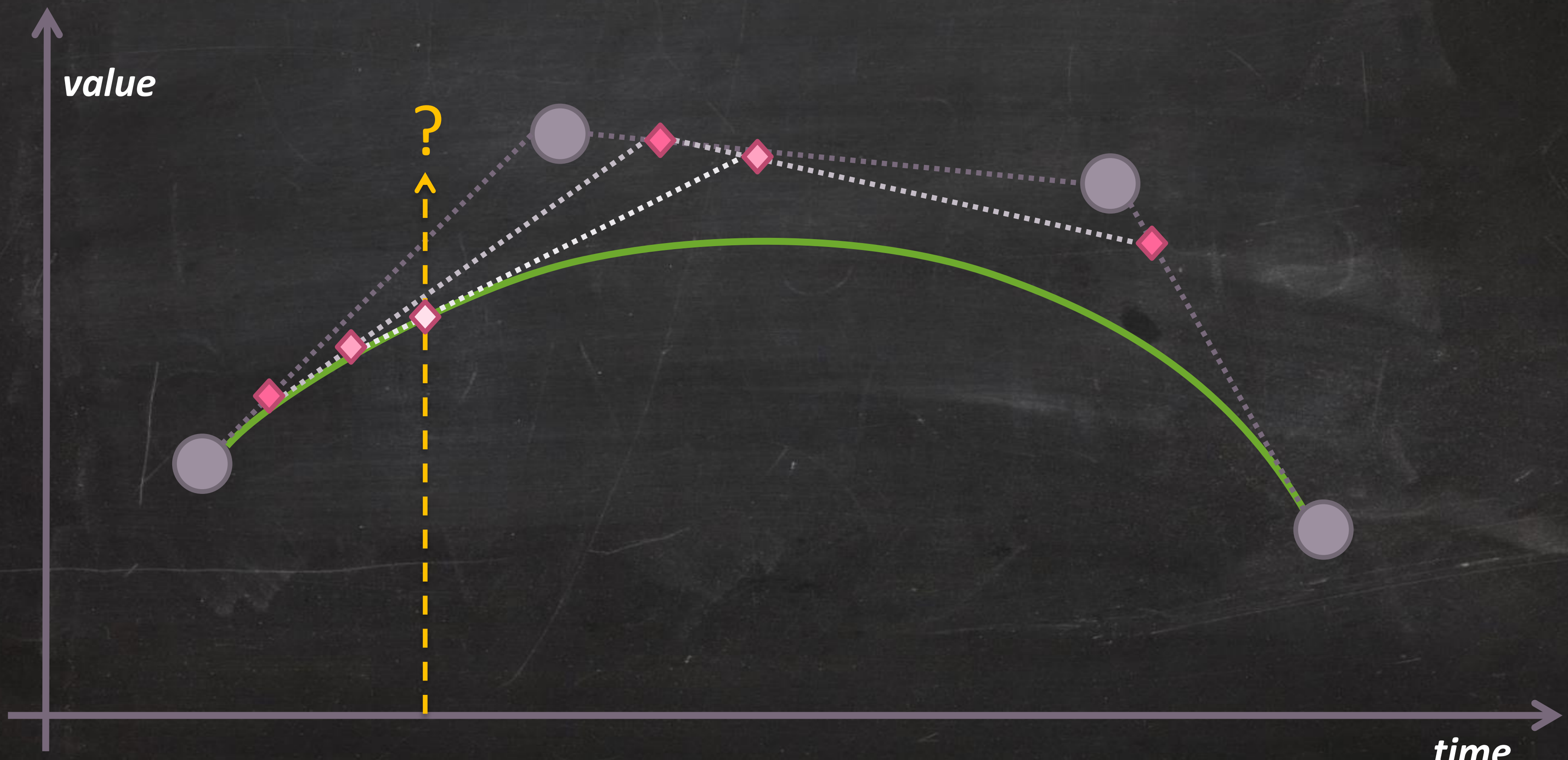
Key-frame Animation

- Animator specifies key-frames, software generate the frames in-between
  - Interpolation is the major operation in
    - time-variant transformations
    - pose-to-pose deformation
- Many animation principles can be modeled from physical law
  - Ex. Squash & stretch, following through, etc.

Data Interpolation



Data Interpolation - Cubic Bezier



Constructing curves using repeated linear interpolation @Pixar In a Box

Interpolation with Parametric Curves

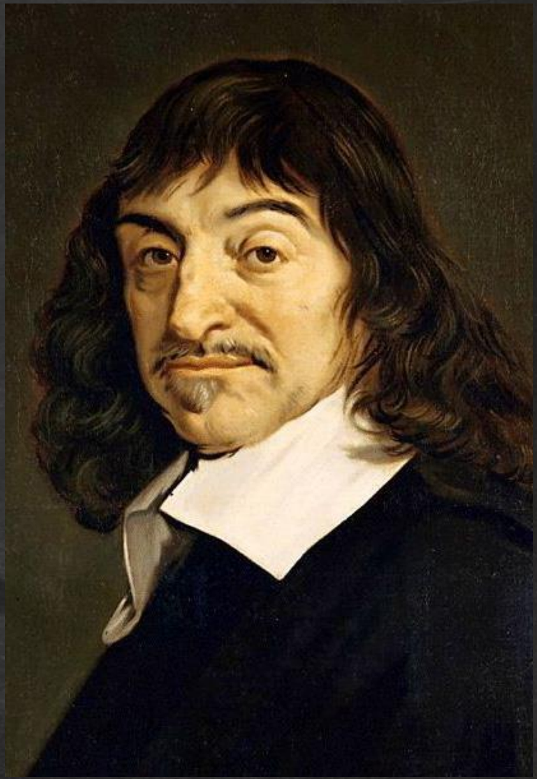
- Cubic Bezier
  - 4 positions
- Catmull-Rom
  - 2 positions, 2 tangents (derived from nearby CVs)
- Hermit Curve
  - 2 (position + tangent)
    - tangents are specified at each CV

Considerations

- Local control
  - Each CV only affects neighboring segments
  - That’s why we need splines
- Smoothness, degree of continuity
  - C^0: matches position
  - C^1: matches tangent
  - C^2: matches curvature

Cartesian Unit Vectors

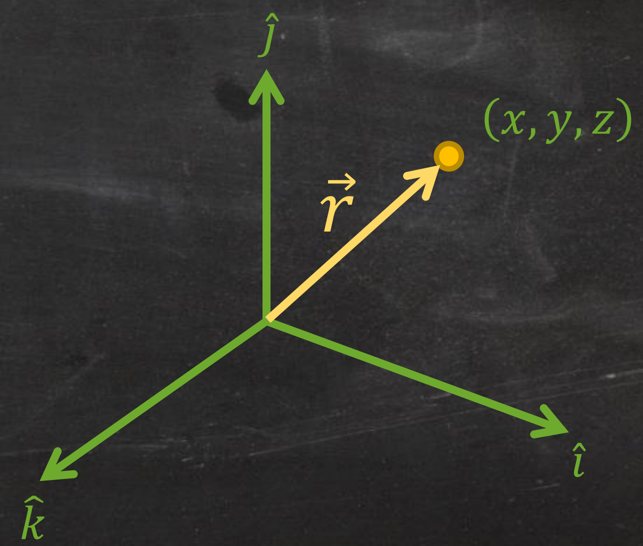
- $\hat{\imath}, \hat{\jmath}, \hat{k}$
  - Coordinate axes
  - Orthonormal
  - Can be drawn at any location, not just at origin
    - Invariant at different locations
- Vector components
  - Projections of the vector onto the coordinate axes



René Descartes (1596-1650)

Change Axes in Cartesian Coordinate

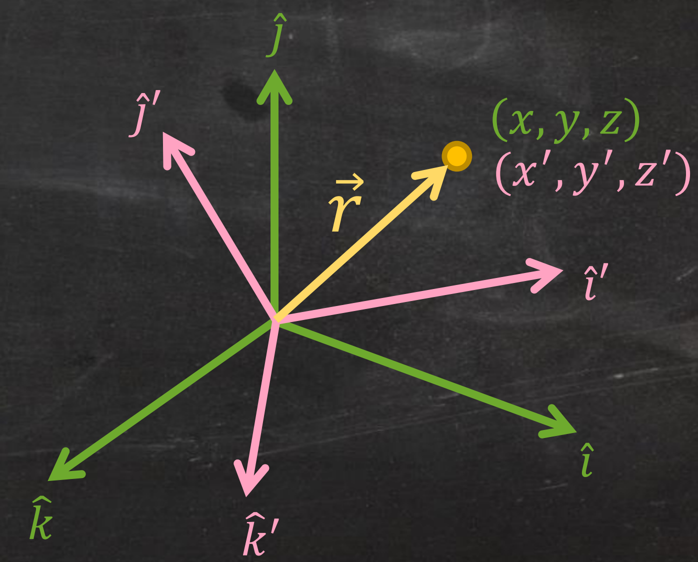
- Geometric information = coordinates + unit basis
  - Coordinates are meaningless without unit basis
- $\vec{r}$ = displacement vector
- $\vec{r} = x\hat{\imath} + y\hat{\jmath} + z\hat{k}$



Change Axes in Cartesian Coordinate

- Geometric information = coordinates + unit basis
  - Coordinates are meaningless without unit basis
- $\vec{r}$ = displacement vector
- $\vec{r} = x\hat{\imath} + y\hat{\jmath} + z\hat{k}$
  $= x'\hat{\imath}' + y'\hat{\jmath}' + z'\hat{k}'$

$\vec{r}$ is fixed!
But its components change!



Two Types of Transformations

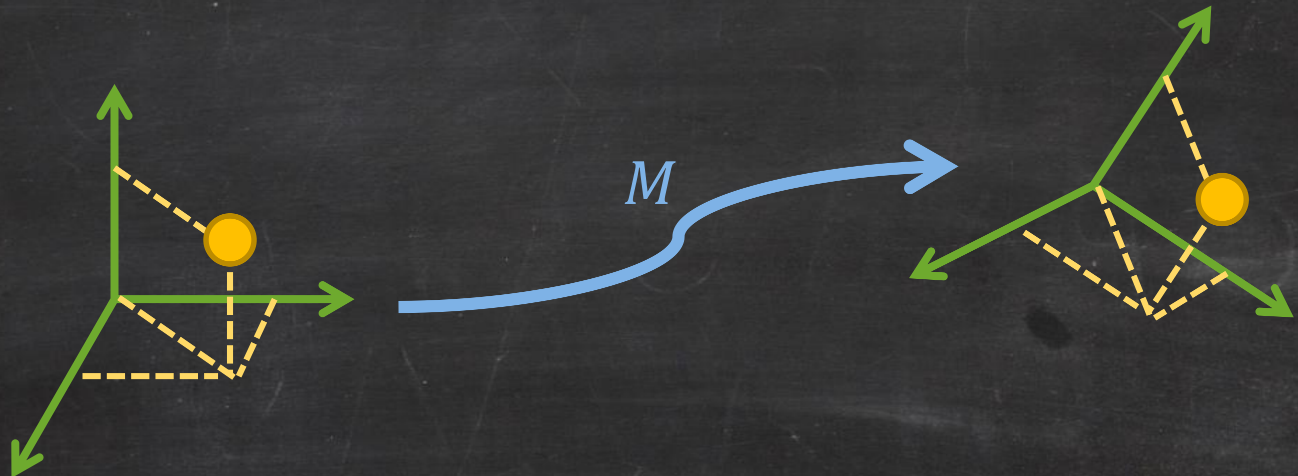
- Coordinate-system transformations
  - Transform basis vector
  - Vector is the same, but components change

World-View-Projection transformation in rendering pipeline

- Transform vector in the same coordinate
  - Vector is different from original one

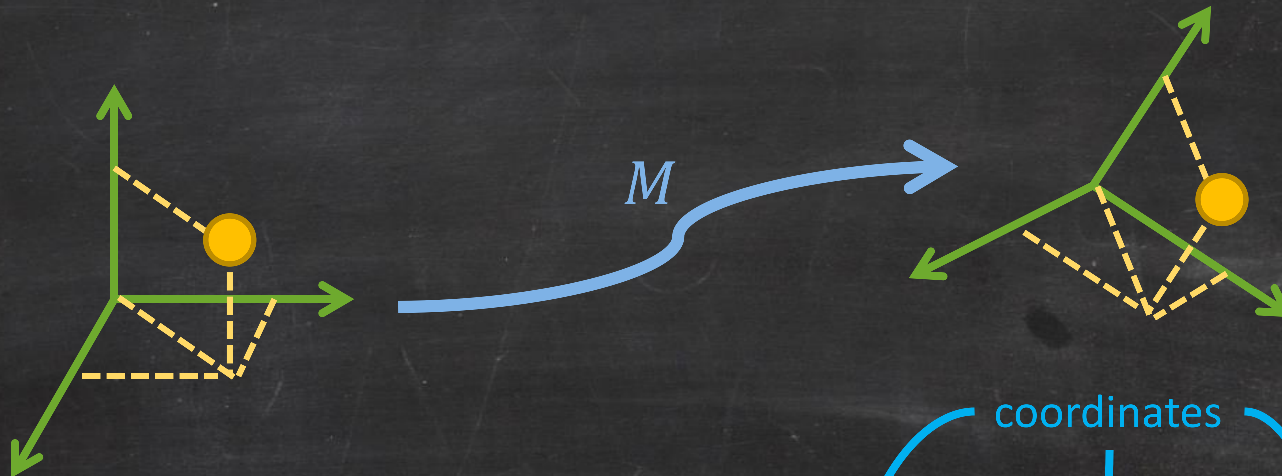
Animation in certain reference frame (ex. world space)

Orientation = Rotation



$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} t_1 & n_1 & b_1 \\ t_2 & n_2 & b_2 \\ t_3 & n_3 & b_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Orientation = Rotation



coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} t_1 & n_1 & b_1 \\ t_2 & n_2 & b_2 \\ t_3 & n_3 & b_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} x + \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix} y + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} z$$

orthogonal matrix: $R^T R = I$

unit basis

Group

- A family of transformations forms a group
- A set G together with a binary operation $\circ$ defined on elements of G is called a group, if it satisfies the axioms of *closure, identity, inverse and associativity*

Group (Cont'd)

Closure

$$g_1, g_2 \in G \rightarrow g_1 \circ g_2 \in G$$

Identity

$$\exists e \in G:\ g \circ e = e \circ g = g$$

Inverse

$$\forall g\ \exists g^{-1} \in G:\ g \circ g^{-1} = g^{-1} \circ g = e$$

Associativity

$$g_1, g_2, g_3 \in G, \qquad g_1 \circ (g_2 \circ g_3) = (g_1 \circ g_2) \circ g_3$$

Two Special Groups in 3D

- SO: Special Orthogonal group
  - $\mathrm{SO}(3) = \{\mathrm{R} \in \mathbb{R}^{3\times 3} : \mathrm{R}\mathrm{R}^{\mathrm{T}} = \mathrm{I}, \det R = +1\}$
    - 3D rotations centered at the origin
- SE: Special Euclidean Group
  - $\mathrm{SE}(3) = \{(p, R) : p \in \mathbb{R}^3, R \in SO(3)\} = \mathbb{R}^3 \times \mathrm{SO}(3)$
    - 3D rotations + translations
    - Rigid motion => preserve distance and orientation

Interpolating Rotation Matrices

$$0.5\begin{bmatrix}0 & 1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix} + 0.5\begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}$$

90°CW around z-axis

90°CCW around z-axis

Interpolating Rotation Matrices

$$0.5\begin{bmatrix}0 & 1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix} + 0.5\begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}$$

90°CW around z-axis

90°CCW around z-axis

?

Oops!! This is **NOT** a rotation matrix!!

Rotation matrix is a group with multiplication **NOT** addition

Representations of Rotations

- Rotation matrix
- Axis-angle
- Euler Angle
- Quaternion
- and many more…

http://rotations.berkeley.edu

After seeing this site, I just realized I didn't know much about rotations at all…

Euler’s Rotation Theorem

- In 3D space, any sequence of rotations about a fixed point is equivalent to a **single** rotation by a given angle θ about a fixed axis

Leonhard Euler (1707-1783)

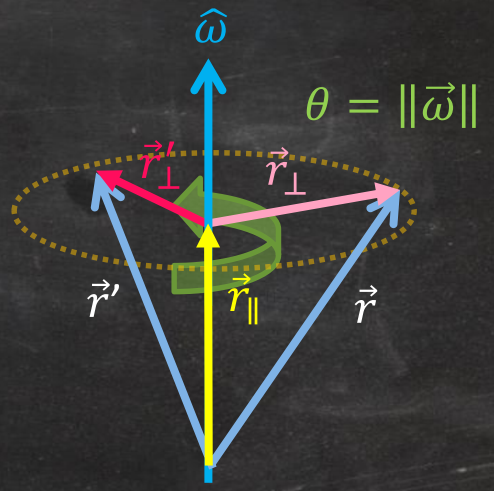
Euler's rotation theorem @Wikipedia

Axis-Angle

- Specify rotation axis $\hat{\omega}$, and rotation angle $\|\vec{\omega}\|$

$$\vec{r}'_{\perp} = \cos\theta\, \vec{r}_{\perp} + \sin\theta\, (\hat{\omega} \times \vec{r})$$

$$\vec{r}_{\perp} = \vec{r} - \vec{r}_{\parallel} = \vec{r} - (\vec{r} \cdot \hat{\omega})\hat{\omega}$$



$$\begin{aligned}\vec{r}' &= \vec{r}'_{\perp} + \vec{r}_{\parallel} \\ &= \cos\theta\, (\vec{r} - (\vec{r} \cdot \hat{\omega})\hat{\omega}) + \sin\theta\, (\hat{\omega} \times \vec{r}) + (\vec{r} \cdot \hat{\omega})\hat{\omega} \\ &= \cos\theta\, \vec{r} + \sin\theta\, (\hat{\omega} \times \vec{r}) + (1 - \cos\theta)\big((\vec{r} \cdot \hat{\omega})\hat{\omega}\big)\end{aligned}$$

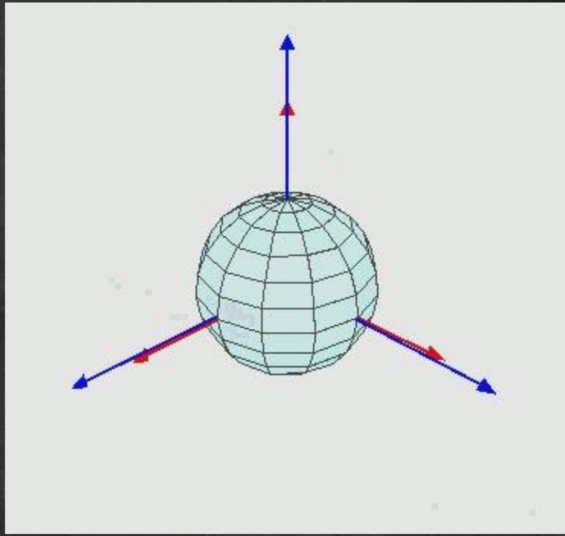
Euler’s Rotation Theorem (in 3D Space)

- Any two orthonormal coordinate frames can be related by a sequence of rotations (not more than three) about coordinate axes

- Any two Cartesian coordinate systems with a common origin are related by a rotation about some fixed axis

Euler Angle

- $R(\alpha, \beta, \gamma) = R_z(\gamma) R_y(\beta) R_x(\alpha)$
  - Product of 3 rotations around local axes
  - Rotation order is important!
    - Ex. XYZ, ZXY, YZX, etc.



✓ Intuitive control

✓ Smallest representation possible

× Non-unique representation for a given orientation

× Hard to interpolate

× Gimbal lock

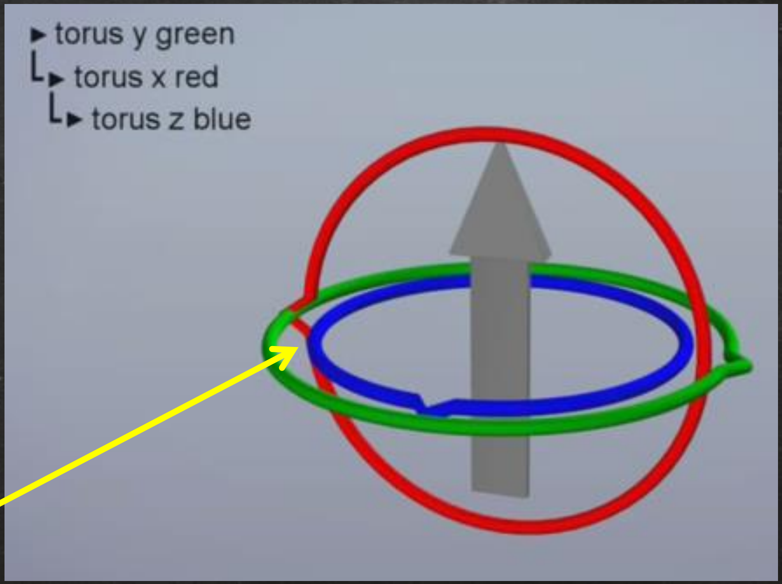
https://en.wikipedia.org/wiki/Euler_angles#/media/File:Euler2a.gif

Degree of Freedom (DOF)

- A variable describing a particular axis or dimension of movement
  - 3D Rotation: 3DOFs
    - Axis-angle: axis θ, ϕ and rotation radius α
    - Euler angle: α, β, γ
  - Rigid body transformation in 3D: 6 DOFs
    - 3 for translation and 3 for rotation

Gimbal Lock

- When the second rotation value is $\pm\pi/2$, one degree of freedom (DOF) would be lost
- Can we use any specific rotation order to avoid this?
  - Not possible!! ☹



z-axis is aligned with y-axis!!

Video: [Euler (gimbal lock) Explained by The Guerrilla CG Project]

Singularity

- A continuous subspace of the parameter space, where
  - all elements correspond to **the same** rotation
  - any movement within the subspace produces **no** change in rotation
- **NEVER** be eliminated in any 3-dimensional representation of SO(3)
  - That's why do we need quaternion!

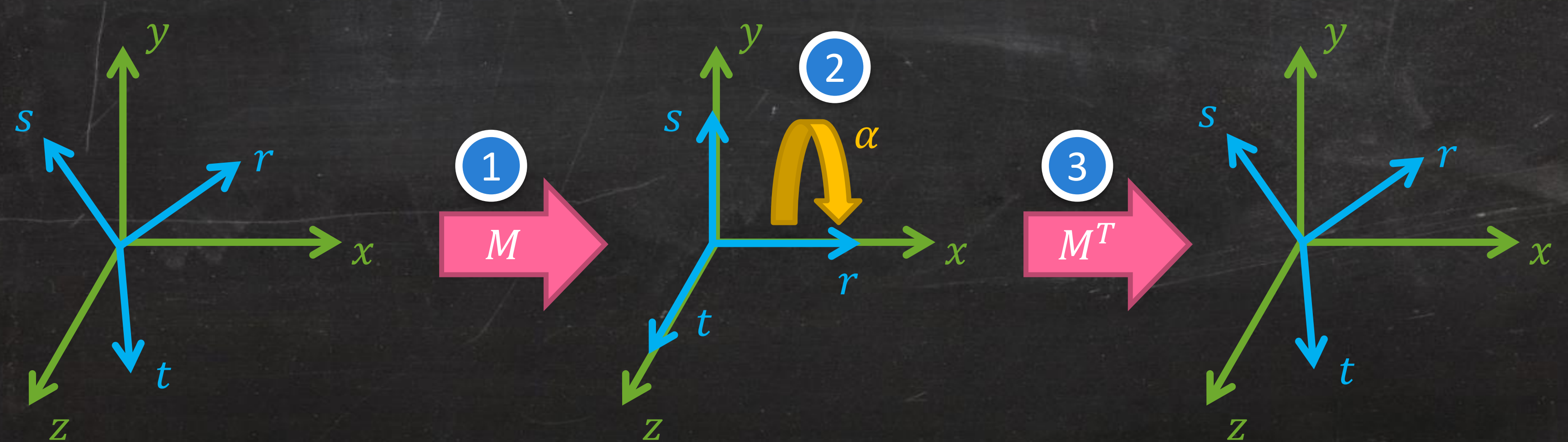
Singularity

When you go east at the **North Pole**, you are still at **the same** position!!

- A continuous subspace of the parameter space, where
  - all elements correspond to **the same** rotation
  - any movement within the subspace produces **no** change in rotation
- **NEVER** be eliminated in any 3-dimensional representation of SO(3)
  - That's why do we need quaternion!

Rotate About an Arbitrary Axis

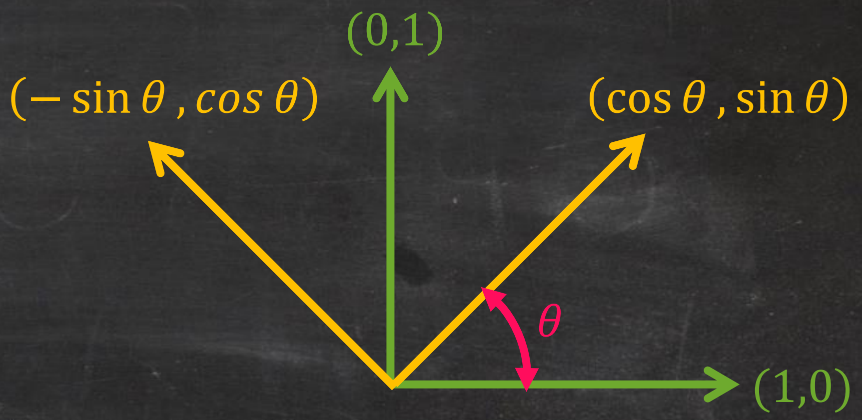
1. Change to new frame
2. Rotate α radians around
3. Transform back to standard basis



2D Rotation in Complex Plane

$$(x' + y'i) = e^{i\theta}(x + yi)$$
$$where\ e^{i\theta} = \cos\theta + i\sin\theta$$

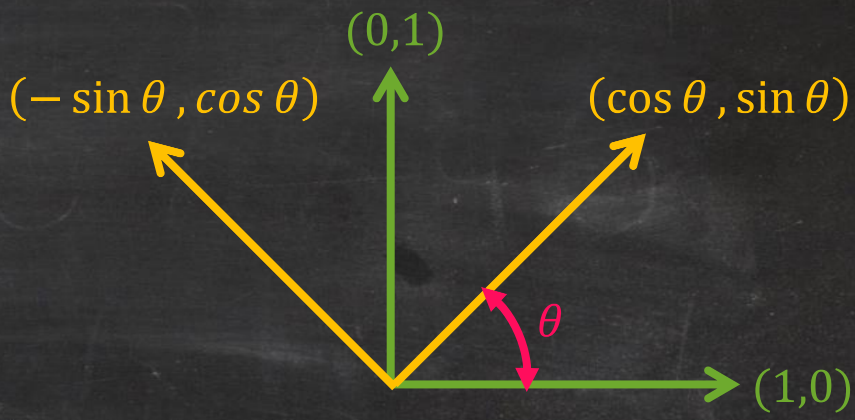
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



2D Rotation in Complex Plane

$$(x' + y'i) = e^{i\theta}(x + yi)$$
$$where\ e^{i\theta} = \cos\theta + i\sin\theta$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



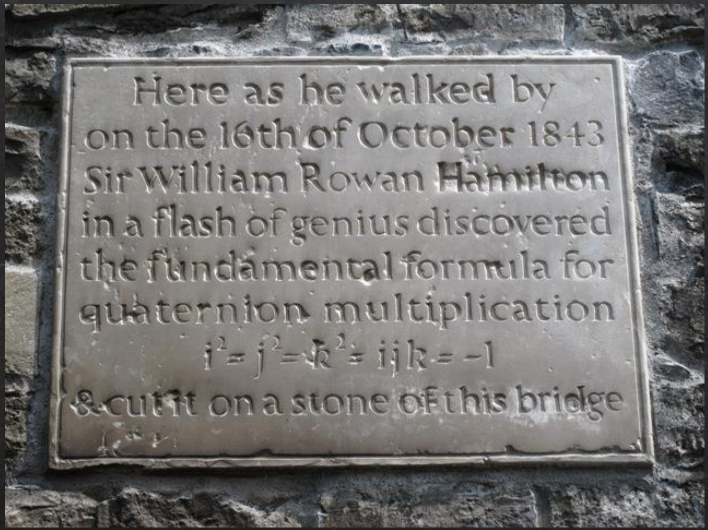
Is it possible to extend this concept to 3D?

Quaternion

- Extend complex number to 3D

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

$$ij = k, \quad jk = i, \quad ki = j$$
$$ji = -k, \quad kj = -i, \quad ik = -j$$



William Rowan Hamilton (1805–1865)

Quaternion

- Can be represented in several ways:

$$q = (w, x, y, z)$$

$$q = w + xi + yj + zk$$

$$q = w + \boldsymbol{v}$$

scalar part *vector part*

Quaternion

$$\mathbf{i^2 = j^2 = k^2 = ijk = -1}$$

$$ij = k, \quad jk = i, \quad ki = j$$
$$ji = -k, \quad kj = -i, \quad ik = -j$$

Hamilton product

$$q_0 * q_1 = (w_0 + x_0 i + y_0 j + z_0 k) * (w_1 + x_1 i + y_1 j + z_1 k)$$
$$= w_0 w_1 - x_0 x_1 - y_0 y_1 - z_0 z_1$$
$$+ (w_0 x_1 + x_0 w_1 + y_0 z_1 - z_0 y_1) i$$
$$+ (w_0 y_1 + y_0 w_1 - x_0 z_1 + z_0 x_1) j$$
$$+ (w_0 z_1 + z_0 w_1 + x_0 y_1 - y_0 x_1) k$$

Quaternion

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

$ij = k, \quad jk = i, \quad ki = j$

$ji = -k, \quad kj = -i, \quad ik = -j$

Hamilton product

$$q_0 * q_1 = (w_0 + x_0 i + y_0 j + z_0 k) * (w_1 + x_1 i + y_1 j + z_1 k)$$

$$\begin{aligned} &= w_0 w_1 - x_0 x_1 - y_0 y_1 - z_0 z_1 \\ &\quad + (w_0 x_1 + x_0 w_1 + y_0 z_1 - z_0 y_1) i \\ &\quad + (w_0 y_1 + y_0 w_1 - x_0 z_1 + z_0 x_1) j \\ &\quad + (w_0 z_1 + z_0 w_1 + x_0 y_1 - y_0 x_1) k \end{aligned}$$

$$= w_0 w_1 - \boldsymbol{v}_0 \cdot \boldsymbol{v}_1 + w_0 \boldsymbol{v}_1 + w_1 \boldsymbol{v}_0 + \boldsymbol{v}_0 \times \boldsymbol{v}_1$$

Quaternion

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

$$ij = k, \quad jk = i, \quad ki = j$$

$$ji = -k, \quad kj = -i, \quad ik = -j$$

Hamilton product

$$q_0 * q_1 = (w_0 + x_0 i + y_0 j + z_0 k) * (w_1 + x_1 i + y_1 j + z_1 k)$$

$$= w_0 w_1 - x_0 x_1 - y_0 y_1 - z_0 z_1$$
$$+ (w_0 x_1 + x_0 w_1 + y_0 z_1 - z_0 y_1) i$$
$$+ (w_0 y_1 + y_0 w_1 - x_0 z_1 + z_0 x_1) j$$
$$+ (w_0 z_1 + z_0 w_1 + x_0 y_1 - y_0 x_1) k$$

$$= w_0 w_1 - \boldsymbol{v}_0 \cdot \boldsymbol{v}_1 + w_0 \boldsymbol{v}_1 + w_1 \boldsymbol{v}_0 + \boldsymbol{v}_0 \times \boldsymbol{v}_1$$

non-commutative!

$$q_1 * q_0 \neq q_0 * q_1$$

Quaternion (Cont'd)

- Identity: $\mathbf{q} = (1, 0, 0, 0)^{\mathrm{T}}$
- Conjugate: $q^* = (w, -\boldsymbol{v})$
  - $(q^*)^* = q$
  - $(pq)^* = q^* p^*$
  - $(p + q)^* = p^* + q^*$
- $q_0 + q_1 = (w_0 + w_1, \boldsymbol{v_0} + \boldsymbol{v_1})$
- $\alpha q = q\alpha = (\alpha w, \alpha \boldsymbol{v})$

Quaternion (Cont'd)

- Norm: $N(q) = qq^* = q^*q = w^2 + x^2 + y^2 + z^2$
  - $N(\boldsymbol{q_0 q_1}) = N(\boldsymbol{q_0})N(\boldsymbol{q_1})$
  - $N(\boldsymbol{q}^*) = N(\boldsymbol{q})$
- Inverse: $\boldsymbol{q^{-1}} = \frac{\boldsymbol{q}^*}{N(\boldsymbol{q})}$
  - $\mathbf{q} \circ \mathbf{q}^{-1} = \mathbf{q}^{-1} \circ \mathbf{q} = (1, 0, 0, 0)^{\mathrm{T}}$
  - $(\boldsymbol{q_0 q_1})^{-1} = \boldsymbol{q_1^{-1} q_0^{-1}}$
- Difference: $\mathbf{q_0 q_d} = \boldsymbol{q_1} \Rightarrow \boldsymbol{q_d} = \boldsymbol{q_0^{-1} q_1}$

Unit Quaternion

$$\mathbf{q} = (\mathrm{w}, \mathrm{x}, \mathrm{y}, \mathrm{z})^T = \left[\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{v}}\right]^{\mathrm{T}}$$

Unit Quaternion

$$\mathbf{q} = (w, x, y, z)^T = \left[\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{v}}\right]^T$$

why $\frac{1}{2}$ ***???***

Rotation with Quaternion

- $\mathrm{p}' = \mathrm{Rotate}(\boldsymbol{p}) = \mathrm{q} \circ \tilde{p} \circ \mathrm{q}^{-1}$
  - Rotate a vector $\mathbf{p} \in \mathbb{R}^3$ by an **unit** quaternion $\mathrm{q} \in \mathcal{S}^3$
  - $\tilde{p} = (0, \boldsymbol{p})^{\mathrm{T}}$ extended with a zero scalar component
  - Rotate() function would *strips off* the scalar part of quaternion

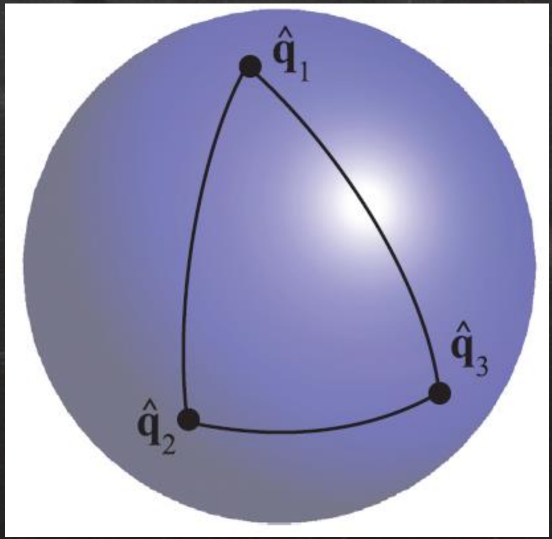


Figure from Real-time Rendering, 3/e

Quaternion – Why $\theta/2$??

Recall: $q_0 q_1 = w_0 w_1 - \boldsymbol{v_0} \cdot \boldsymbol{v_1} + w_0 \boldsymbol{v_1} + w_1 \boldsymbol{v_0} + \boldsymbol{v_0} \times \boldsymbol{v_1}$

$$
\begin{aligned}
qpq^{-1} &= (w + t\hat{v})\vec{p}(w + t\hat{v})^{-1} \\
&= (-t\hat{v} \cdot \vec{p} + w\vec{p} + t\hat{v} \times \vec{p})(w - t\hat{v}) \\
&= -wt\hat{v} \cdot \vec{p} + (w\vec{p} + t\hat{v} \times \vec{p}) \cdot t\hat{v} + w(w\vec{p} + t\hat{v} \times \vec{p}) \\
&\quad +(t\hat{v} \cdot \vec{p})t\hat{v} - (w\vec{p} + t\hat{v} \times \vec{p}) \times t\hat{v} \\
&= w^2\vec{p} + 2wt\hat{v} \times \vec{p} + t^2(\hat{v} \cdot \vec{p})\hat{v} - t^2\hat{v} \times \vec{p} \times \hat{v} \\
&= (w^2 - t^2)\vec{p} + 2wt\hat{v} \times \vec{p} + 2t^2(\vec{p} \cdot \hat{v})\hat{v}
\end{aligned}
$$

ps. suppose $\boldsymbol{q}$ is an unit quaternion

Quaternion – Why $\theta/2$??

Recall: $q_0 q_1 = w_0 w_1 - \boldsymbol{v_0} \cdot \boldsymbol{v_1} + w_0 \boldsymbol{v_1} + w_1 \boldsymbol{v_0} + \boldsymbol{v_0} \times \boldsymbol{v_1}$

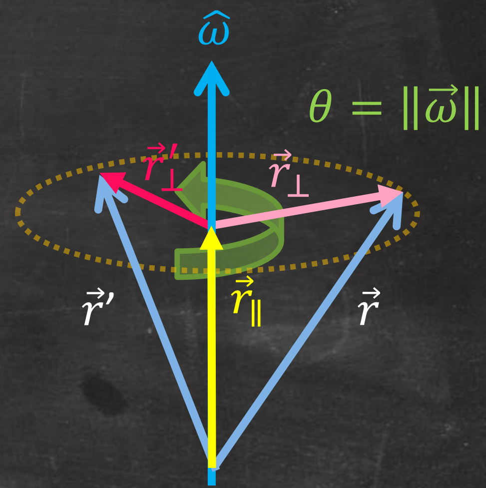
$$
\begin{aligned}
qpq^{-1} &= (w + t\hat{v})\vec{p}(w + t\hat{v})^{-1} \\
&= (-t\hat{v} \cdot \vec{p} + w\vec{p} + t\hat{v} \times \vec{p})(w - t\hat{v}) \\
&= -wt\hat{v} \cdot \vec{p} + (w\vec{p} + t\hat{v} \times \vec{p}) \cdot t\hat{v} + w(w\vec{p} + t\hat{v} \times \vec{p}) \\
&\quad +(t\hat{v} \cdot \vec{p})t\hat{v} - (w\vec{p} + t\hat{v} \times \vec{p}) \times t\hat{v} \\
&= w^2\vec{p} + 2wt\hat{v} \times \vec{p} + t^2(\hat{v} \cdot \vec{p})\hat{v} - t^2\hat{v} \times \vec{p} \times \hat{v} \\
&= (w^2 - t^2)\vec{p} + 2wt\hat{v} \times \vec{p} + 2t^2(\vec{p} \cdot \hat{v})\hat{v}
\end{aligned}
$$

Look familiar??

ps. suppose $\boldsymbol{q}$ is an unit quaternion

Recall

Axis-Angle Rotation



$$\vec{r}' = \cos\theta \; \vec{r} + \sin\theta \, (\hat{\omega} \times \vec{r}) + (1 - \cos\theta)\big((\vec{\mathrm{r}} \cdot \hat{\omega})\hat{\omega}\big)$$

Quaternion – Why $\theta/2$?? (Cont'd)

$$
\begin{aligned}
qpq^{-1} &= (w + t\hat{v})\vec{p}(w + t\hat{v})^{-1} \\
&= (-t\hat{v}\cdot\vec{p} + w\vec{p} + t\hat{v}\times\vec{p})(w - t\hat{v}) \\
&= -wt\hat{v}\cdot\vec{p} + (w\vec{p} + t\hat{v}\times\vec{p})\cdot t\hat{v} + w(w\vec{p} + t\hat{v}\times\vec{p}) \\
&\quad +(t\hat{v}\cdot\vec{p})t\hat{v} - (w\vec{p} + t\hat{v}\times\vec{p})\times t\hat{v} \\
&= w^2\vec{p} + 2wt\hat{v}\times\vec{p} + t^2(\hat{v}\cdot\vec{p})\hat{v} - t^2\hat{v}\times\vec{p}\times\hat{v} \\
&= (w^2 - t^2)\vec{p} + 2wt\hat{v}\times\vec{p} + 2t^2(\vec{p}\cdot\hat{v})\hat{v}
\end{aligned}
$$

$$
\vec{r}\,' = \cos\theta\ \vec{r} + \sin\theta\,(\hat{\omega}\times\vec{r}) + (1 - \cos\theta)\big((\vec{\mathrm{r}}\cdot\hat{\omega})\hat{\omega}\big)
$$

Quaternion – Why $\theta/2$?? (Cont'd)

$$w^2 - t^2 = \cos\theta$$
$$2wt = \sin\theta$$
$$2t^2 = 1 - \cos\theta \;\Rightarrow t = \sin\frac{\theta}{2} \;\Rightarrow w = \cos\frac{\theta}{2}$$

where $2\sin\theta\cos\theta = \sin 2\theta$

Therefore the **unit** quaternion is

$\boldsymbol{q} = (\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{v}}) \leftrightarrow$ rotate θ around axis $\hat{\mathbf{v}}$

Quaternion – Why $\theta/2$?? (Cont'd)

$$w^2 - t^2 = \cos\theta$$

$$2wt = \sin\theta$$

$$2t^2$$

Read More

1. Quaternions, Ken Shoemake.
2. Game Physics 2/e, Ch10, David H. Eberly.

$$\sin\theta\cos\theta = \sin 2\theta$$

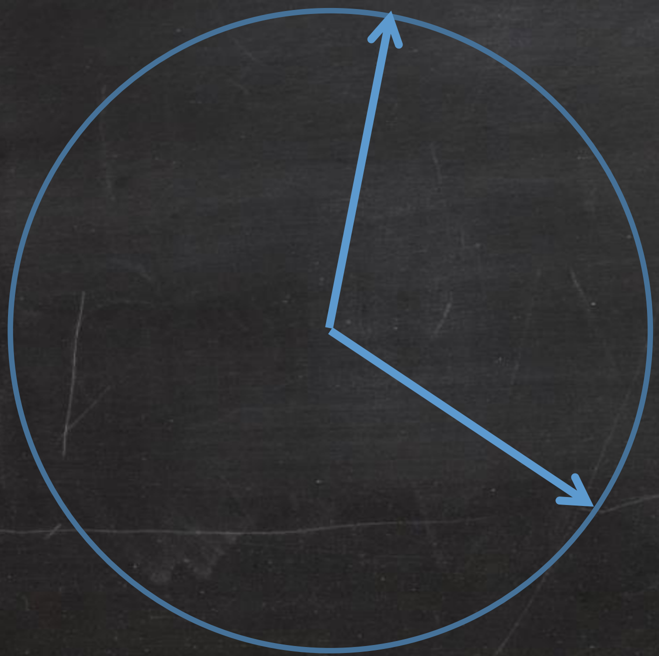
Therefore the **unit** quaternion is

$$\boldsymbol{q} = \left(\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{v}}\right) \leftrightarrow \text{rotate } \theta \text{ around axis } \hat{\mathbf{v}}$$

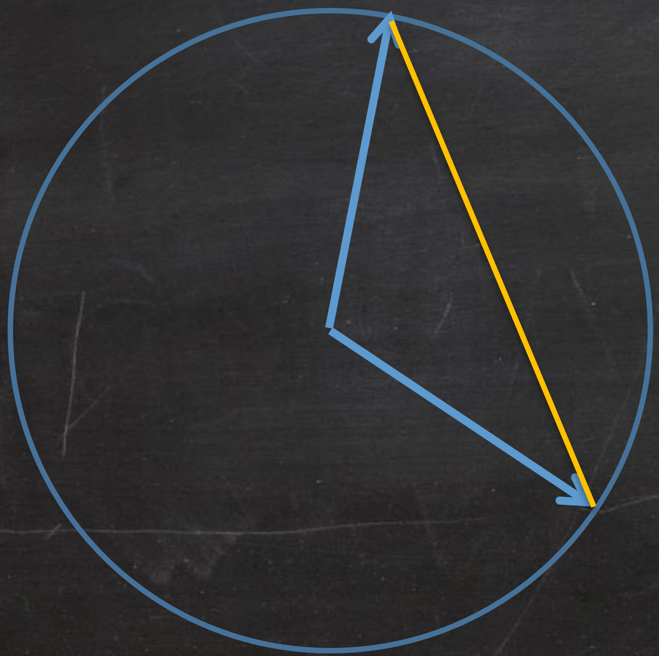
Quaternion qpq^{-1}

- Concatenation
  - $q_1 \cdot (q_0 \cdot p \cdot q_0^{-1}) \cdot q_1^{-1} = (q_1 \cdot q_0) \cdot p \cdot (q_1 \cdot q_0)^{-1}$
- Any non-zero real multiple of q gives the same action
  - $(sq)p(sq)^{-1} = (sq)p(q^{-1}s^{-1}) = qpq^{-1}ss^{-1} = qpq^{-1}$

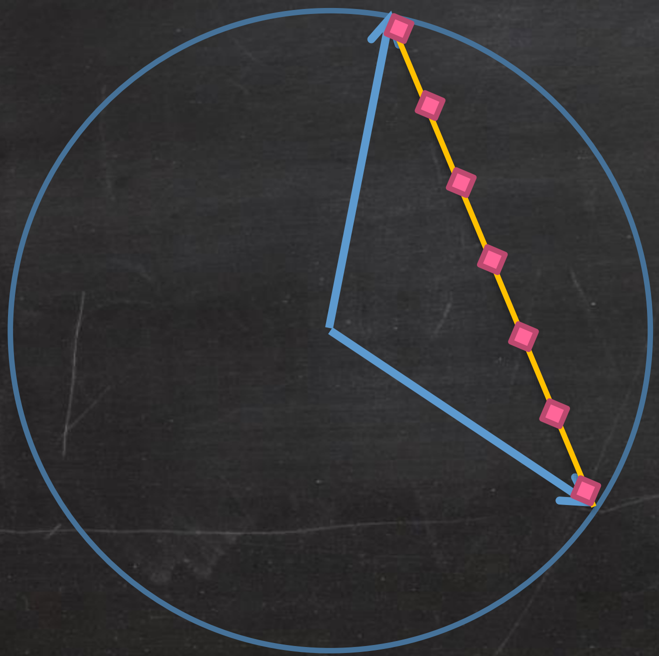
Quaternion – Linear Interpolation



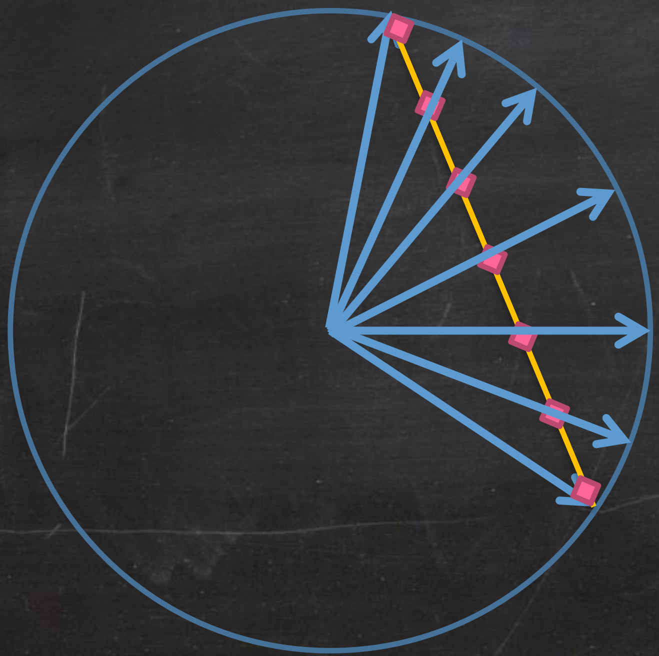
Quaternion – Linear Interpolation



Quaternion – Linear Interpolation

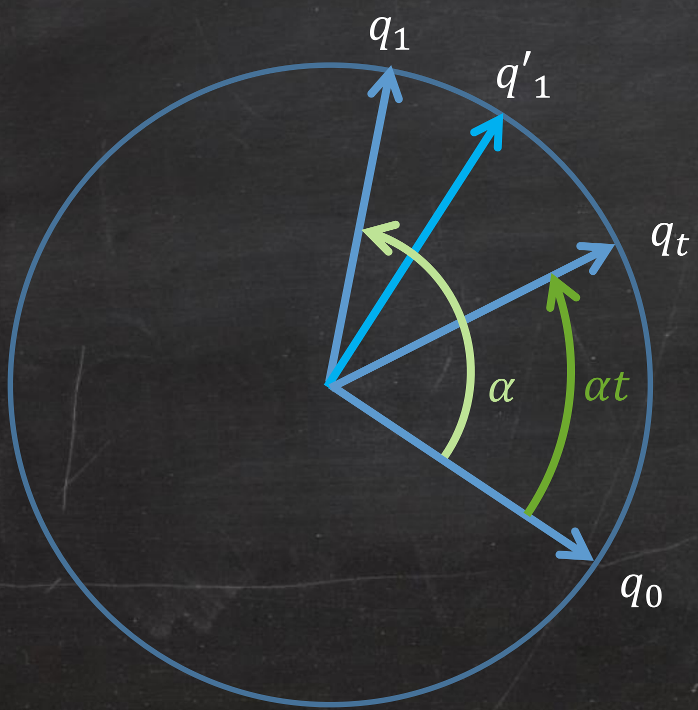


Quaternion – Linear Interpolation



Its angular speed is **NOT** constant!

Quaternion – Spherical Linear Interpolation



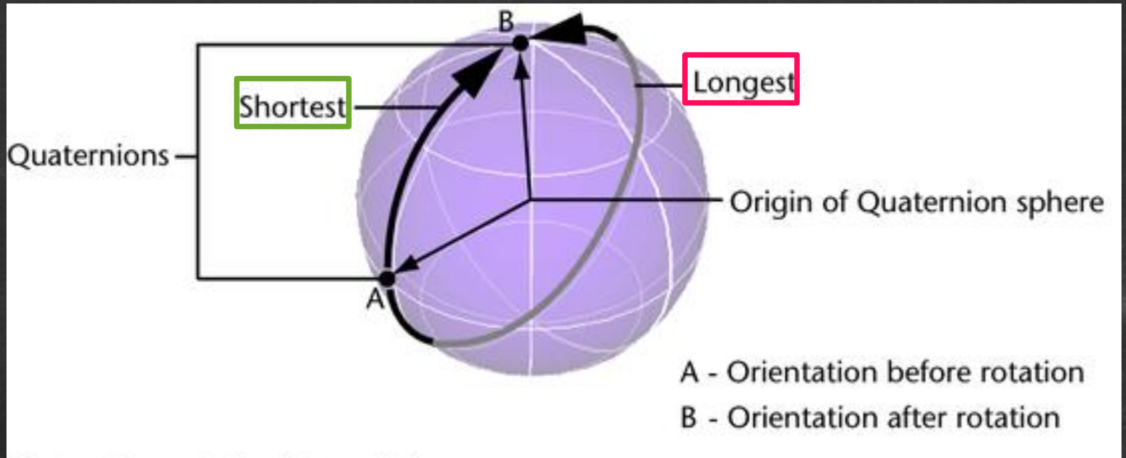
$$q_t = (\cos \alpha t) q_0 + (\sin \alpha t) q'_1$$

$$q'_1 = \frac{q_1 - \cos \alpha\, q_0}{\sin \alpha}$$

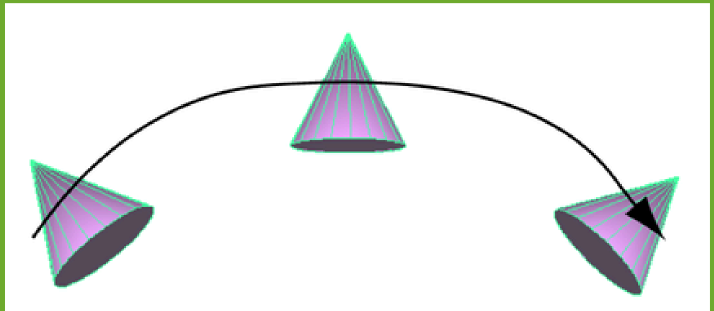
$$q_t = \frac{\sin(1-t)\alpha}{\sin \alpha} \mathrm{q}_0 + \frac{\sin \alpha t}{\sin \alpha} \mathrm{q}_1$$

Numerical error as $\alpha \to 0$, use lerp instead!

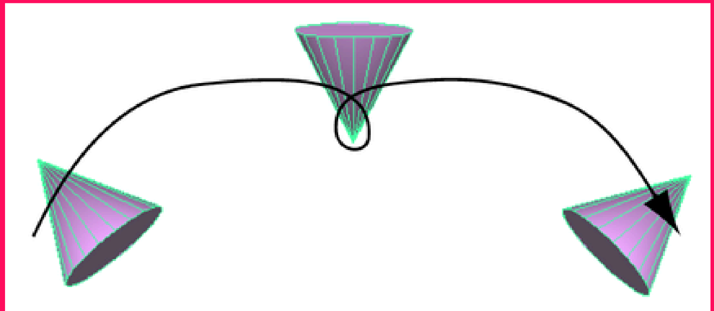
Quaternion - Interpolation Path



Quaternion rotation interpolation



Quaternion Shortest



Quaternion Longest

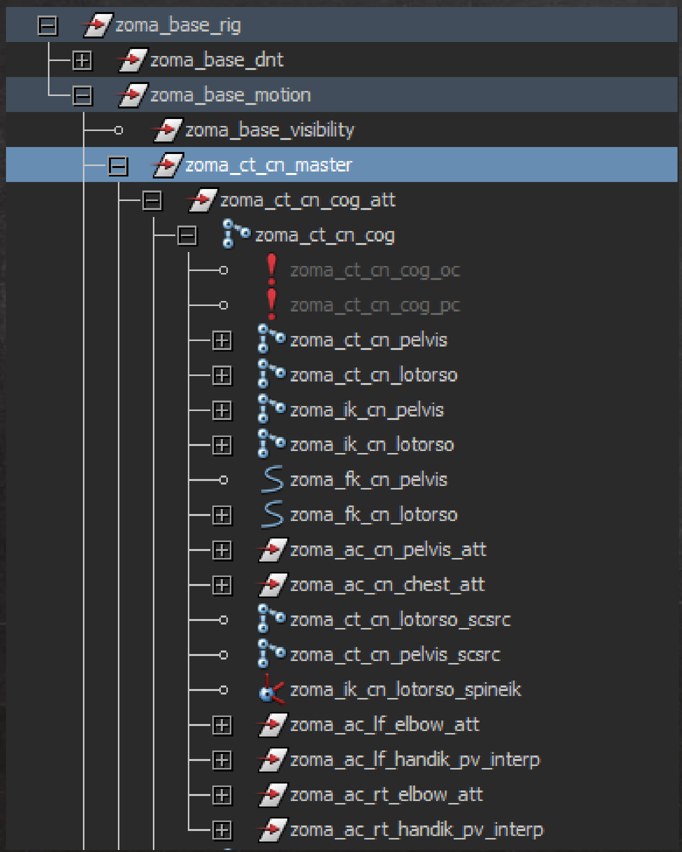
[Animated rotation in Maya]()

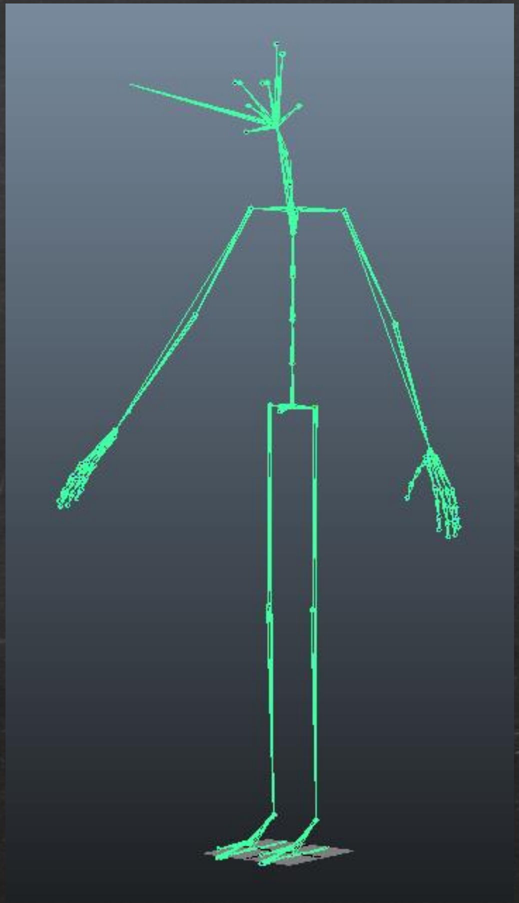
Why Quaternion?

- Smooth interpolation with slerp
- Without singularity (Gimbal Lock)
- Compact representation (only 4 numbers)
- Fast conversion from/to matrix representation
- Fast concatenation and inversion of angular displacements

Character Animation

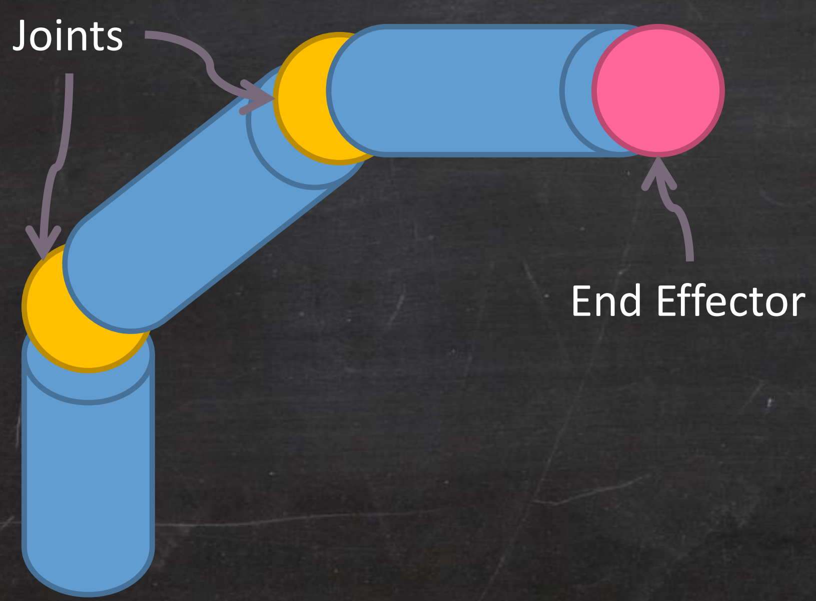
Skeleton

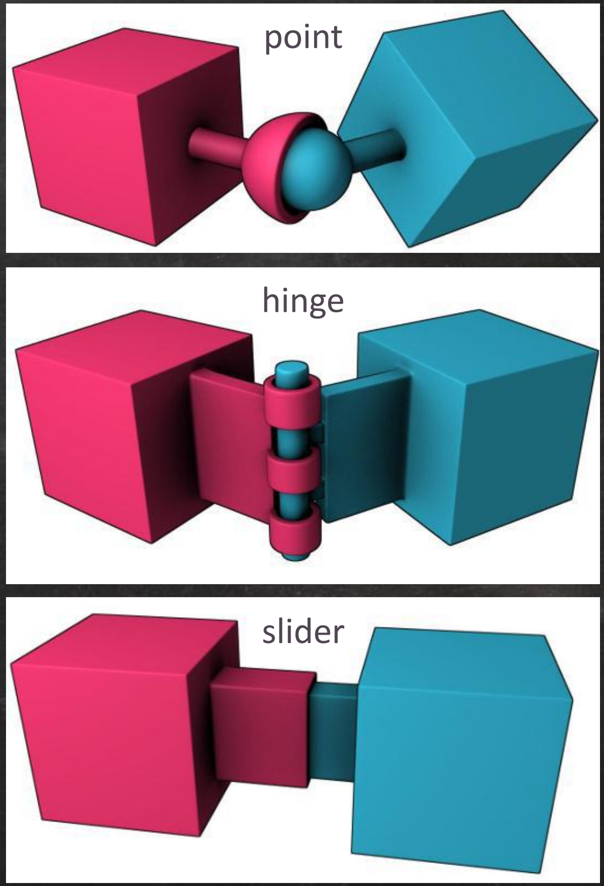




Hotel Transylvania / Zombie Rig from SONY Pictures Animation

Kinematic Chain





Bullet constraint types

Recall

Degree of Freedom (DOF)

- A variable describing a particular axis or dimension of movement within a joint
- Rigid body transformation
  - 6 DOFs
  - 3 for position and 3 for rotation
- **Pose:** a vector of N numbers that maps to a set of DOFs in the skeleton

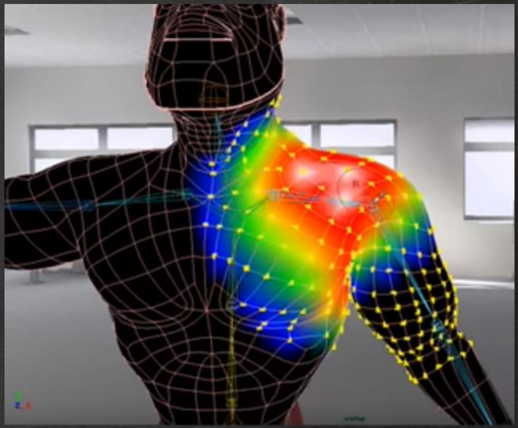
Forward Kinematics

Hotel Transylvania / Zombie Rig from SONY Pictures Animation

Inverse Kinematics

[Hotel Transylvania / Zombie Rig](#) *from SONY Pictures Animation*

Linear Blend Skinning (LBS)



transformation of *joint j*

$$v_i' = \sum_{j=1}^{m} w_{i,j} T_j v_i = \left(\sum_{j=1}^{m} w_{i,j} T_j \right) v_i$$

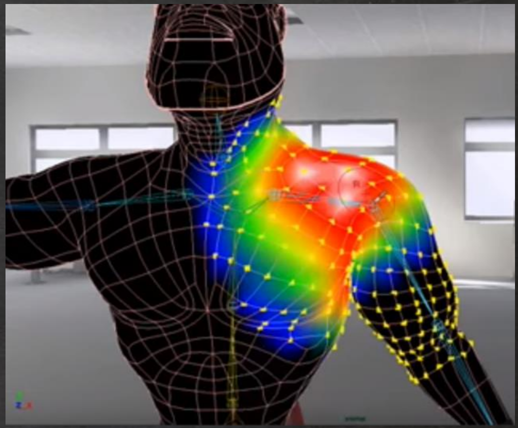
$$\sum_{j=1}^{m} w_{i,j} = 1,$$
$$0 \leq w_{i,j} \leq 1$$

blending weights for *joint j* to *vertex i*

Rigid binding: each vertex is only affected by one joint

Smooth binding: each vertex is affected by multiple joints (< 4)

Linear Blend Skinning (LBS)



transformation of *joint j*

$$v_i' = \sum_{j=1}^{m} w_{i,j} T_j v_i = \left(\sum_{j=1}^{m} w_{i,j} T_j \right) v_i$$

$$\sum_{j=1}^{m} w_{i,j} = 1, \quad 0 \le w_{i,j} \le 1$$

blending weights for *joint j* to *vertex i*

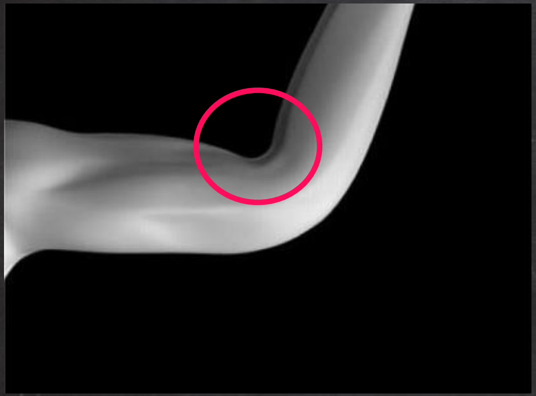
Bad smell, lerping matrices!?

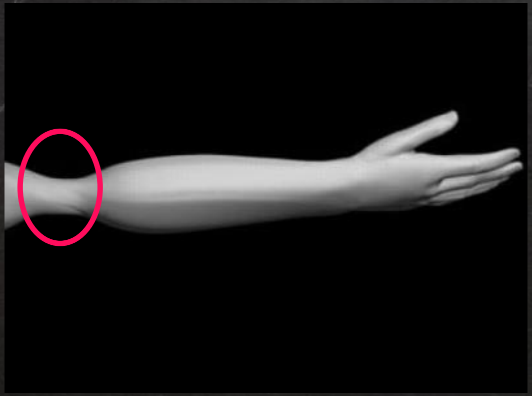
Rigid binding: each vertex is only affected by one joint

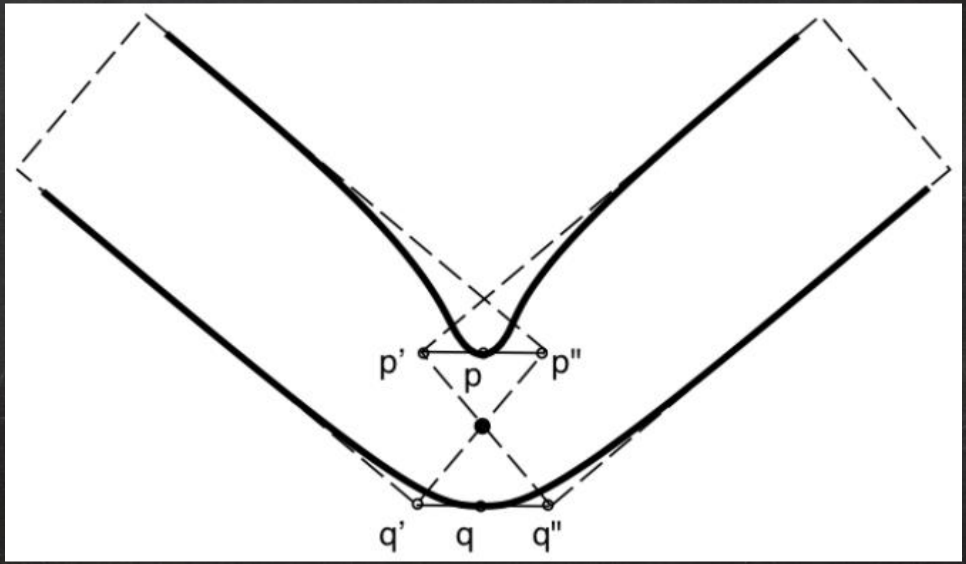
Smooth binding: each vertex is affected by multiple joints (< 4)

Direct Matrix Interpolation

- Lerped rotation matrix is NOT a rotation matrix



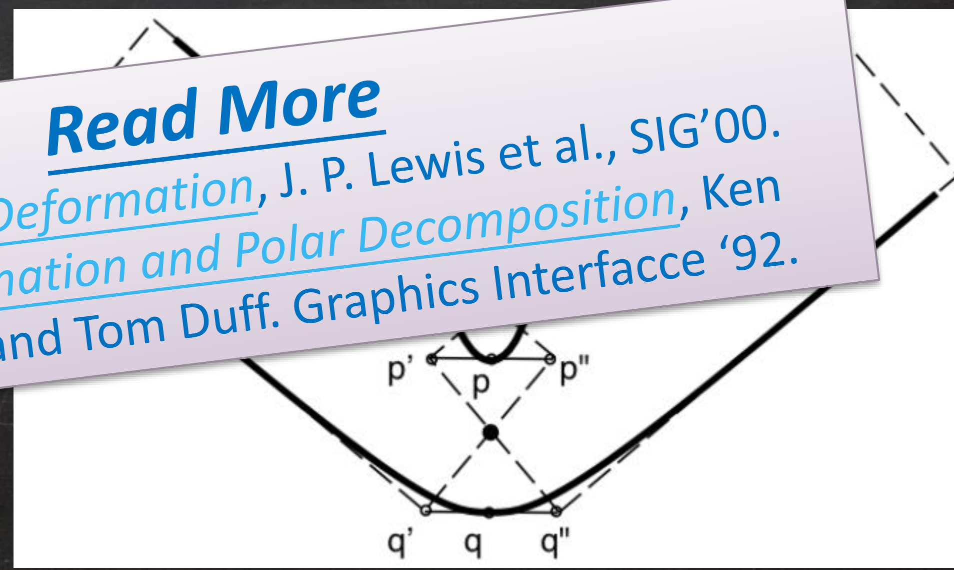




[Lewis et al., SIG'00]

Direct Matrix Interpolation

- Lerped rotation matrix is NOT a rotation matrix



[Lewis et al., SIG'00]

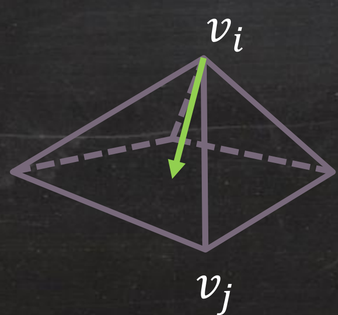
Read More

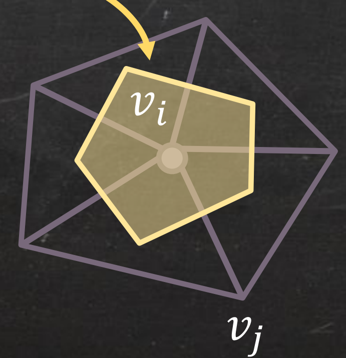
1. *Pose Space Deformation*, J. P. Lewis et al., SIG'00.
2. Matrix Animation and Polar Decomposition, Ken Shoemake and Tom Duff. Graphics Interfacce '92.

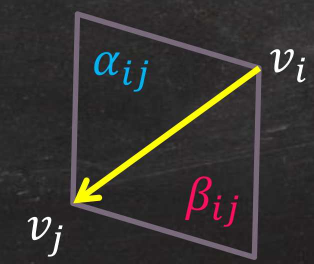
Discrete Laplace-Beltrami

Measures the difference between the value of the function at that point and the average of the values at surrounding points

$$L_C(v_i) = \frac{1}{2A(v_i)} \sum_{v} (\cot \alpha_{ij} + \cot \beta_{ij})(v_j - v_i)$$

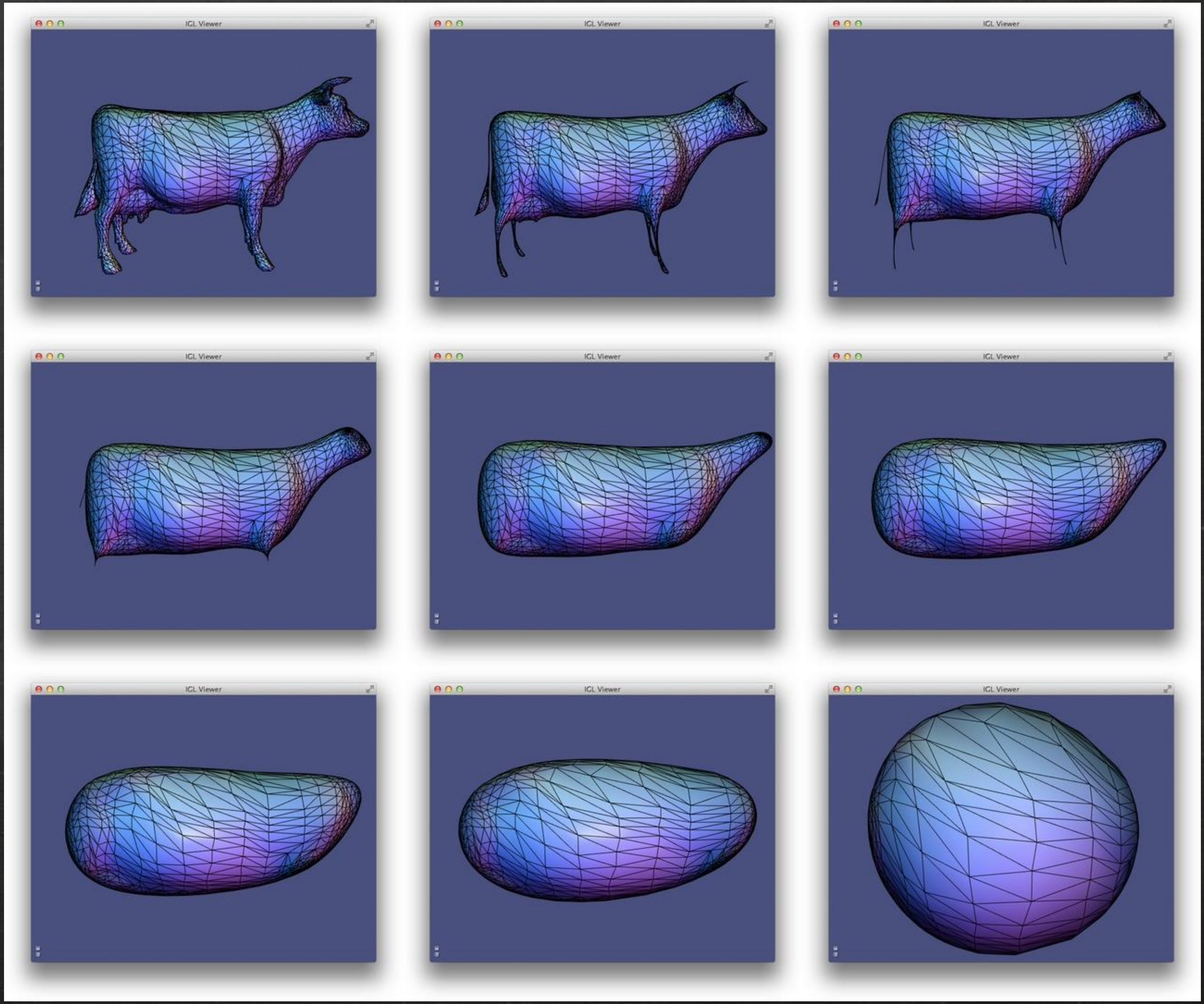




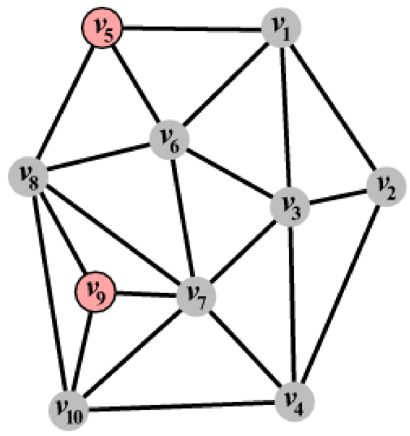


Mesh Smoothing

$$V' = L_C(V) + V$$



http://libigl.github.io/libigl/tutorial/tutorial.html



The mesh

4	–1	–1		–1	–1				
–1	3	–1	–1						
–1	–1	5	–1		–1	–1			
	–1	–1	4			–1			–1
–1				3	–1		–1		
–1		–1			4	–1	–1		
		–1	–1		–1	6	–1	–1	–1
				–1	–1	–1	6	–1	–1
						–1	–1	3	–1
			–1			–1	–1	–1	4

The symmetric Laplacian L_s

4	–1	–1			–1				
–1	3	–1	–1						
–1	–1	5	–1		–1	–1			
	–1	–1	4			–1			–1
–1		–1			4	–1	–1		
		–1	–1		–1	6	–1		–1
					–1	–1	6		–1
			–1			–1	–1		4

Invertible Laplacian

4	–1	–1		–1	–1				
–1	3	–1	–1						
–1	–1	5	–1		–1	–1			
	–1	–1	4			–1			–1
–1				3	–1		–1		
–1		–1			4	–1	–1		
		–1	–1		–1	6	–1	–1	–1
				–1	–1	–1	6	–1	–1
						–1	–1	3	–1
			–1			–1	–1	–1	4
				1					
								1	

2-anchor $\tilde{L}$

[Sorkine et al., EG'05]

Deformation

$$shape = f(space)$$

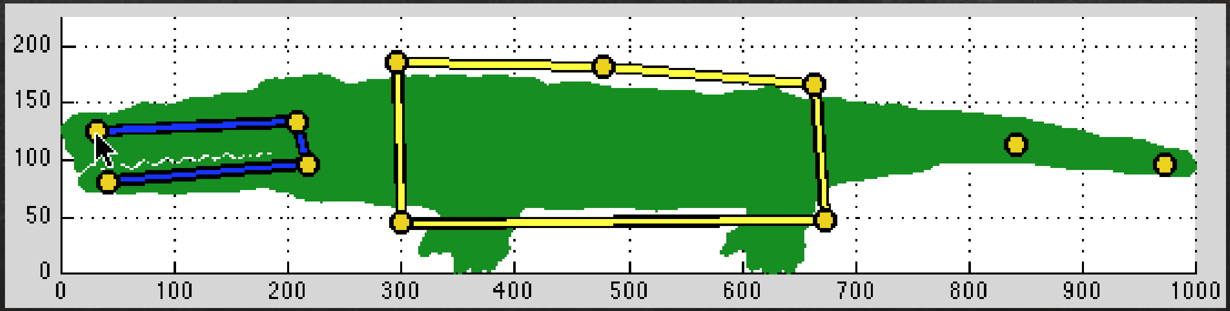
space/volume/free-form deformer

$$shape = f(shape)$$

surface deformer

Deformer

- Change the position of vertices
  - Vertices in, vertices out
  - Topology is unchanged
- Users manipulate the shape via handles such as
  - curve
  - cage
  - proxy mesh
  - etc.



[Jacobson et al., SIG'11]

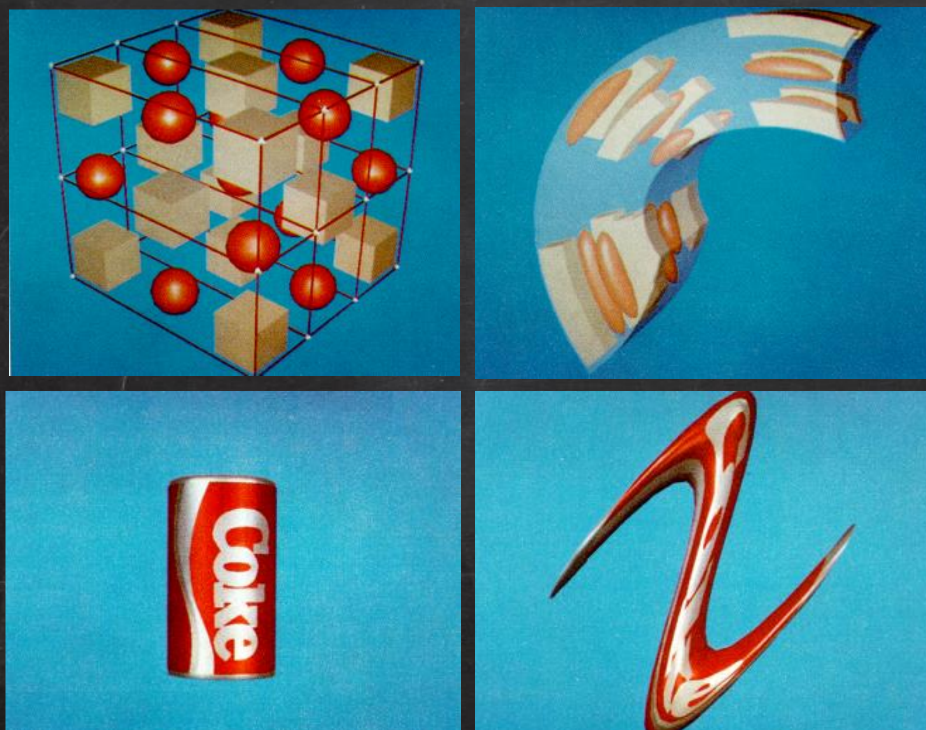
Why Deformer?

- Manipulate mesh for aesthetic purposes
  - Squash, stretch, collision, etc.
- Character posing for animation
- Fake dynamics
  - Secondary animation by using procedural
- Simulation post-fix?
  - I think it would be great for production

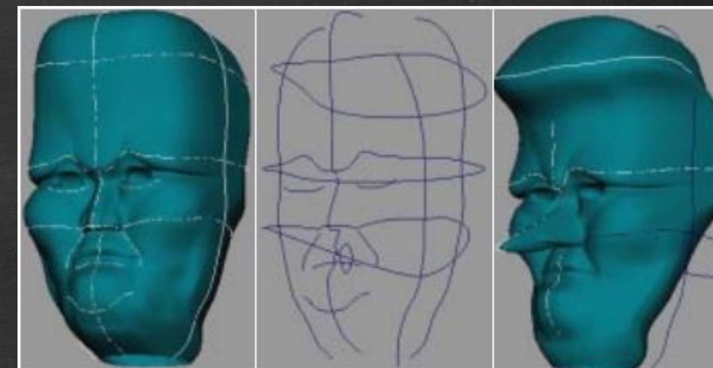
Deformer Requirements

- Sufficiently fast & robust
- Easy to setup and control
- Aesthetically pleasing
  - Physically plausible
  - Preserve local details or volume
- Large scale deformation (optional)

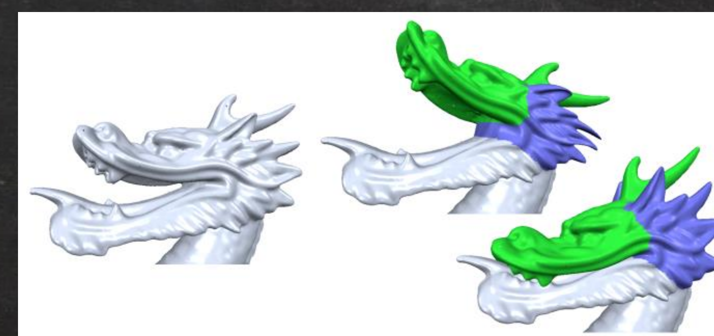
Space Deformation: $shape = f(space)$



[Sederberg and Parry, SIG'86]



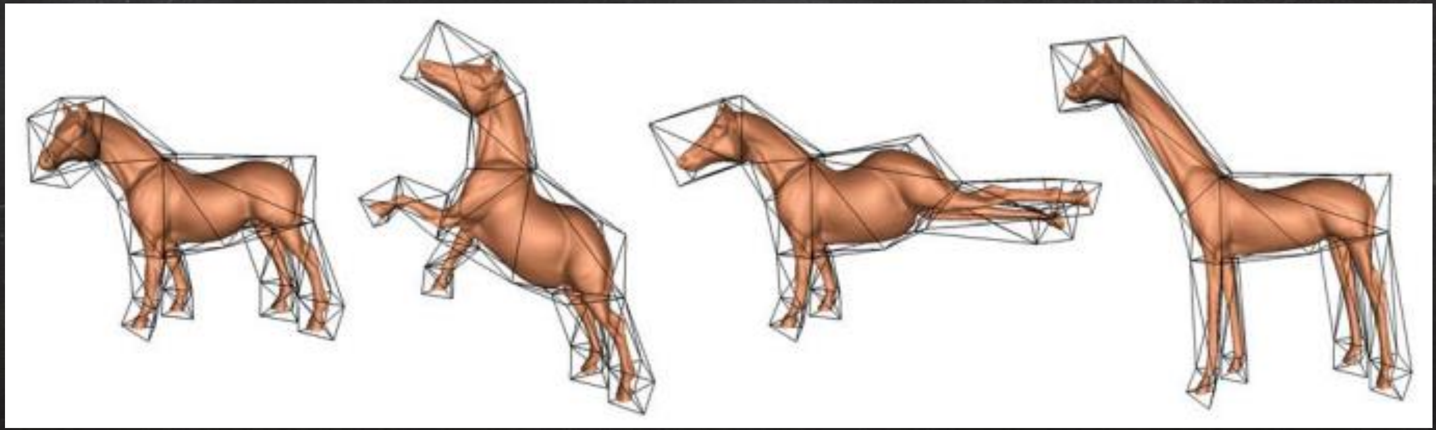
[Singh and Fiume 98]



[Botsch and Kobbelt, EG'05]

Space Deformation: $shape = f(space)$

[Joshi et al., SIG'07]



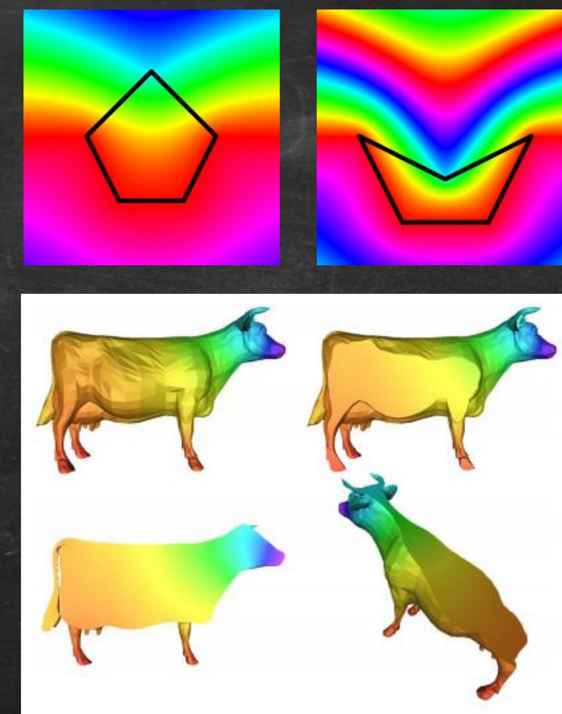
[Ju et al., SIG'05]

Coordinate Mapping

- How do we compute the weights inside?
  Ans.: Generalized Barycentric Coordinates

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

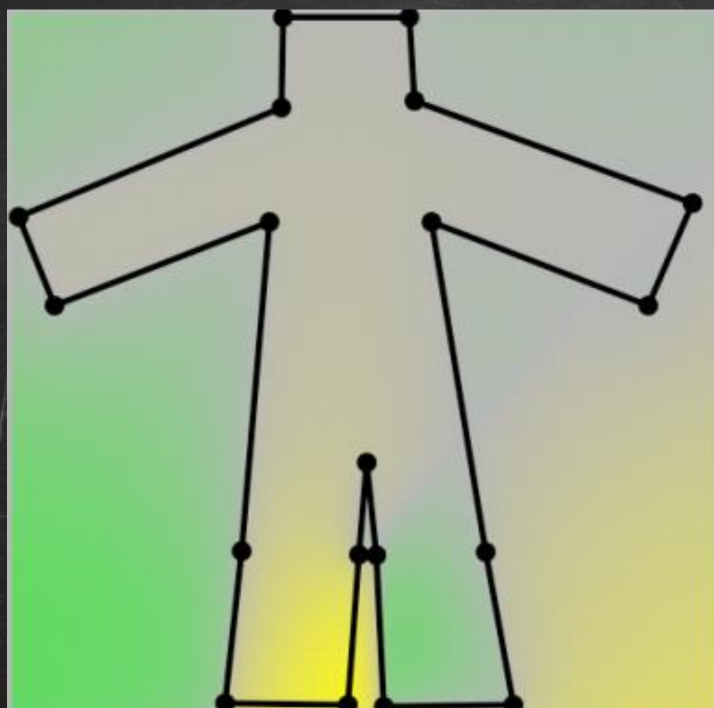
should be smooth!!



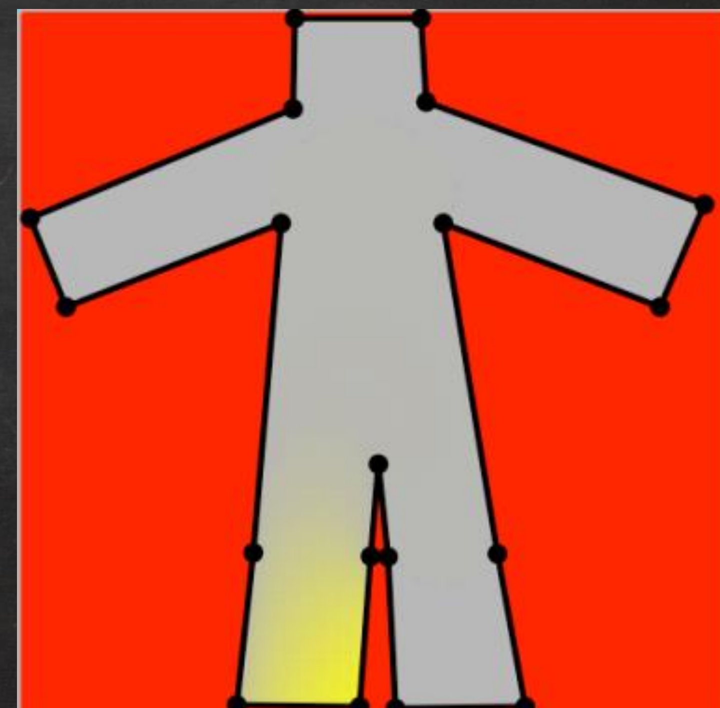
[Ju et al. '05]

Coordinate Mapping (Cont'd)

Mean Value Coordinate



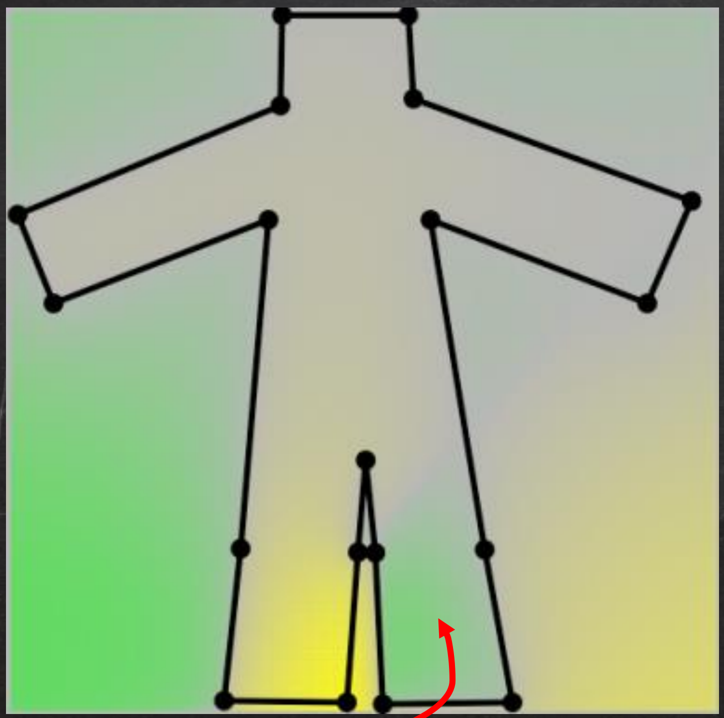
Harmonic Coordinate



[Joshi et al., SIG'07]

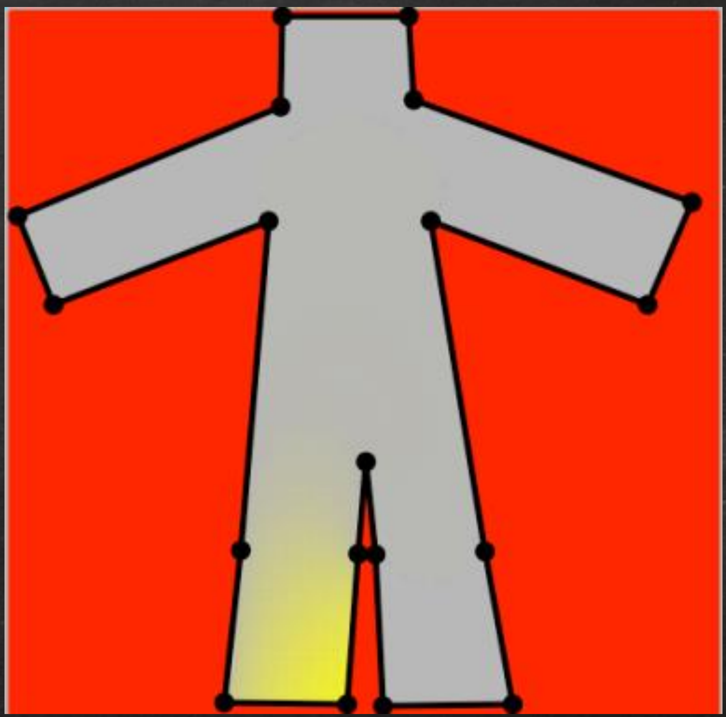
Coordinate Mapping (Cont'd)

Mean Value Coordinate



negative weights!!

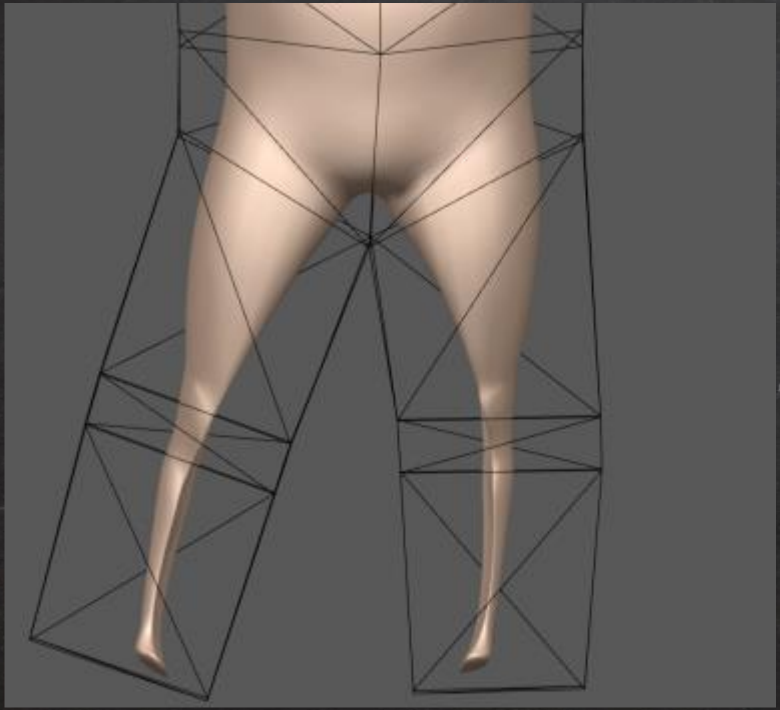
Harmonic Coordinate



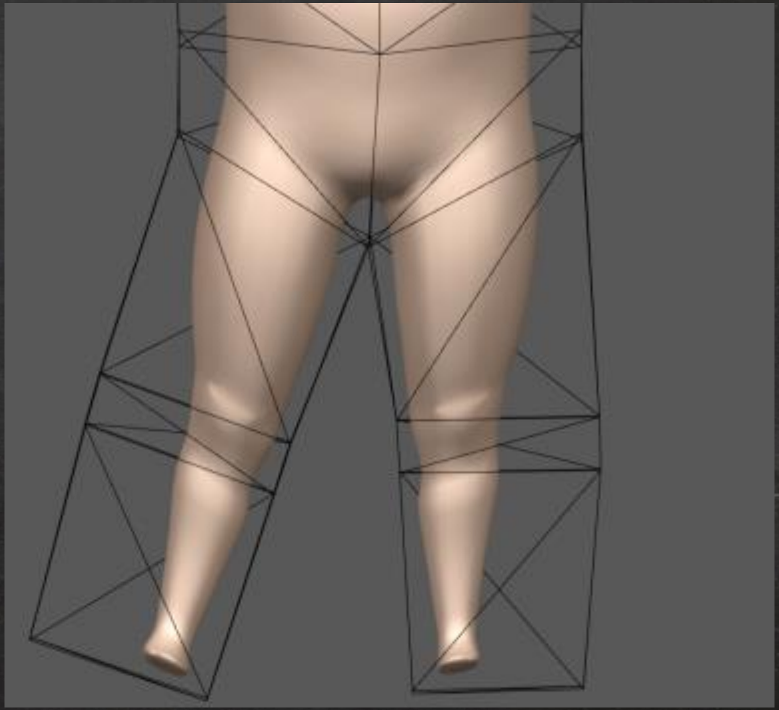
[Joshi et al., SIG'07]

Coordinate Mapping (Cont'd)

Mean Value Coordinate

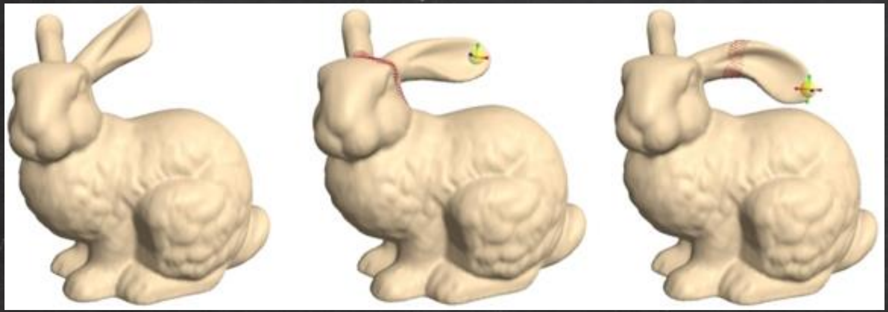


Harmonic Coordinate

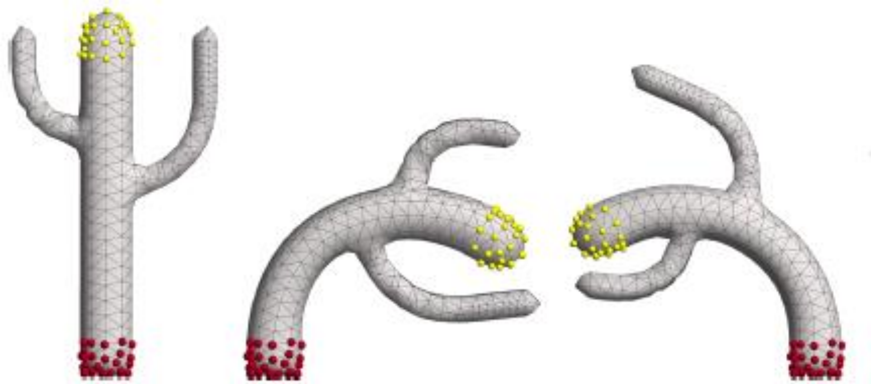


[Joshi et al., SIG'07]

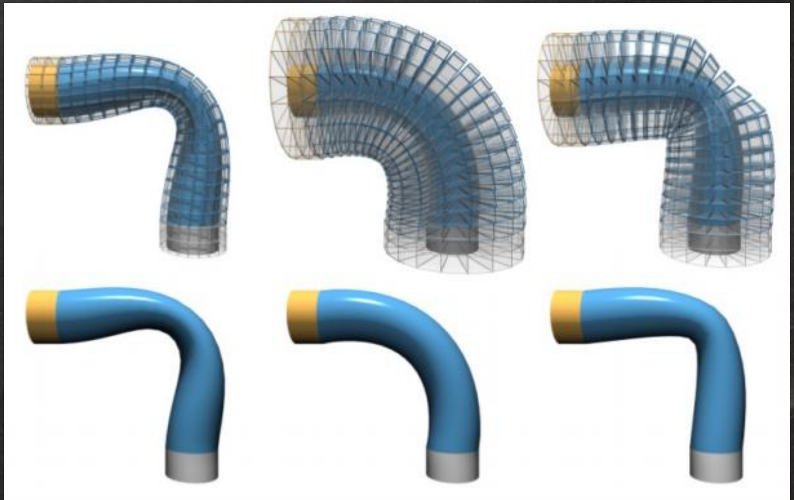
Surface Deformation: $shape = f(shape)$



[Sorkine et al., SGP'04]



[Sorkine and Alexa, SGP'07]



[Botsch et al., SGP'06]

General Framework of Surface Deformation

$$x' = \arg\min_{x'} f(x')$$

$$\text{subject to } x_i' = c_i$$

General Framework of Surface Deformation

objective (energy function)

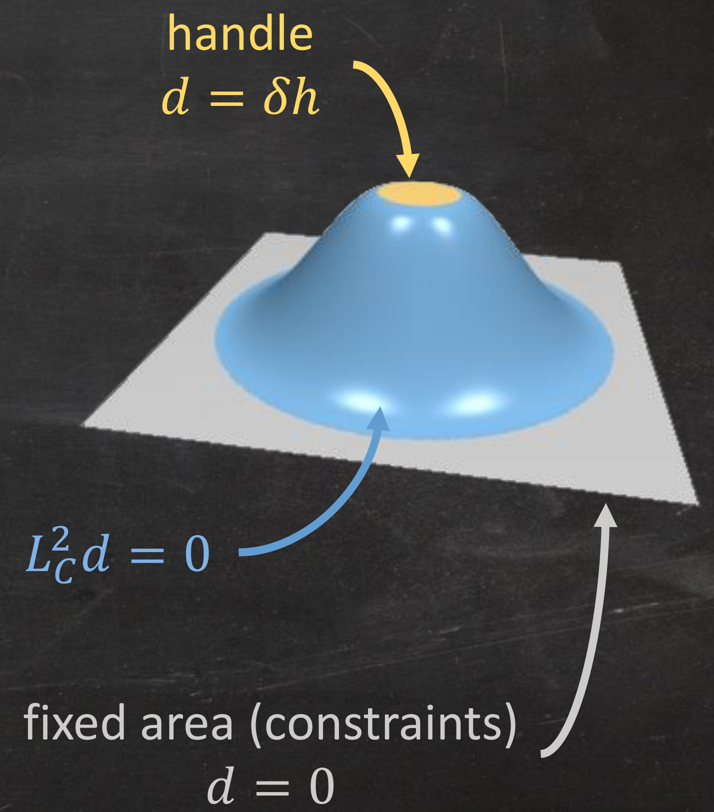
$$x' = \arg\min_{x'} f(x')$$

$$\text{subject to } x'_i = c_i$$

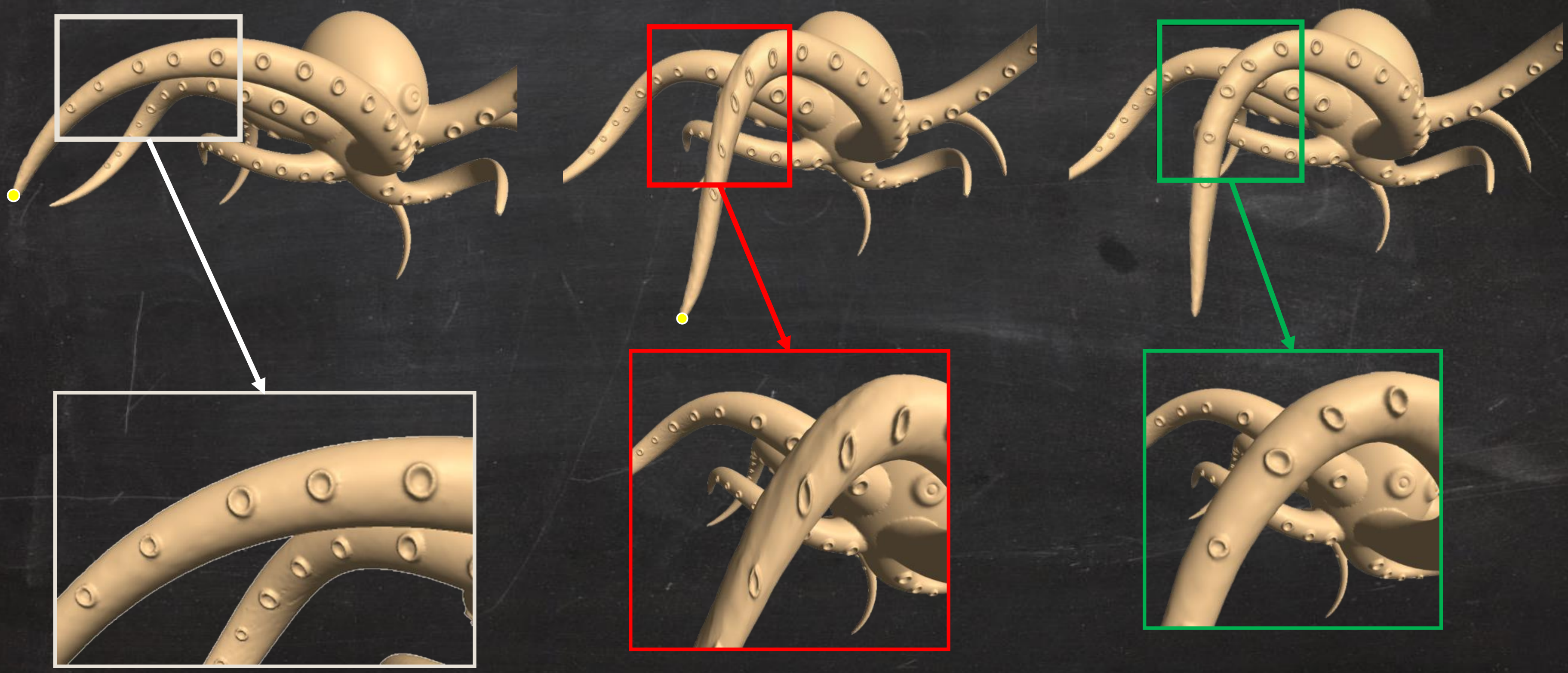
equality constraints

Bi-Harmonic Deformation

$$\begin{bmatrix} & L_c^2 & \\ 0 & \mathbf{I} & 0 \\ 0 & 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} \vdots \\ d_i \\ \vdots \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \delta h_i \end{bmatrix}$$



Laplacian Surface Editing



[Sorkine et al., SGP'04]

Laplacian Surface Editing (Cont'd)

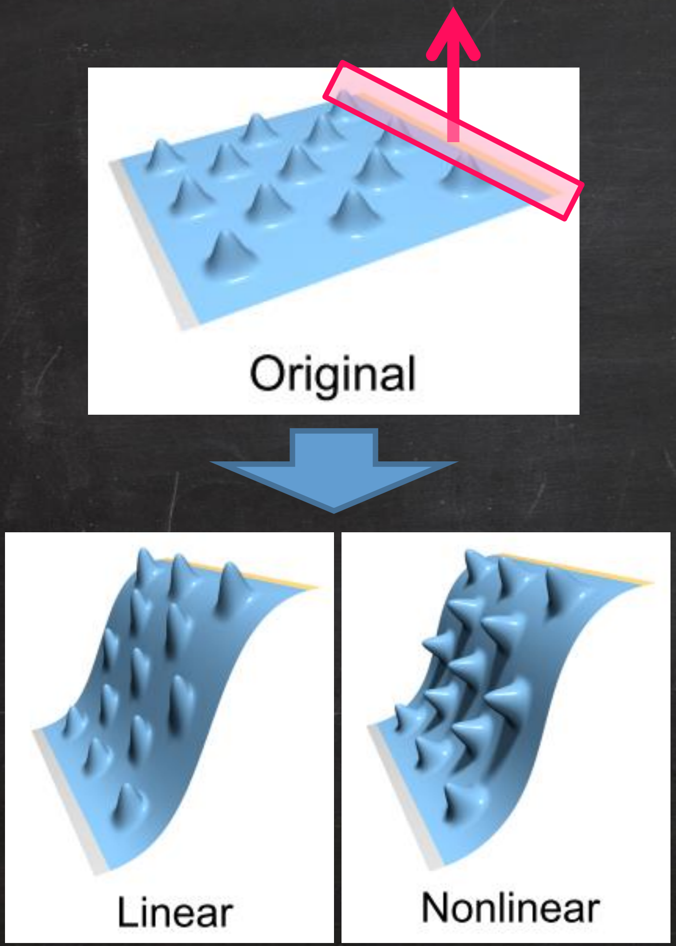
$$v' = \arg\min_{\mathrm{v}'} \left(\sum_{i=1}^{n} \| L_c(v_i') - T_i L_C(v_i) \|^2 + \sum_{j \in C} \| v_j' - u_j \|^2 \right)$$

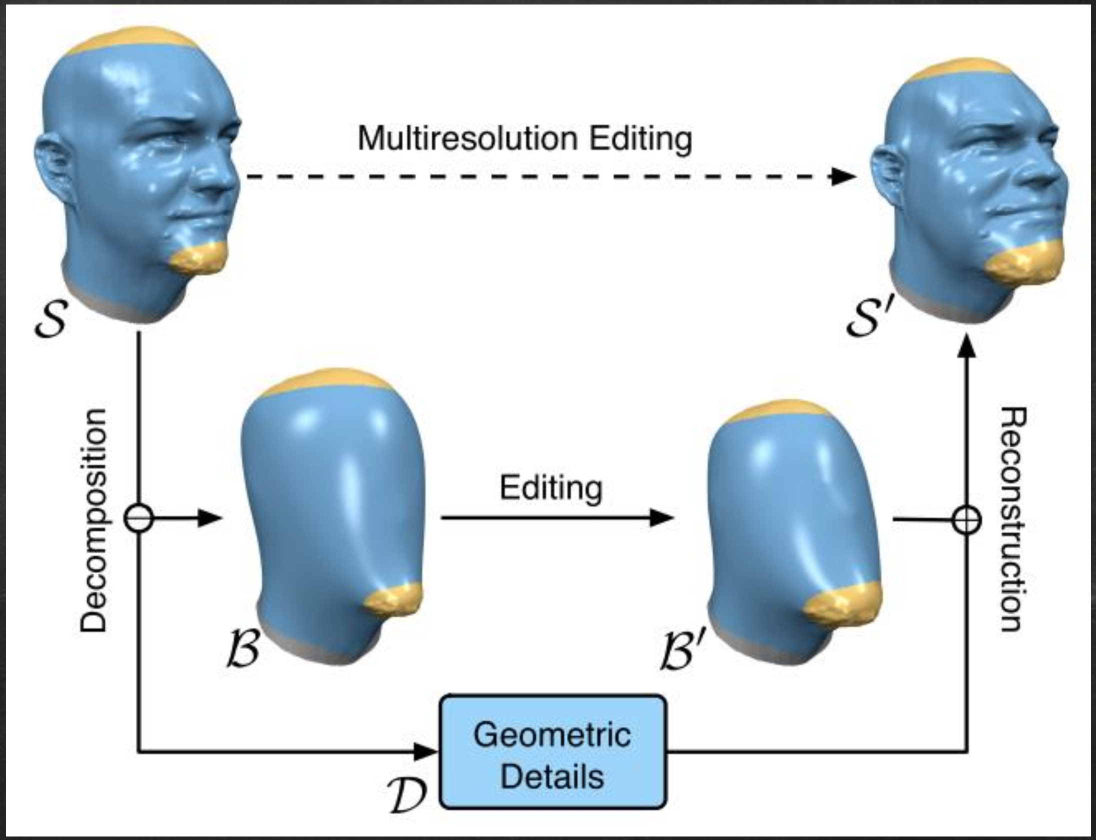
similarity transformation

Laplacian coordinate is not rotation invariant,
thus we need T_i for alignment (rotation + scale).

user constraints

Multiresolution Editing





[Botsch and Sorkine. TVCG'08]

Face Animation

- Given a set of models for each facial expression
  - Each model has identical topology
- How to tweak the expression via parameters?
  - PCA (Principal Component Analysis)
  - BlendShapes

BlendShape

$$f = b_0 + \sum_{k=1}^{n} w_k(b_k - b_0)$$

$$f = b_0 + Bw$$

Character Rig from Mery Project

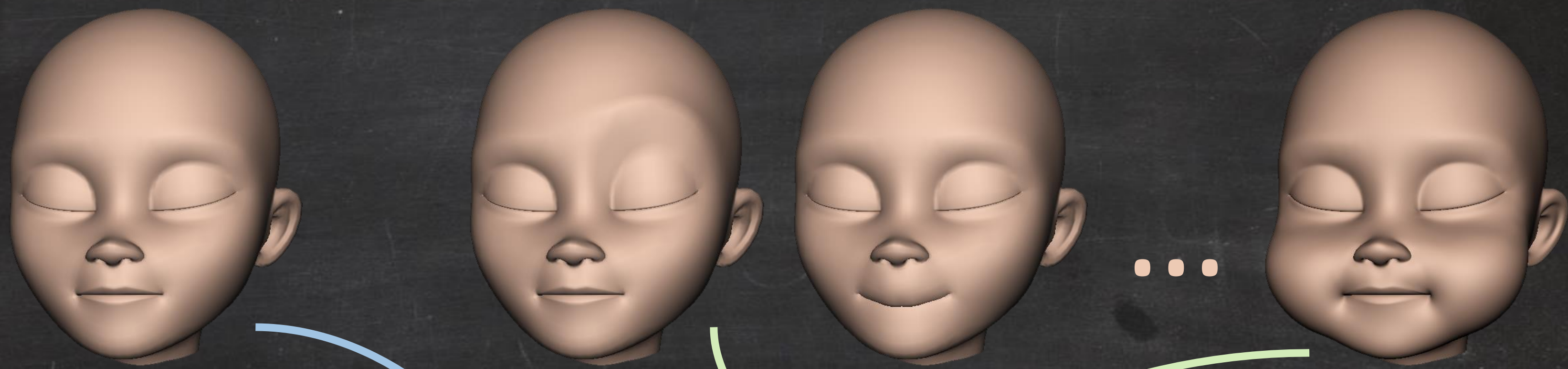
BlendShape

$$f = b_0 + \sum_{k=1}^{n} w_k (b_k - b_0)$$

$$f = b_0 + Bw$$

Character Rig from Mery Project

BlendShape



$$f = b_0 + \sum_{k=1}^{n} w_k (b_k - b_0)$$

$$f = b_0 + Bw$$

Character Rig from Mery Project

Comparison

PCA

- Orthogonal
- Lack the interpretability

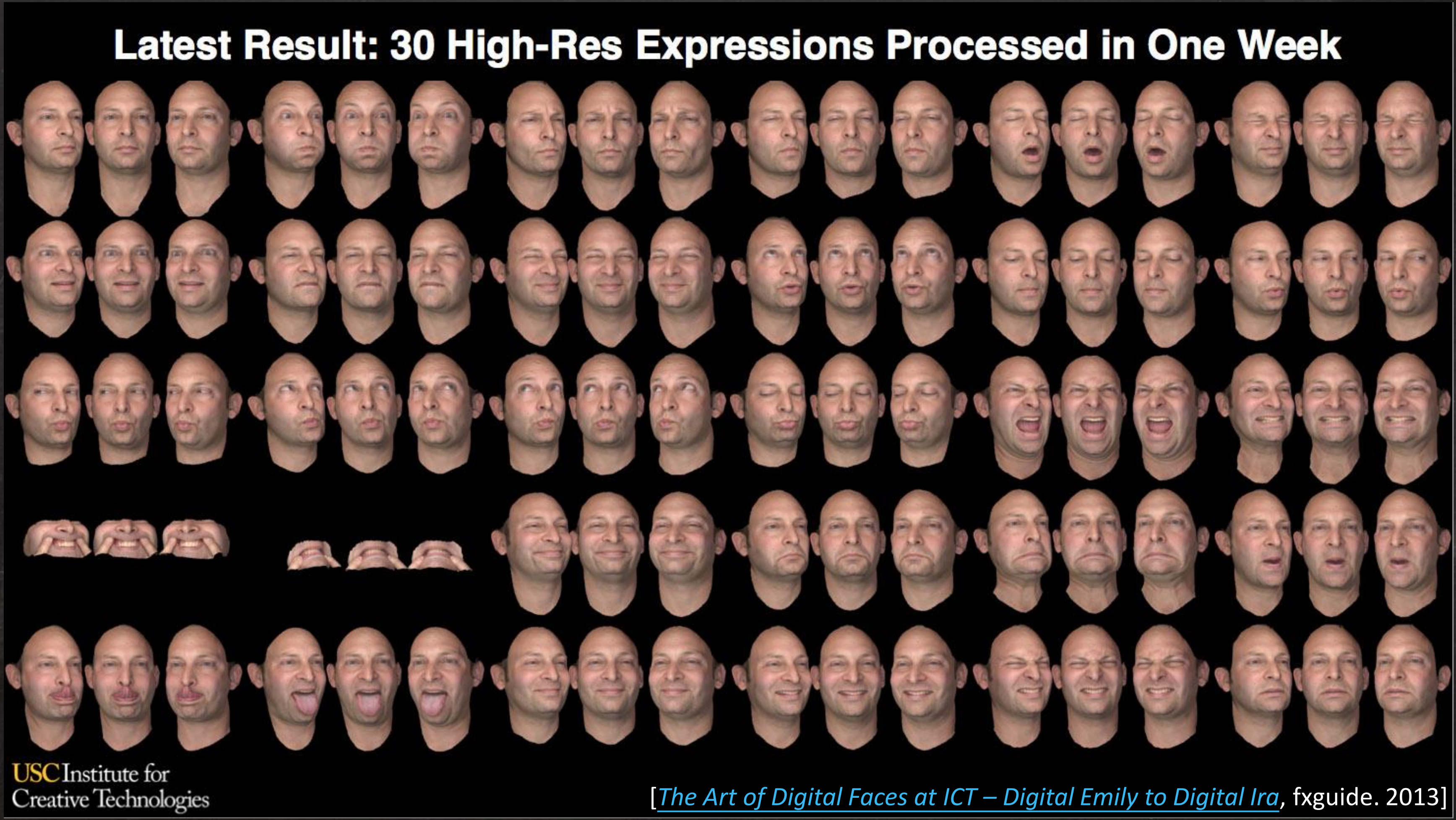
BlendShape

- Semantic parameterization
- Consistent appearance

- Lack of orthogonality
- Not unique:

$$f = B(RR^{-1})w$$

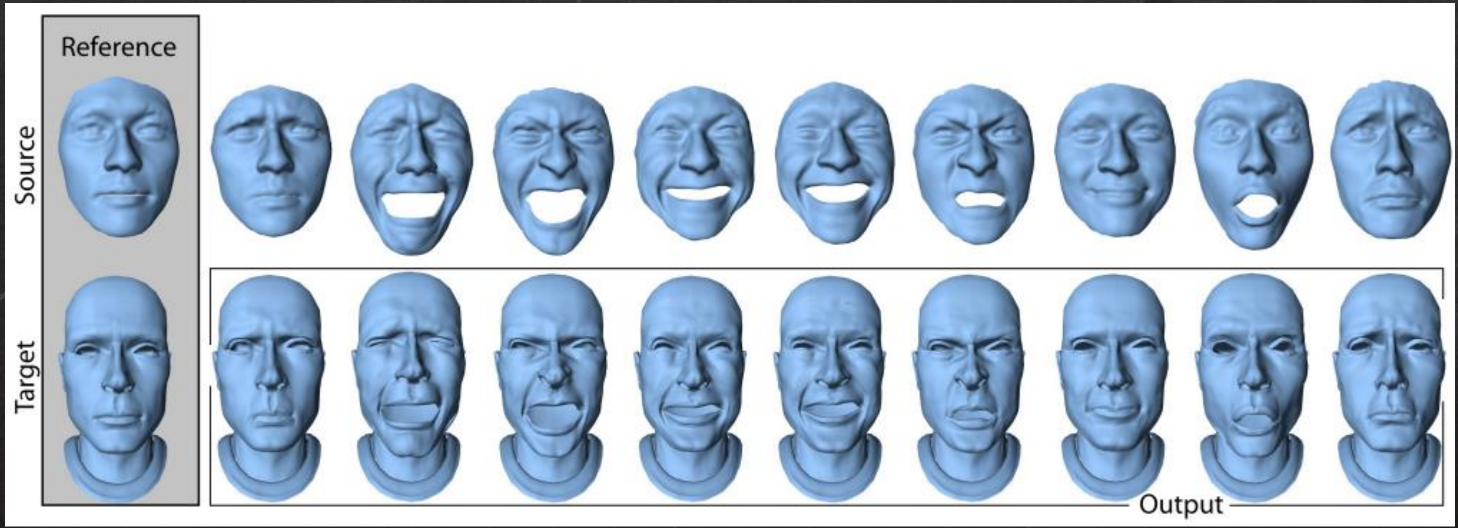
Facial Action Coding System (FACS)



[*The Art of Digital Faces at ICT – Digital Emily to Digital Ira*, fxguide. 2013]

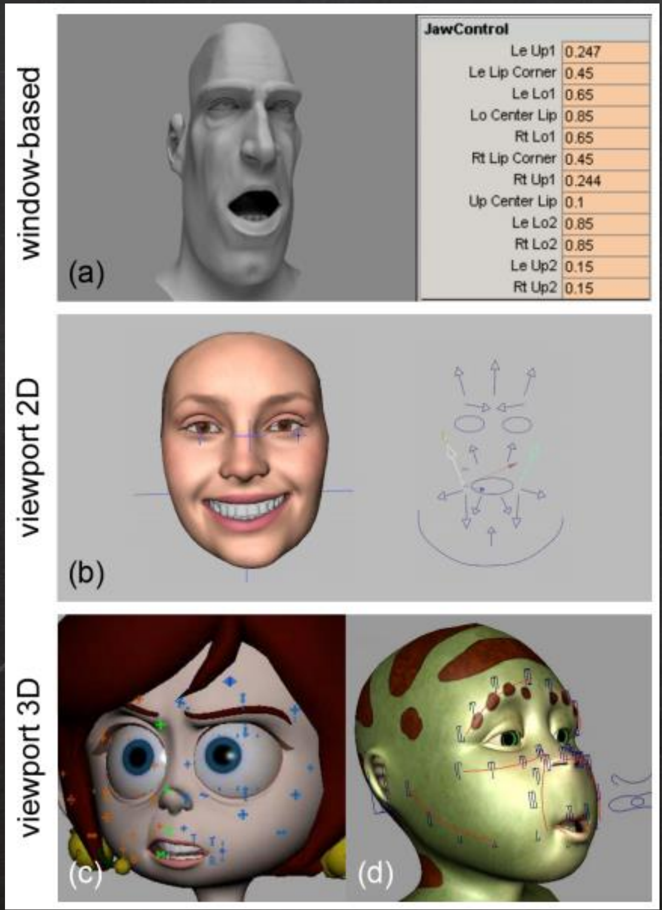
Practical Issues

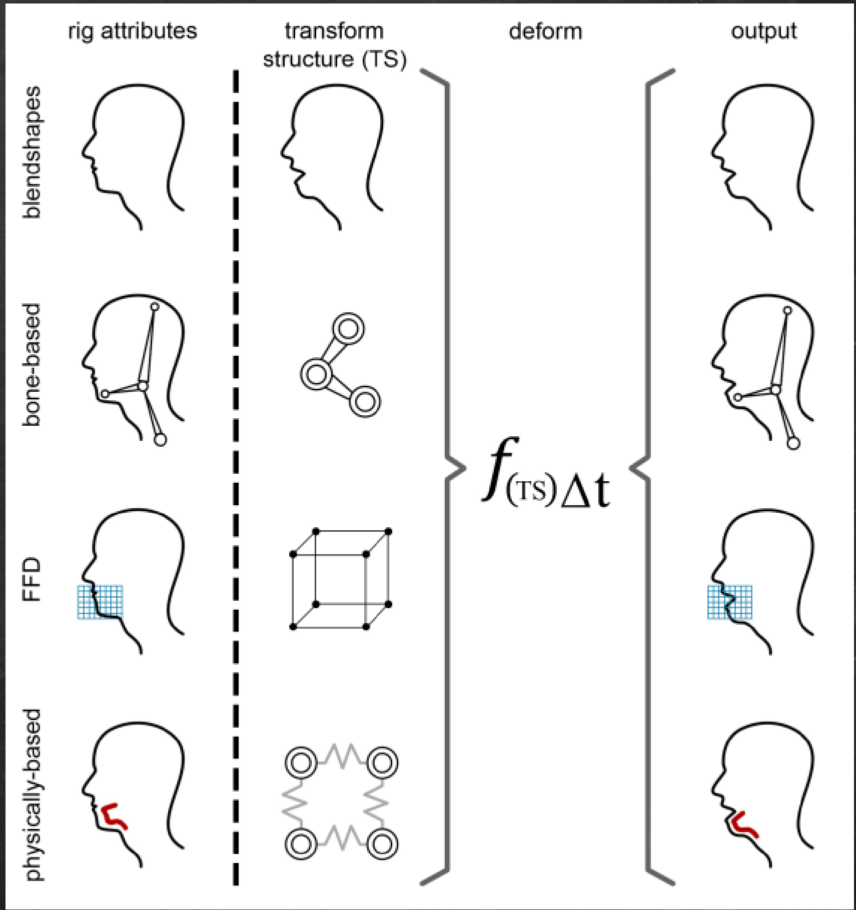
- How to compress BlendShape data?
- Expression transfer between multiple characters
  - Use deformation transfer for BlendShape targets



[Sumner and Popović, SIG'04]

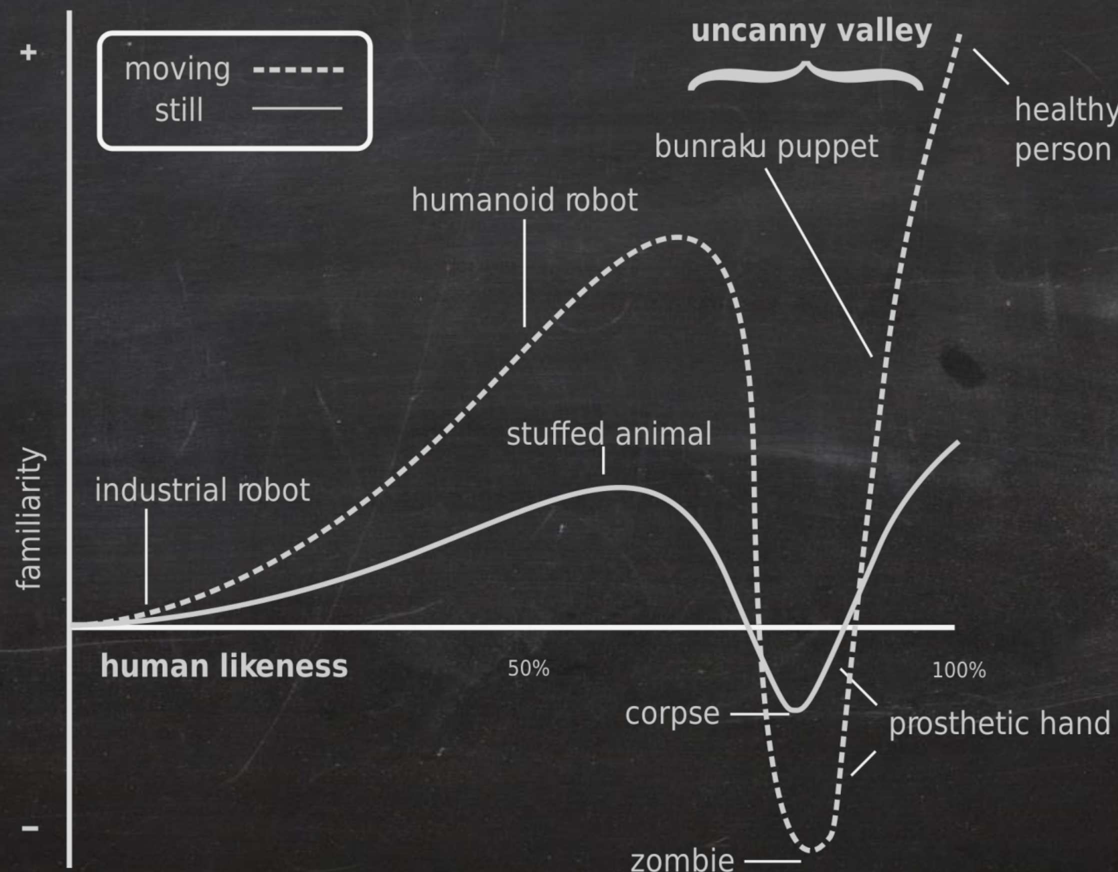
Facial Rigging





[Orvalho et al., EG'12]

Uncanny Valley

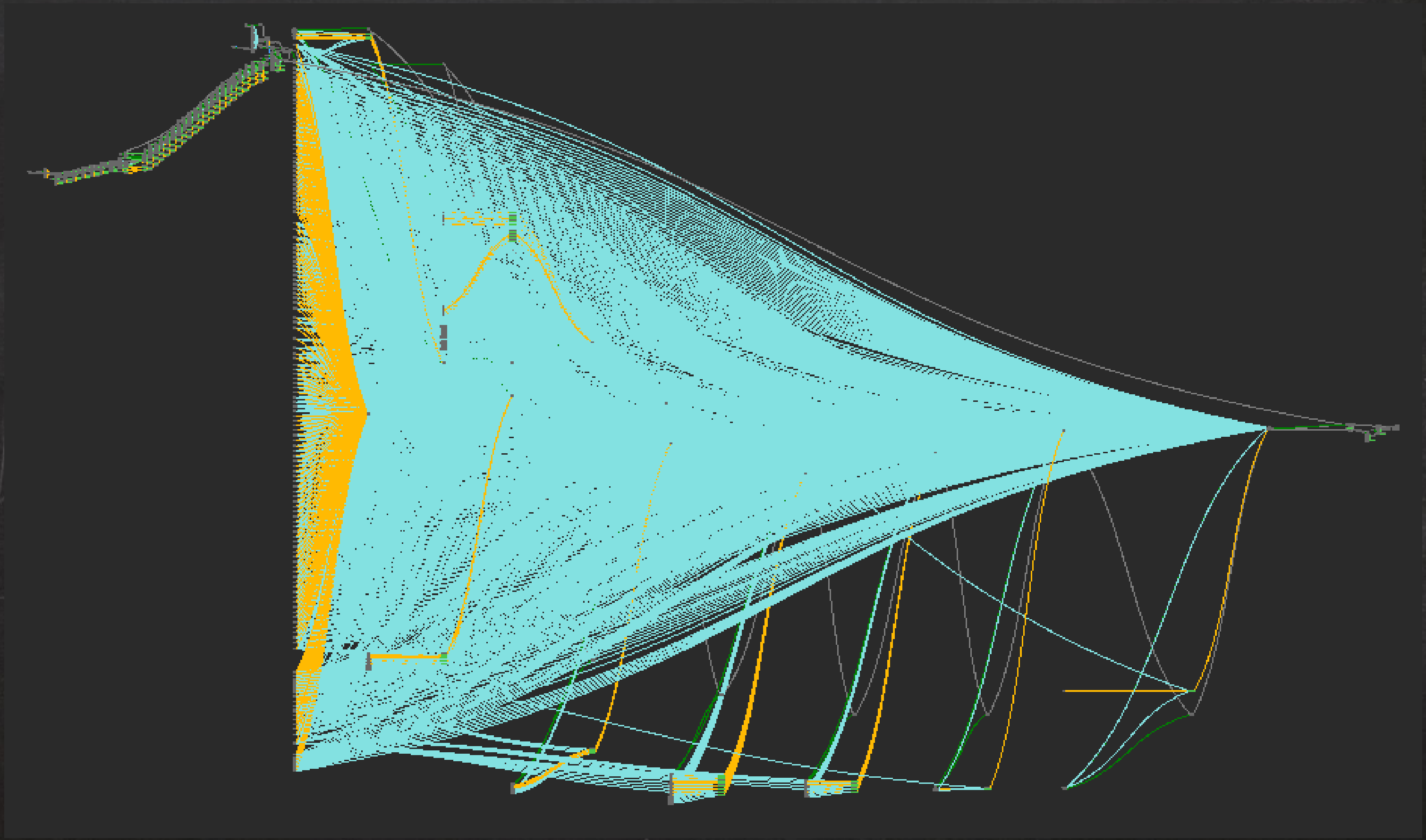


https://en.wikipedia.org/wiki/File:Mori_Uncanny_Valley.svg

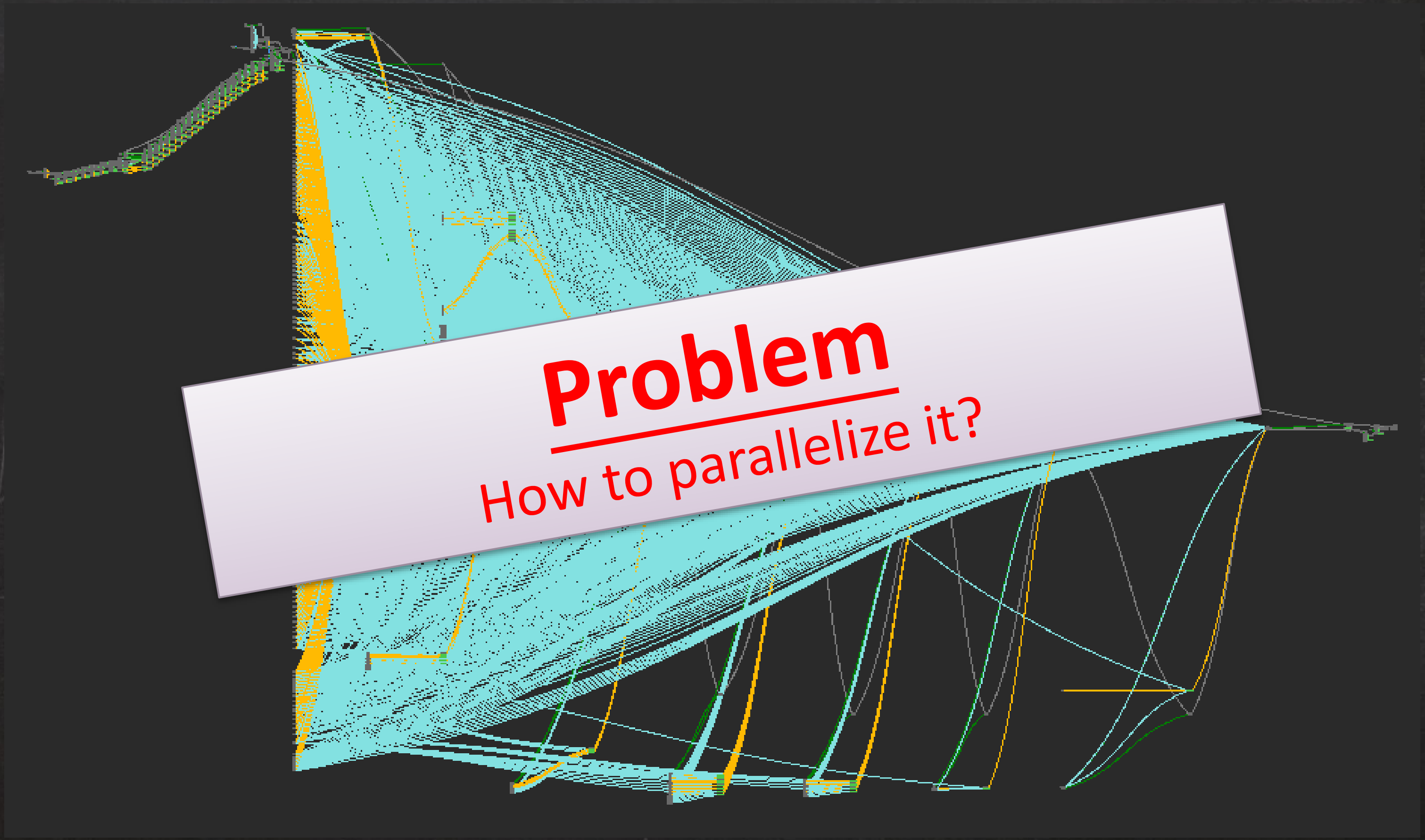
Practical Issues

- How to provide intuitive controls?
  - Too many => hard to manipulate
  - Not enough => can't get enough animation details
- In node-based framework, computation = graph evaluation
  - How do we separate the evaluation graph for parallelism?

Parallel Graph Evaluation



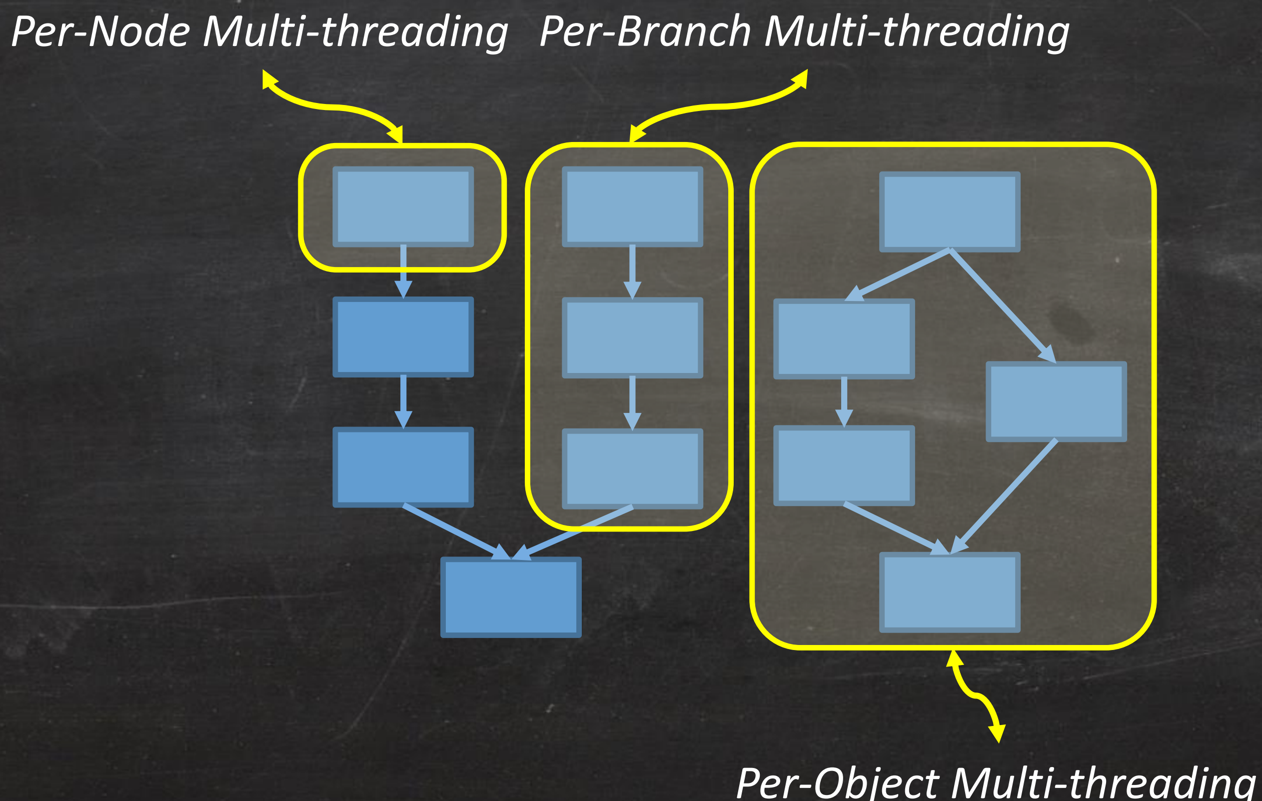
Parallel Graph Evaluation



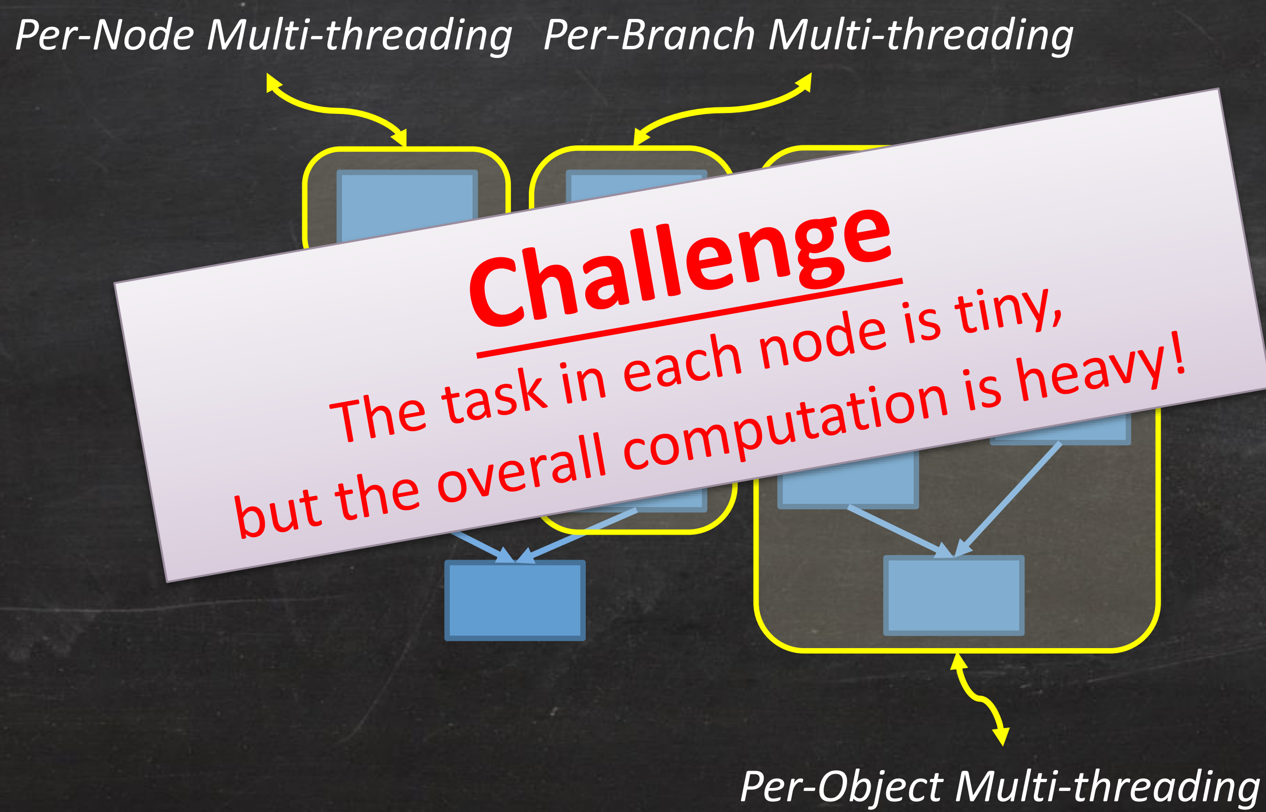
Parallel Graph Evaluation (Cont’d)

- Parallelization is **NOT** just about using TBB or CUDA
- Graph analysis is a key for performance gain
  - But the graph evaluation routine in Maya is a black box!!
- Numerical issue
  - Consistency between serial and parallel implementation
    - Due to rounding error and truncation of floating point
  - Deterministic algorithm?

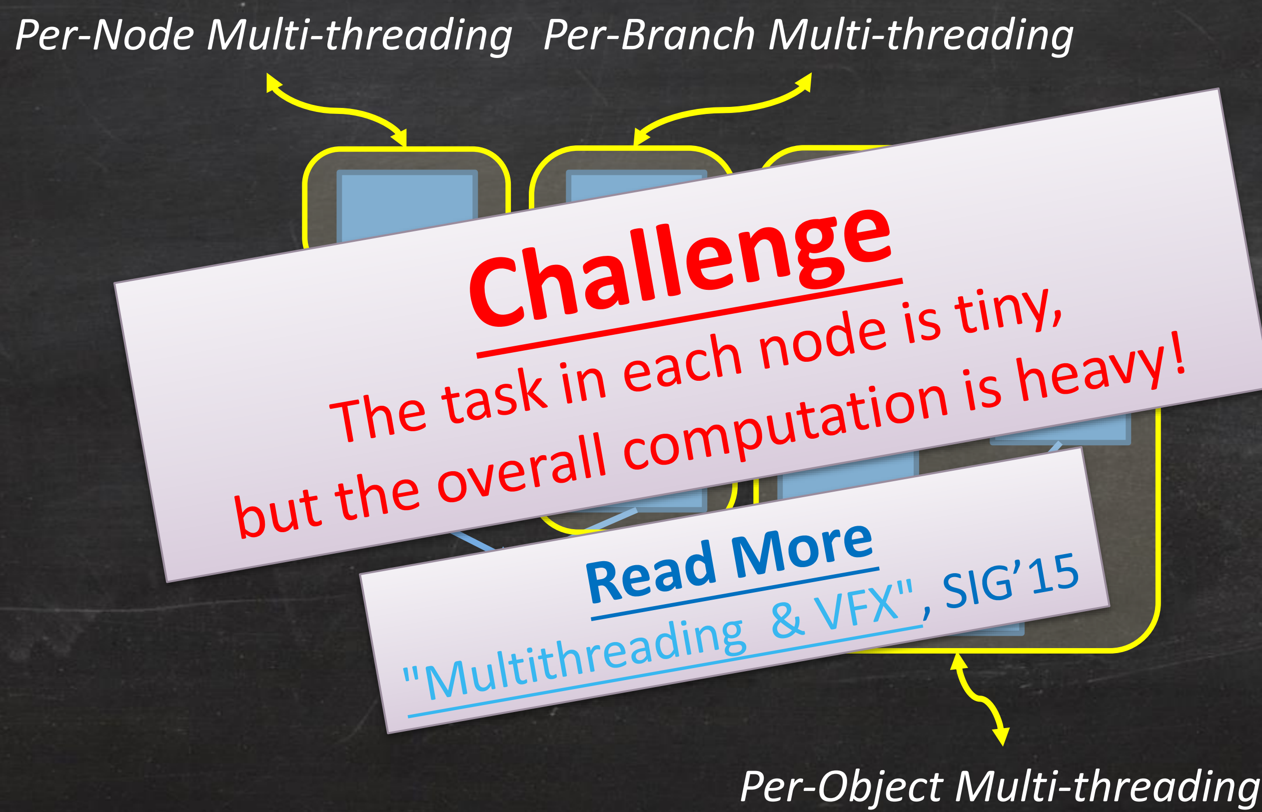
Multi-threading in Node-Based Architecture



Multi-threading in Node-Based Architecture



Multi-threading in Node-Based Architecture



Physically Based Animation

Cloth

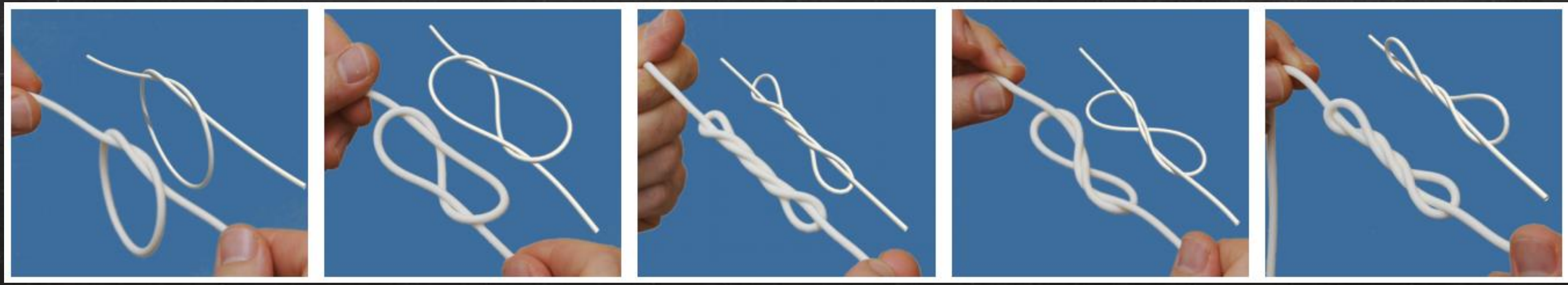
[Baraff and Witkin. SIG'98]

[Tamstorf et al., SIGA'15]

Hair

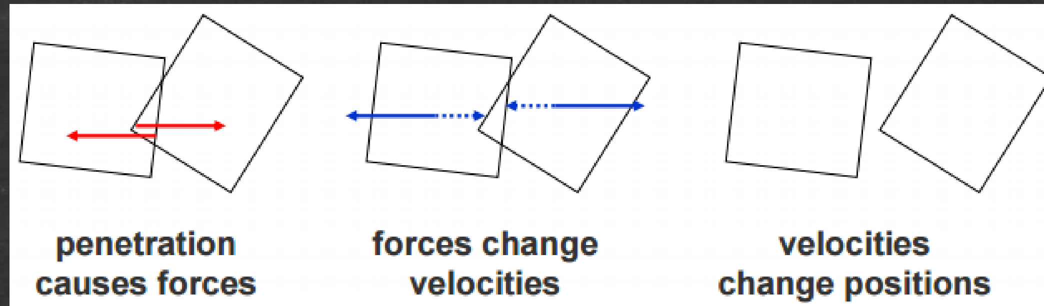
[Iben et al., SCA’13]

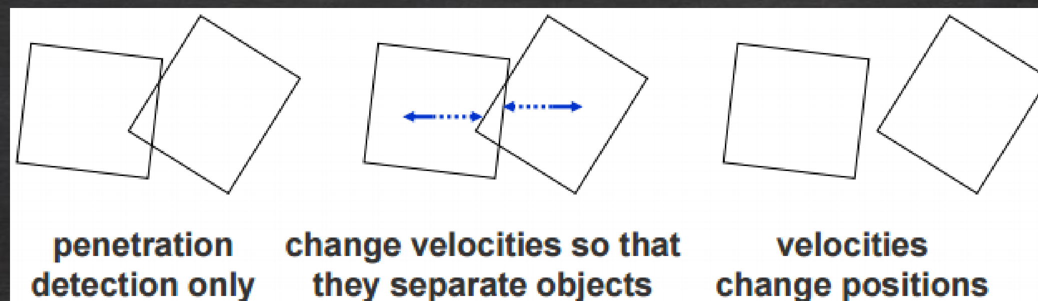
[Selle et al., SIG’08]



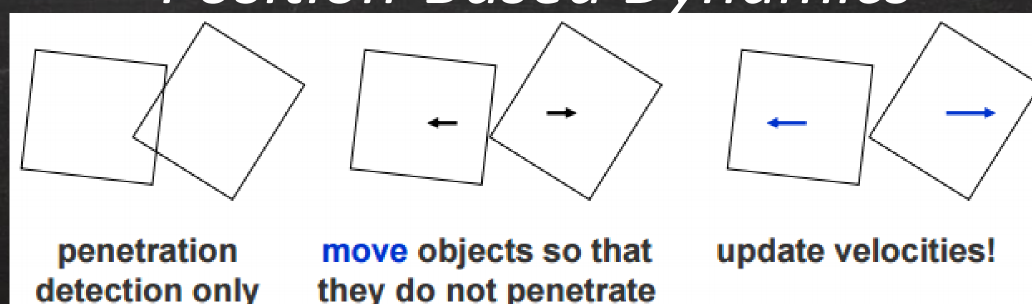
[Bergou et al., SIG’08]

Force-Based Dynamics





Position-Based Dynamics



[Figures from Müller et al., *"Position Based Dynamics"*, VRIPHYS'06]

Comparison

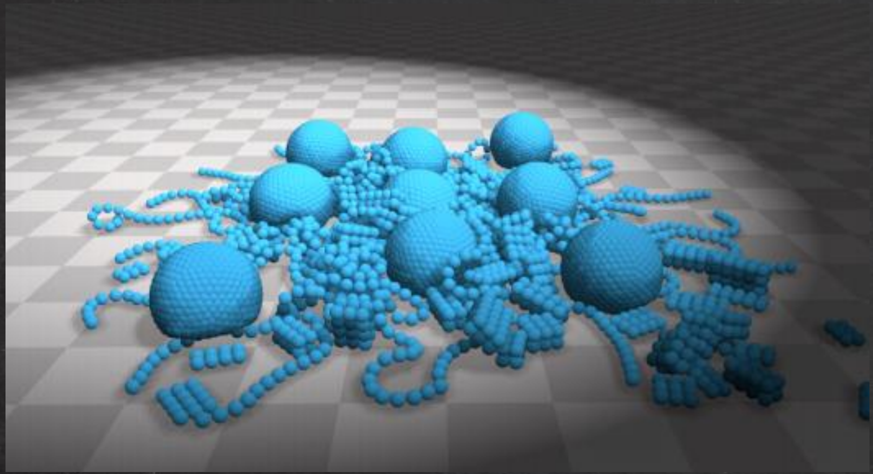
Force-Based

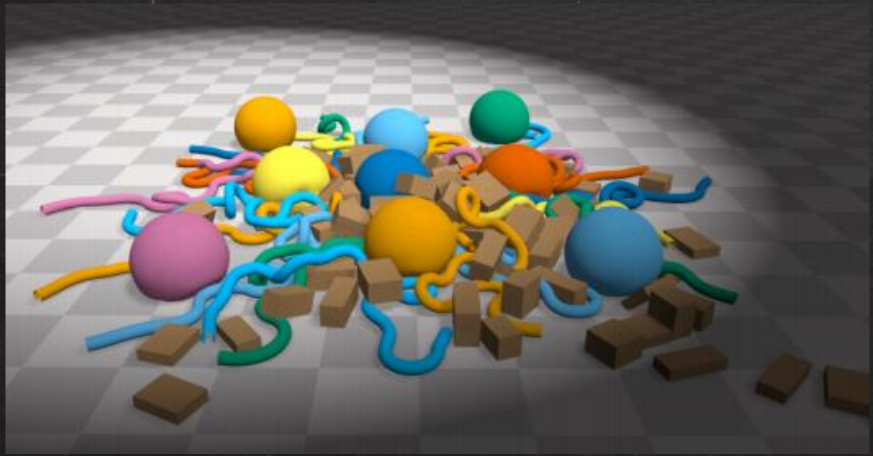
- ✓ Physically accurate
  - Newton second law
  - Navier-Stokes
  - …, etc.
- Explicit integration
  - Not stable for stiff system
  - Overshooting
- Implicit integration
  - Computationally expensive
  - Numerical damping

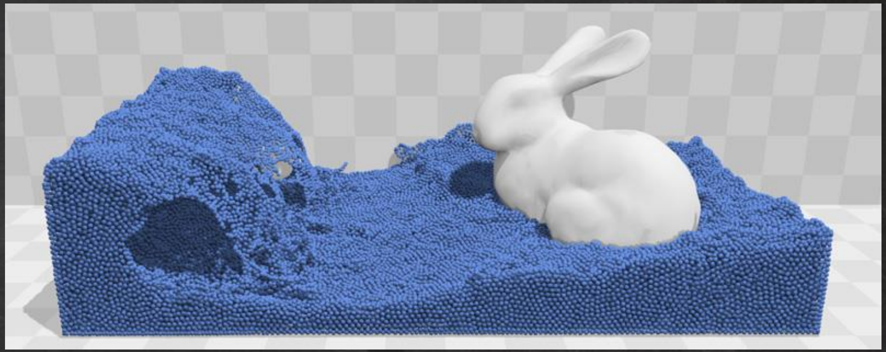
Position-Based

- ✓ Fast
- ✓ Unconditionally stable
- ✓ Controllable
- Less physically accurate
- Need to explore new ways to update velocity

Unified Particle Physics







[Macklin et al., SIG'14]

References

- Quaternions, Ken Shoemake.
- Understanding Rotations, Jim Van Verth.
- On Linear Variational Surface Deformation Methods, Mario Botsch, Olga Sorkine-Hornung.
- Skinning: Real-time Shape Deformation, SIG'14.
- *Laplace-Beltrami: The Swiss Army Knife of Geometry Processing*, SGP'14.