# Shape Deformation

# Representing fundamental deformations

# Affine/Linear Deformations

Most used operations $p' = f(p)$

can be represented by a matrix $\rightarrow$ can be sent as uniform parameter to the shader.

- In standard 3D coordinates $p = (x, y, z)$

$p' = \mathrm{L}\, p + t$

$\mathrm{L}$: linear component ($3 \times 3$ matrix)

$t$: translation

- In homogeneous coordinates $p = (x, y, z, 1)$

$p' = \mathrm{A}\, p$

$\mathrm{A}$: $4 \times 4$ matrix

$$\mathrm{A} = \left(\begin{array}{c|c} \mathrm{L} & t \\ \hline 0 & 1 \end{array}\right) = \left(\begin{array}{ccc|c} \mathrm{L}_{xx} & \mathrm{L}_{xy} & \mathrm{L}_{xz} & t_x \\ \mathrm{L}_{yx} & \mathrm{L}_{yy} & \mathrm{L}_{yz} & t_y \\ \mathrm{L}_{zx} & \mathrm{L}_{zy} & \mathrm{L}_{zz} & t_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

Properties:

$\det(\mathrm{A})$: Change of volume when applying $A$ to a shape

$\mathrm{A}^T\, \mathrm{A} = 1 \Rightarrow$ Isometry

# Affine/Linear Deformations

## Translation

$$t(p) = (x + t_x, y + t_y, z + t_z)$$

$$\mathrm{T} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

## Scaling

$$s(p) = (s_x\, x, s_y\, y, s_z\, z)$$

$$\mathrm{S} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

## Rotation

Several possible representations (see later)

*Note: Isometry*

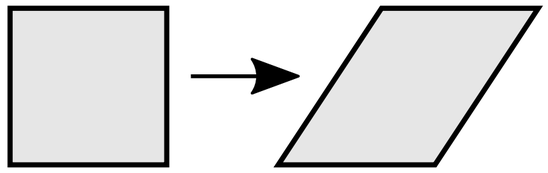
## Shearing

$$sh_{xy}(p) = (x + \lambda y, y, z)$$
$$sh_{xz}(p) = (x + \lambda z, y, z)$$
$$sh_{yx}(p) = (x, y + \lambda x, z)$$
...

$$\mathrm{Sh}_{xy} = \begin{pmatrix} 1 & \lambda & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \mathrm{Sh}_{xz} = \begin{pmatrix} 1 & 0 & \lambda & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \mathrm{Sh}_{yx} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \lambda & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \ldots$$

Usually avoided in Graphics

*Note:* $\det(\mathrm{Sh}) = 1 \rightarrow$ *constant volume (but no isometry)*



# Rotations

3D rotations have 3 dof, No unique representation

### Matrix

$R^T R = \mathrm{I}$
$\det(R) = 1$

(+) Computationaly convenient
(-) Non-explicit dof, redundancies

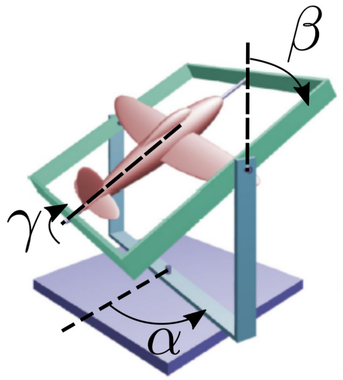
### Euler Angles

3 angles: $(\alpha, \beta, \gamma)$

Composition of rotation around basic axes

Not unique (x-y-z, y-z-x, x-y-x', x-z-x', ...)

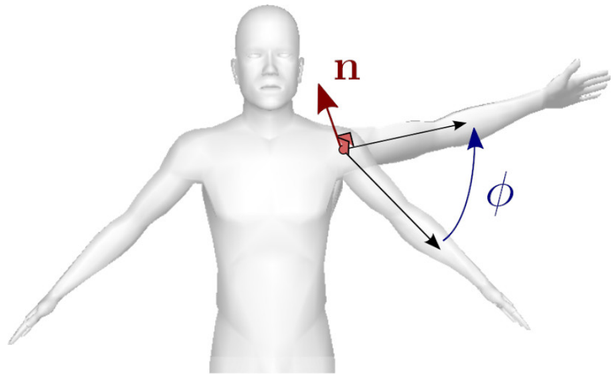
(+) Meaningfull parameters
(-) Gimbal-lock




### Axis angle

$(\mathbf{n}, \theta)$

(+) Meaningfull parameters
(-) No direct composition




### Quaternion

$q = (x, y, z, w)$
$= \left(\mathbf{n}\, \cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right)\right)$

(+) Composition and interpolation
(-) Less intuitive components

# Rotation: Focus Euler angles

Three consecutive rotations along fixed axis along $(x, y, z)$ coordinates.

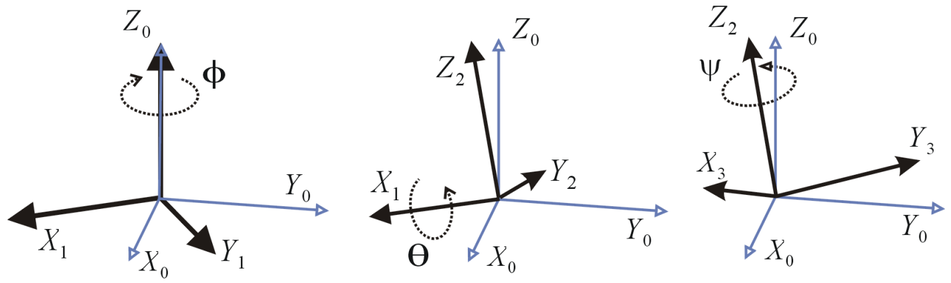
Can use basic rotation matrices composition

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin\alpha & \cos(\alpha) \end{pmatrix} \quad R_y = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix} \quad R_z = \begin{pmatrix} \cos(\gamma) & \sin(\gamma) & 0 \\ -\sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Multiple Euler angles conventions

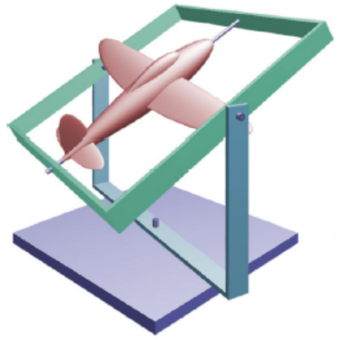
- *Proper Euler : z-x-z', x-y-x', y-z-y', ...*
- *Trait-Bryan : x-y-z (global), ...*

*Take care when exporting/importing/parsing between Softwares*




**Pro**

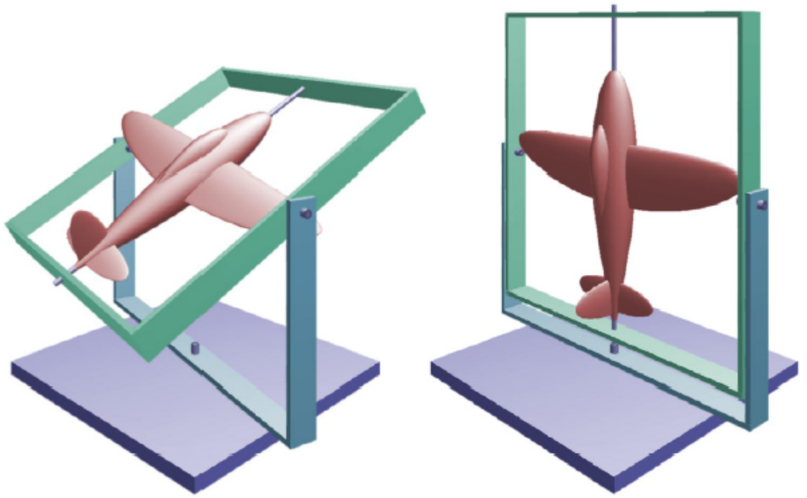
- Combination of rotation around known axis
- Comprehensive parameters (3 dof)
- Animators can interact with angular curves
- *Widely used in robotics*



# Rotation: Focus Euler angles

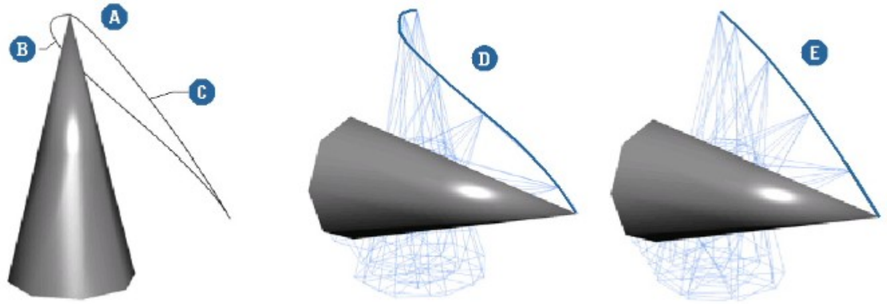
**Limitations of Euler Angle**

- *Gimbal Lock* when composing b/w some rotations



http://www.fho-emden.de/~hoffmann/gimbal09082002.pdf

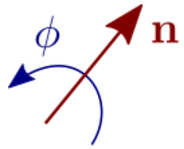
- Interpolation of 3 angles possible but can lead to non-intuitive trajectory.



# Rotation: Focus Axis Angle

Any 3D rotation can be represented by

- A unit axis $n$
- An angle $\theta$

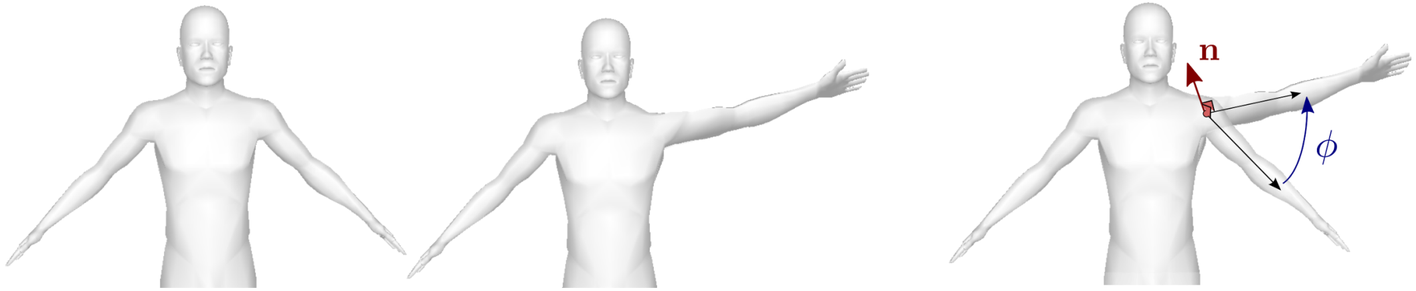



**Pro.**

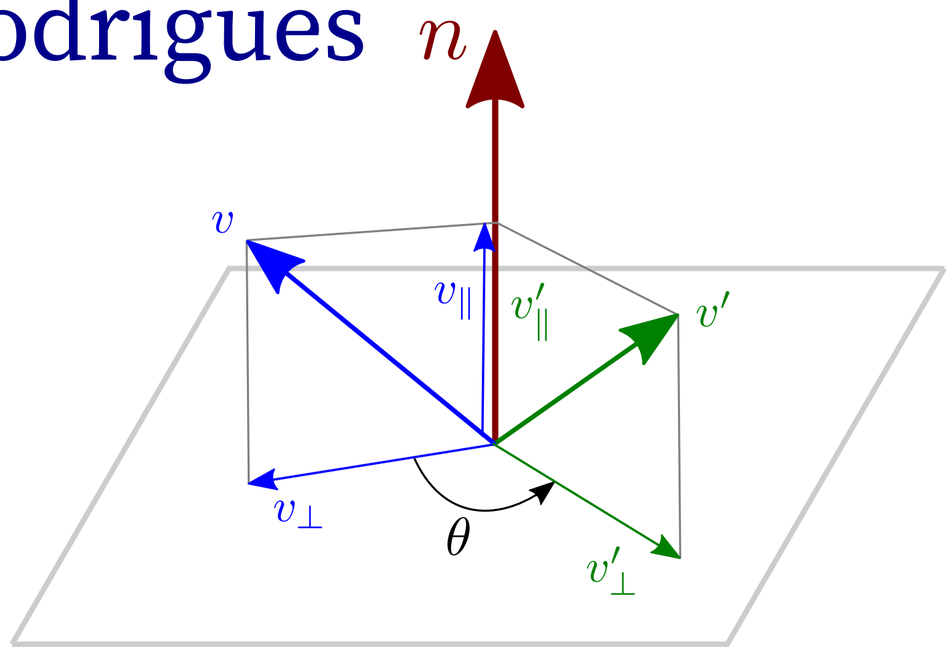
- Concise representation: 3 DOF
- Meaningfull parameters
- Describes well the rotation between two vectors

Ex. Rotation between $u_1$ and $u_2$

- Axis of rotation $n = (u_1 \times u_2)/\|u_1 \times u_2\|$
- Angle of rotation $\theta = \text{acos}(u_1 \cdot u_2)$
- Twist around $u_2$ can be arbitrarily chosen



# Rotation: Focus Axis Angle - Rodrigues




Applying a rotation $(n, \theta)$ to a vector $v$

$$v = v_\parallel + v_\perp$$
$$v' = v'_\parallel + v'_\perp$$
$$\Rightarrow v' = v_\parallel + (\cos(\theta)\, v_\perp + \sin(\theta)\, n \times v_\perp)$$

$$v_\parallel = (v \cdot n)\, n$$
$$v_\perp = v - (v \cdot n)\, n$$

$$v' = (v \cdot n)\, n + \cos(\theta)(v - (v \cdot n)\, n) + \sin(\theta)\, n \times (v - (v \cdot n)\, n)$$

$$\Rightarrow v' = \cos(\theta)\, v + \sin(\theta)\, n \times v + (1 - \cos(\theta))\, (v \cdot n)\, n$$

**Rodrigues' rotation Formula**

# Rotation: Focus Axis Angle as Matrix

$$v' = \cos(\theta)\, v + \sin(\theta)\, n \times v + (1 - \cos(\theta))\, (v \cdot n)\, n = R(n, \theta)\, v$$

$$v' = \cos(\theta) v + \sin(\theta) \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \times v + (1 - \cos(\theta)) \begin{pmatrix} n_x & n_y & n_z \end{pmatrix} v \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}$$

$$v' = \cos(\theta)\, v + \sin(\theta) \underbrace{\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}}_{\mathrm{K}} v + (1 - \cos(\theta)) \underbrace{\begin{pmatrix} n_x^2 & n_x\, n_y & n_x\, n_z \\ n_x\, n_y & n_y^2 & n_y\, n_z \\ n_x\, n_z & n_y\, n_z & n_z^2 \end{pmatrix}}_{\mathrm{K}^2} v$$

$$v' = \begin{pmatrix} \cos(\theta) + n_x^2(1 - \cos(\theta)) & n_x n_y (1 - \cos(\theta)) - n_z \sin(\theta) & n_x n_z (1 - \cos(\theta)) + n_y \sin(\theta) \\ n_x n_y (1 - \cos(\theta)) + n_z \sin(\theta) & \cos(\theta) + n_y^2 (1 - \cos(\theta)) & n_y n_z (1 - \cos(\theta)) - n_x \sin(\theta) \\ n_x n_z (1 - \cos(\theta)) - n_y \sin(\theta) & n_y n_z (1 - \cos(\theta)) + n_x \sin(\theta) & \cos(\theta) + n_z^2 (1 - \cos(\theta)) \end{pmatrix} v$$

# Rotation: Focus Axis Angle - Summary

Given a rotation $(n, \theta)$ - Corresponding rotation matrix

$$R(n,\theta) = \mathrm{I} + \sin(\theta)\,\mathrm{K} + (1-\cos(\theta))\,\mathrm{K}^2$$

$$R(n,\theta) = \begin{pmatrix} \cos(\theta) + n_x^2(1-\cos(\theta)) & n_x n_y(1-\cos(\theta)) - n_z\sin(\theta) & n_x n_z(1-\cos(\theta)) + n_y\sin(\theta) \\ n_x n_y(1-\cos(\theta)) + n_z\sin(\theta) & \cos(\theta) + n_y^2(1-\cos(\theta)) & n_y n_z(1-\cos(\theta)) - n_x\sin(\theta) \\ n_x n_z(1-\cos(\theta)) - n_y\sin(\theta) & n_y n_z(1-\cos(\theta)) + n_x\sin(\theta) & \cos(\theta) + n_z^2(1-\cos(\theta)) \end{pmatrix}$$

**Pro**

- Concise and general representation
- Expressive parameters in 3D space (axe, angles)
- Efficient rotation and correspondance with matrix

**Cons**

- No simple composition expression between two rotations.
- No direct interpolation

(well handled by quaternion representation)

# Rotation: Focus Axis Angle - Exponential Map

Matrix $\mathrm{K}$ is such that $\mathrm{K}\,v = n \times v$.

$\mathrm{K} = \begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}$ is skew-antisymetric. And $K^3 = -K$

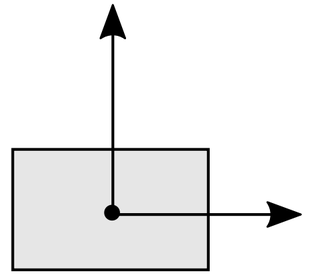
The rotation matrix can be obtained as the matrix exponential

$$\begin{aligned}
\mathrm{R} &= \exp(\theta\,\mathrm{K}) = \sum_k \frac{(\theta\,\mathrm{K})^k}{k!} \\
&= I + \theta\mathrm{K} + \frac{1}{2!}(\theta\mathrm{K})^2 + \frac{1}{3!}(\theta\mathrm{K})^3 + \dots \\
&= I + \underbrace{\left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \dots\right)}_{\sin(\theta)}\mathrm{K} + \underbrace{\left(\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \frac{\theta^6}{6!} - \dots\right)}_{1-\cos(\theta)}\mathrm{K}^2
\end{aligned}$$

$= I + \sin(\theta)\,\mathrm{K} + (1 - \cos(\theta))\,\mathrm{K}^2$ Rodrigues formula

Allows for matrix interpolation with varying $\theta$ (exponential map).

# Cautions with transformations order

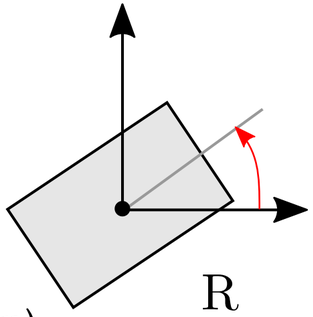


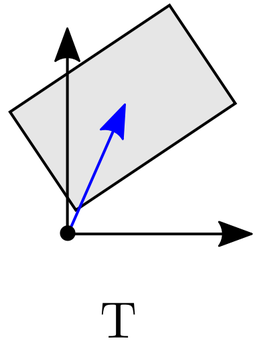
Take care, order of operation does matter !

Rotation $r$, Translation $t$: $r \circ t \neq t \circ r \quad \Rightarrow \quad \mathrm{M}_1 = \mathrm{T}\,\mathrm{R} \neq \mathrm{R}\,\mathrm{T} = \mathrm{M}_2$

Take care (2): transformation matrices applied to coordinates from right to left.

$$\mathrm{M}_1 = \mathrm{TR} = \left(\begin{array}{c|c} 1 & t \\ \hline 0 & 1 \end{array}\right) \left(\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array}\right) = \left(\begin{array}{c|c} R & t \\ \hline 0 & 1 \end{array}\right)$$

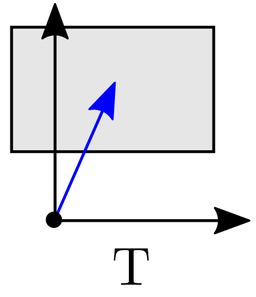


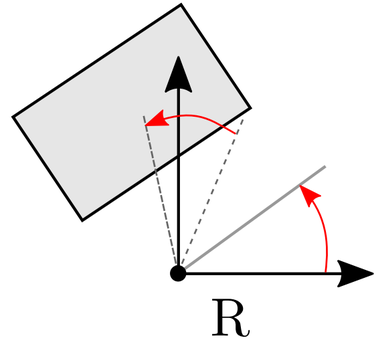




The object rotates around its "local origin".
Translates in the global reference frame.

*ex. Apply a rotation centered around a point of an object (good for interactive transformation).*

$$\mathrm{M}_2 = \mathrm{RT} = \left(\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array}\right) \left(\begin{array}{c|c} 1 & t \\ \hline 0 & 1 \end{array}\right) = \left(\begin{array}{c|c} R & \mathrm{R}\,t \\ \hline 0 & 1 \end{array}\right)$$




The object rotates around its initial origin (before translation).
Translates in the rotated frame.

*ex. Apply a translation in the local rotated reference (good for hierarchical organisation).*

*Warning: Some libraries (old OpenGL, Three.js) apply transformation from "left to right" using transposed matrix multiplication*

# Common case

Rotation of an object around an axis passing by a point $p_0$

1- Translate the object to center $p_0$ at the origin $\mathrm{M}_1 = \left(\begin{array}{c|c} 1 & -p_0 \\ \hline 0 & 1 \end{array}\right)$

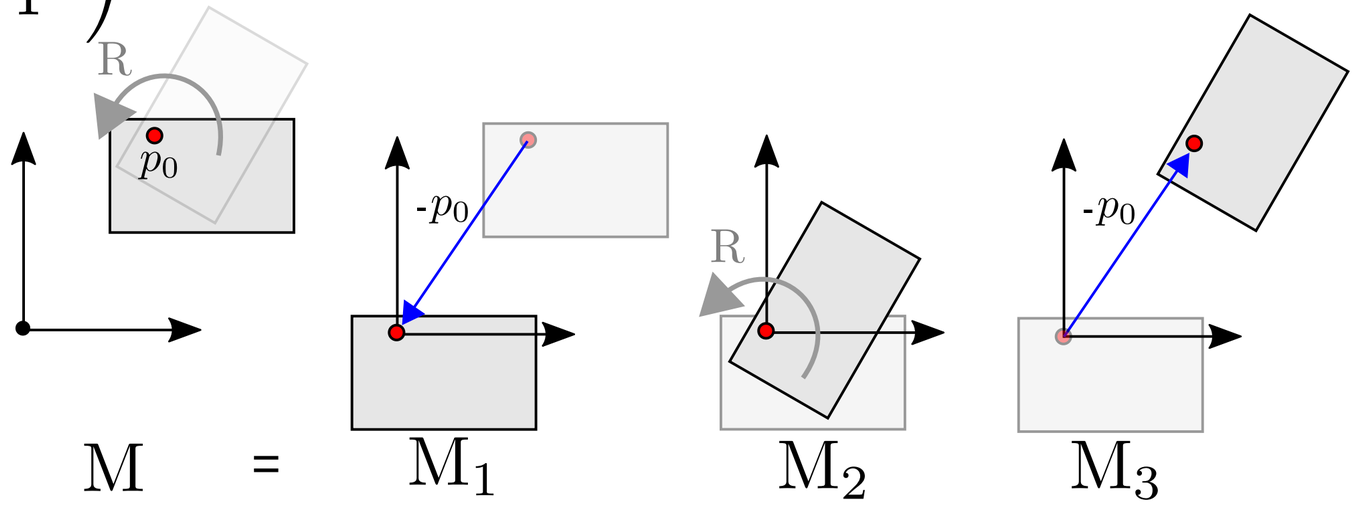
2- Rotate the object $\mathrm{M}_2 = \left(\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array}\right)$

3- Translate back the object $\mathrm{M}_3 = \left(\begin{array}{c|c} 1 & p_0 \\ \hline 0 & 1 \end{array}\right)$

Global transformation:

$\mathrm{M} = \mathrm{M}_3\,\mathrm{M}_2\,\mathrm{M}_1$

$= \left(\begin{array}{c|c} R & -R\,p_0 + p_0 \\ \hline 0 & 1 \end{array}\right)$



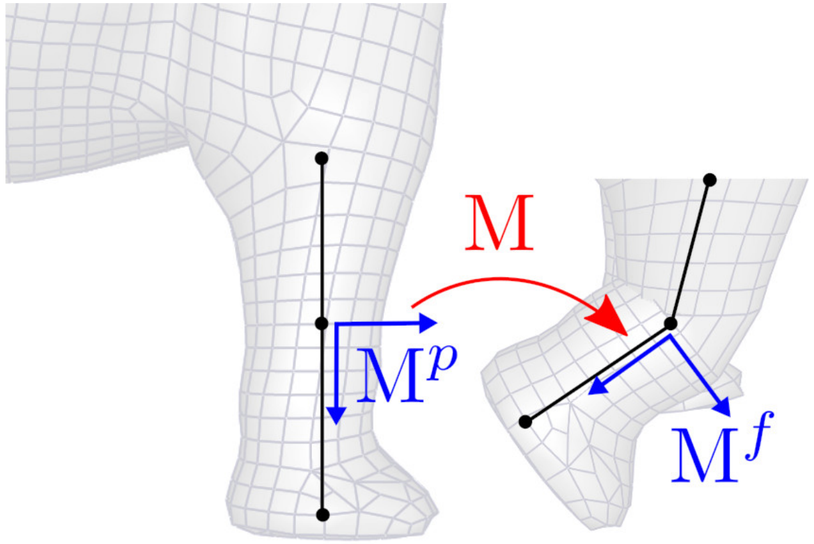
# Hierarchical transformation

Transformation expressed with respect to a parent frame (local coordinates).

$\mathrm{M} = (R, t)$: transformation expressed in local coordinates

$\mathrm{M}^p = (R^p, t^p)$: parent frame expressed

$\mathrm{M}^f = (R^f, t^f)$: current frame (after transformation)




Hierarchical relation:

$$\mathrm{M}_f = \mathrm{M}\,\mathrm{M}^p \text{ or } \mathrm{M} = \mathrm{M}_f\,(\mathrm{M}^p)^{-1}$$

Expression of $\mathrm{M}^f$

$$\mathrm{M}_f = \mathrm{M}\,\mathrm{M}^p$$

$$\mathrm{M}_f = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R^p & t^p \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R\,R^p & R\,t^p + t \\ 0 & 1 \end{pmatrix}$$

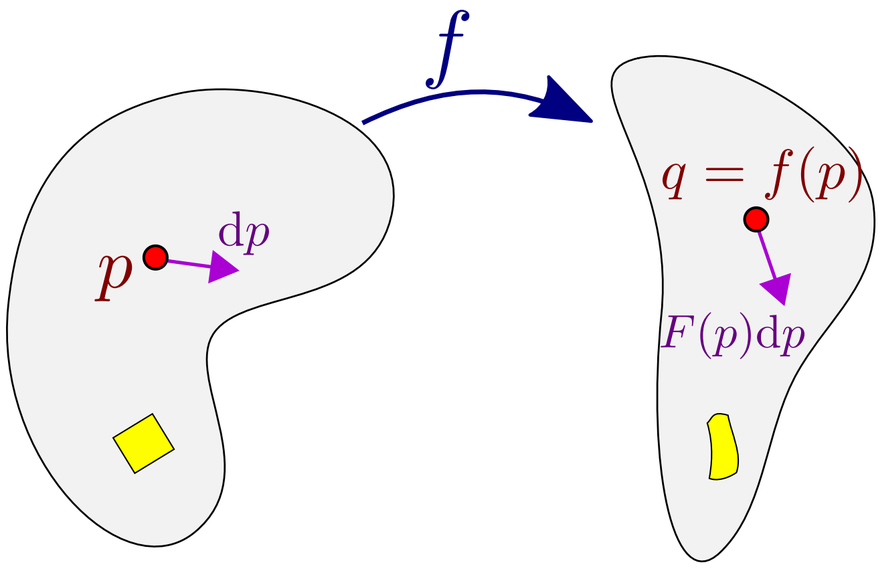
- $R^f = R\,R^p$
- $t^f = R\,t^p + t$

# Non linear deformation

In general: $q = f(p)$

$(q_x, q_y, q_z) = (f_x(p_x, p_y, p_z),\ f_y(p_x, p_y, p_z),\ f_z(p_x, p_y, p_z))$

or $q_i = f_i(p_0, p_1, p_2)$ using indices




Considering infinitesimal displacements

$$\mathrm{d}q_i = \sum_j \underbrace{\frac{\partial f_i}{\partial p_j}}_{\mathrm{F}_{ij}} \mathrm{d}p_j \qquad \Rightarrow \mathrm{d}q = \mathrm{F}\,\mathrm{d}p$$

$$\mathrm{F} = \begin{pmatrix} \frac{\partial f_x}{\partial x} & \frac{\partial f_x}{\partial y} & \frac{\partial f_x}{\partial z} \\ \frac{\partial f_y}{\partial x} & \frac{\partial f_y}{\partial y} & \frac{\partial f_y}{\partial z} \\ \frac{\partial f_z}{\partial x} & \frac{\partial f_z}{\partial y} & \frac{\partial f_z}{\partial z} \end{pmatrix}$$

F: **Deformation gradient** (Jacobian of $f$)

Principle measure of deformation

*called material deformation gradient tensor in mechanics*

Small displacements vector $\mathrm{d}p$ at point $p$ are transformed into $\mathrm{F}(p)\,\mathrm{d}p$.

Special cases

- $\det(\mathrm{F}(p))$ measure of the local change of volume at point $p$.
- $\mathrm{F}^{-1}$ called *spatial* deformation gradient (or pull back).
- $f$ translation: $\mathrm{F} = \mathrm{I}$
- $f$ linear transformation of matrix A: $\mathrm{F} = \mathrm{A}$

# Interpolation between affine transform

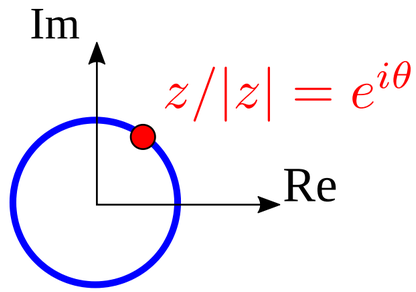
1- Reminder on Quaternion

2- Interpolation

# Intuition for quaternion

**Going back to 2D and complex space**

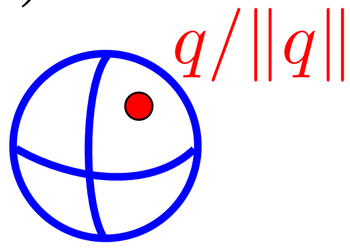
- 2D rotations have 1 dof. (rotation angle)
- 2D rotation represented as a point on a unit circle ($S^1$).
  - *1D structure embedded in a 2D space.*
  - *Complex space has algebraic structure adapted to model 2D rotation.*




rotations
sphere $S^2$

**Moving to 3D**

- 3D rotations have 3 dof.
  - *(point on a unit 3D sphere allows only 2 dof : not sufficient ex. orientation that can twist)*
  - point on a unit 4D sphere allows 3 dof
- 3D rotation are represented as **point on the unit sphere in 4D** ($S^3$).
  - *3D structure embedded in a 4D space.*
  - *Complex quaternion space has algebraic structure adapted to model 3D rotation.*




rotations
sphere $S^4$

# Big picture of quaternions

**Share similarities with 2D complex**

- Complex value object with 4 components $q = (x, y, z, w)$
- Unit quaternions represent rotations $q/\|q\|$.
- Quaternion product represent rotation composition $q_1\, q_2$

**Some differences**

- 3D vectors are points in pure quaternion subspace $q_v = (x, y, z, 0)$
- Applying quaternion to points $q_{v'} = q\, q_v\, q^\star$

**Advantage of quaternion**

- Interpolation works well in quaternion space (interpolation of points on sphere).

# Rotation representation: Quaternions

Quaternions: generalization of complex numbers.

$q = x\,\mathbf{i} + y\,\mathbf{j} + z\,\mathbf{k} + w$ $w$ real part, $(x, y, z)$ imaginary (or pure quaternion) part.

We write in short $q = (x, y, z, w)$

*(don't confound with 4D vectors in homogeneous coordinates)*

Properties of imaginary basis vectors

$$\begin{cases} \mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1 \\ \mathbf{ij} = -\mathbf{ji} = \mathbf{k} \\ \mathbf{jk} = -\mathbf{kj} = \mathbf{i} \\ \mathbf{ki} = -\mathbf{ik} = \mathbf{j} \\ \mathbf{ijk} = -1 \end{cases}$$

- Provides the algebraic properties

# Basics operations on quaternions

- Conjugated quaternion $q^\star = (-x, -y, -z, w)$

- Quaternion norm $\|q\| = \sqrt{q\,q^\star} = \sqrt{x^2 + y^2 + z^2 + w^2}$. Unit quaternion satisfies $\|q\| = 1$.

- Quaternion product

$$q_1\ q_2 = (x_1\,\mathbf{i} + y_1\,\mathbf{j} + z_1\,\mathbf{k} + w_1)\,(x_2\,\mathbf{i} + y_2\,\mathbf{j} + z_2\,\mathbf{k} + w_2) = \ldots$$

$$q_1\ q_2 = \begin{pmatrix} x_1 w_2 + w_1 x_2 + y_1 z_2 - z_1 y_2 \\ y_1 w_2 + w_1 y_2 + z_1 x_2 - x_1 z_2 \\ z_1 w_2 + w_1 z_2 + x_1 y_2 - y_1 x_2 \\ w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2 \end{pmatrix}$$

Note: Quaternion product is
- associative: $(q_1 q_2) q_3 = q_1 (q_2 q_3) = q_1\ q_2\ q_3$
- **non-commutative**: $q_1\ q_2 \neq q_2\ q_1$

Sometimes interesting to separate *real part* $w$ from *pure quaternion* part $s = (x, y, z)$.
- Shorthand vector form $q = (s, w)$
- Quaternion product in vector form $q_1\ q_2 = (s_1\,w_2 + s_2\,w_1 + s_1 \times s_2\ ,\ w_1\,w_2 - s_1 \cdot s_2)$

# Relation between quaternion and rotation

Consider

- a quaternion $q = (s, w)$ of unit norm $\|q\| = 1$.
- a vector $v = (v_x, v_y, v_z)$ assimilated to the pure quaternion $q_v = (v, 0) = (v_x, v_y, v_z, 0)$.

Then

- $q_{v'} = \mathcal{R}_q(v) = q\, q_v\, q^\star$ is a pure quaternion $q_{v'} = (v'_x, v'_y, v'_z, 0)$
- And $v' = (v'_x, v'_y, v'_z)$ is the rotation of vector $v$ around the axis $n = s/\|s\|$, with angle $2\,\mathrm{acos}(w)$.

*Demonstration*

$\mathcal{R}_q(v) = (s, w)\ (v, 0)\ (-s, w) = \ldots = ((w^2 - s^2)\, v + 2(s \cdot v)\, s + 2w\, (s \times v), 0)$

As $\|q\| = 1$, we can write $q = (s, w) = (n \sin(\phi), \cos(\phi))$, where $\|n\| = 1$

Then $\mathcal{R}_q(v) = (\underbrace{(\cos^2(\phi) - \sin^2(\phi))}_{\cos(2\phi)}\, v + \underbrace{2 \sin^2(\phi)}_{1-\cos(2\phi)} (n \cdot v)\, n + \underbrace{2\, \cos(\phi) \sin(\phi)}_{\sin(2\phi)}\, n \times v\,,\ 0)$

$\Rightarrow$ Rodrigues formula for axis $n$ and angle $2\phi$.

The unit quaternion $q = (n\, \sin(\theta/2), \cos(\theta/2))$ represents the rotation of angle $\theta$ around the axis $n$.

# Composition of rotations

Consider two rotation $(R_1, R_2)$ associated to their unit quaternions $(q_1, q_2)$.

The product $q_1\, q_2$ represents the composition $R_1 \circ R_2$.

*Demonstration*

We show that $\mathcal{R}_{q_1\, q_2}(v) = \mathcal{R}_{q_1} \circ \mathcal{R}_{q_2}(v)$.

$$\mathcal{R}_{q_1\, q_2}(v) = (q_1 q_2)\; v\; (q_1 q_2)^\star$$

$$\mathcal{R}_{q_1\, q_2}(v) = (q_1\; q_2)\; v\; (q_2^\star\; q_1^\star), \text{ as } (q_1 q_2)^\star = q_2^\star\; q_1^\star$$

$$\mathcal{R}_{q_1\, q_2}(v) = q_1\; (q_2\; v\; q_2^\star)\; q_1^\star$$

$$\mathcal{R}_{q_1\, q_2}(v) = q_1\; \mathcal{R}_{q_2}(v)\; q_1^\star = \mathcal{R}_{q_1} \circ \mathcal{R}_{q_2}(v)$$

# Correspondance quaternion to rotation matrix

The unit quaternion $q = (x, y, z, w)$ represents the rotation given by the matrix

$$R = \begin{pmatrix} 1 - 2(y^2 + z^2) & 2(xy - wz) & 2(xz + wy) \\ 2(xy + wz) & 1 - 2(x^2 + z^2) & 2(yz - wx) \\ 2(xz - wy) & 2(yz + wx) & 1 - 2(x^2 + y^2) \end{pmatrix}$$

*Demonstration*

$$v' = \mathcal{R}_q(v) = q\, q_v\, q^\star = ((w^2 - s^2)\, v + 2(s \cdot v)\, s + 2w\, (s \times v), 0) \quad \text{with } s = (x, y, z)$$

$$v' = (w^2 - x^2 - y^2 - z^2)v + 2 \begin{pmatrix} x & y & z \end{pmatrix} v \begin{pmatrix} x & y & z \end{pmatrix}^T + 2w \begin{pmatrix} x & y & z \end{pmatrix} \times v$$

$$v' = \left( (w^2 - x^2 - y^2 - z^2)\, \mathrm{I} + 2 \begin{pmatrix} x^2 & xy & xz \\ xy & y^2 & yz \\ xz & yz & z^2 \end{pmatrix} + 2w \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix} \right) v$$

$$v' = R\, v, \text{ with } x^2 + y^2 + z^2 + w^2 = 1$$

# Summary - Correspondance quaternion / matrix-vector

## Representation and Operations

| | **3D space** | **Quaternion space** |
|---|---|---|
| *Vector* | $v = (v_x, v_y, v_z)$ | $q_v = (v, 0) = (v_x, v_y, v_z, 0)$ |
| *Rotation* | $R$ ($3 \times 3$ matrix) | $q = (x, y, z, w)$, $\|q\| = 1$ |
| *Apply rotation to vector* | $Rv$ | $q\ q_v\ q^\star$ |
| *Rotation composition* | $R_1\ R_2$ | $q_1\ q_2$ |

## Rotation to Quaternion

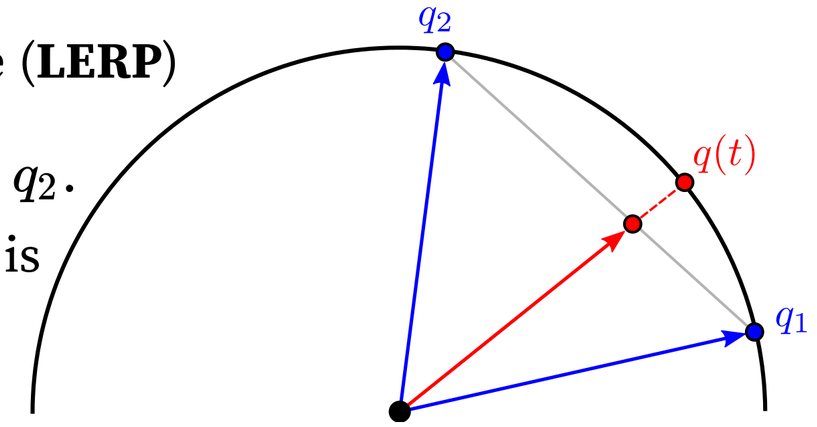
Rotation of axis $n$ and angle $\theta \Rightarrow q = (n \sin(\theta/2), \cos(\theta/2))$

## Quaternion to Rotation

Unit quaternion $q = (x, y, z, w) \Rightarrow R = \begin{pmatrix} 1 - 2(y^2 + z^2) & 2(xy - wz) & 2(xz + wy) \\ 2(xy + wz) & 1 - 2(x^2 + z^2) & 2(yz - wx) \\ 2(xz - wy) & 2(yz + wx) & 1 - 2(x^2 + y^2) \end{pmatrix}$

# Interpolation of Quaternion - LERP

Linear interpolation (with normalization) in quaternion space (**LERP**)




- Consider two rotations with corresponding quaternion $q_1$, $q_2$.
- The *linearly interpolated* quaternion at parameter $t \in [0, 1]$ is

$$q(t) = \frac{(1-t)\,q_1 + t\,q_2}{\|(1-t)\,q_1 + t\,q_2\|}$$

(+) Follows a shortest path (great circle on 4D sphere)

(+) Can be generalized to more general parametric curves (Splines, etc)

(-) Angular speed of orientation is not-constant

*ex. extreme case of two opposite quaternions*

# Interpolation of Quaternion - SLERP

Spherical Linear interpolation (**SLERP**)

- Consider two rotations with corresponding quaternion $q_1, q_2$.
- The *spherical linear interpolated* quaternion at parameter $t \in [0, 1]$ is

$$q(t) = \frac{\sin((1-t)\Omega)}{\sin(\Omega)} q_1 + \frac{\sin(t\Omega)}{\sin(\Omega)} q_2 \quad \text{with } \cos(\Omega) = q_1 \cdot q_2$$

*Demonstration*

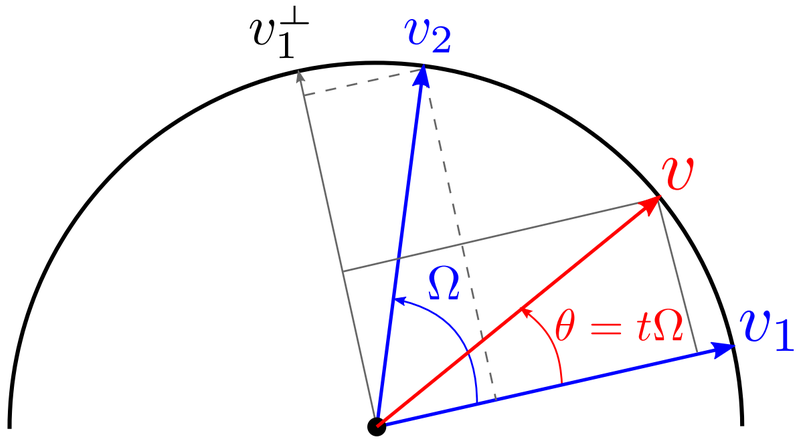
Consider

- Two unit vectors (arbitrary dimensions) $v_1, v_2$.
- $\Omega$: angle b/w $v_1$ and $v_2$
- The interpolated vector $v$ at angle $\theta = \Omega t, t \in [0, 1]$.

$v = v_1 \cos(\theta) + v_1^{\perp} \sin(\theta)$, and $v_1^{\perp} = \frac{v_2 - \cos(\Omega)\, v_1}{\sin(\Omega)}$

$\Rightarrow v = \left( \frac{\sin(\Omega)\ \cos(\theta) - \cos(\Omega)\ \sin(\theta)}{\sin(\Omega)} \right) v_1 + \frac{\sin(\theta)}{\sin(\Omega)} v_2$

$\Rightarrow v = \frac{\sin(\Omega - \theta)}{\sin(\Omega)} v_1 + \frac{\sin(\theta)}{\sin(\Omega)}$

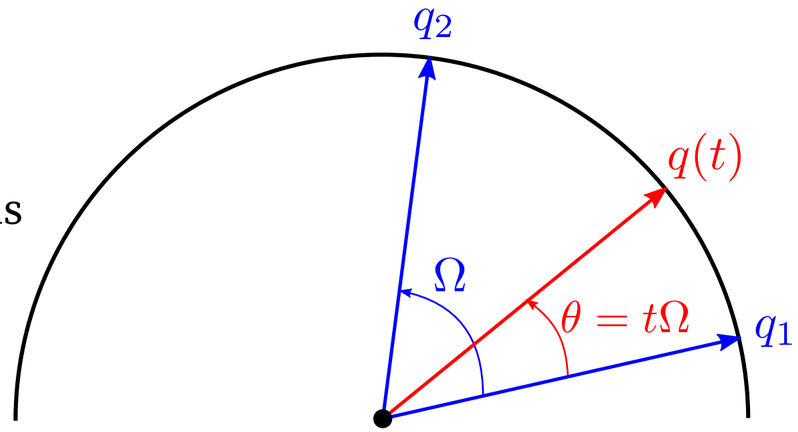


# Interpolation of Quaternion - SLERP

Spherical Linear interpolation (**SLERP**)

- Consider two rotations with corresponding quaternion $q_1, q_2$.
- The *spherical linear interpolated* quaternion at parameter $t \in [0, 1]$ is

$$q(t) = \frac{\sin((1-t)\Omega)}{\sin(\Omega)} q_1 + \frac{\sin(t\Omega)}{\sin(\Omega)} q_2 \quad \text{with } \cos(\Omega) = q_1 \cdot q_2$$



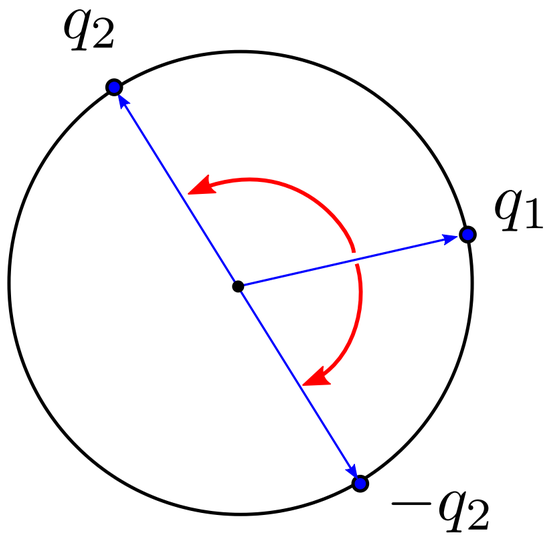

(+) Follows a shortest path (great circle on 4D sphere)

(+) Constant angular speed

(-) Cannot be generalized to more general curves.

*Cannot interpolates between more than two quaternions*

# Care with quaternion negation




$+q$ and $-q$ correspond to the same rotation $(n \to -n, \theta \to 2\pi - \theta)$

*Warning* $-q$ ***does not*** *corresponds to the rotation matrix* $-R$.

But to a **different path** when interpolated in the 4D quaternion space.

$\Rightarrow$ path $q_1 \to -q_2$ is shorter than $q_1 \to q_2$ when $q_1 \cdot q_2 < 0$.

In practice we check for the shorter path before applying SLERP.

*Algorithm*

```
if( dot(q1,q2)<0 )
    q2 = -q2
q(t) = SLERP(q1,q2,t)
```

# Interpolating rigid motion

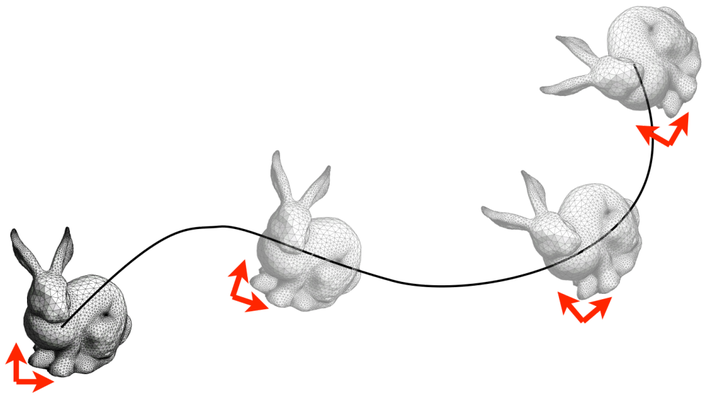


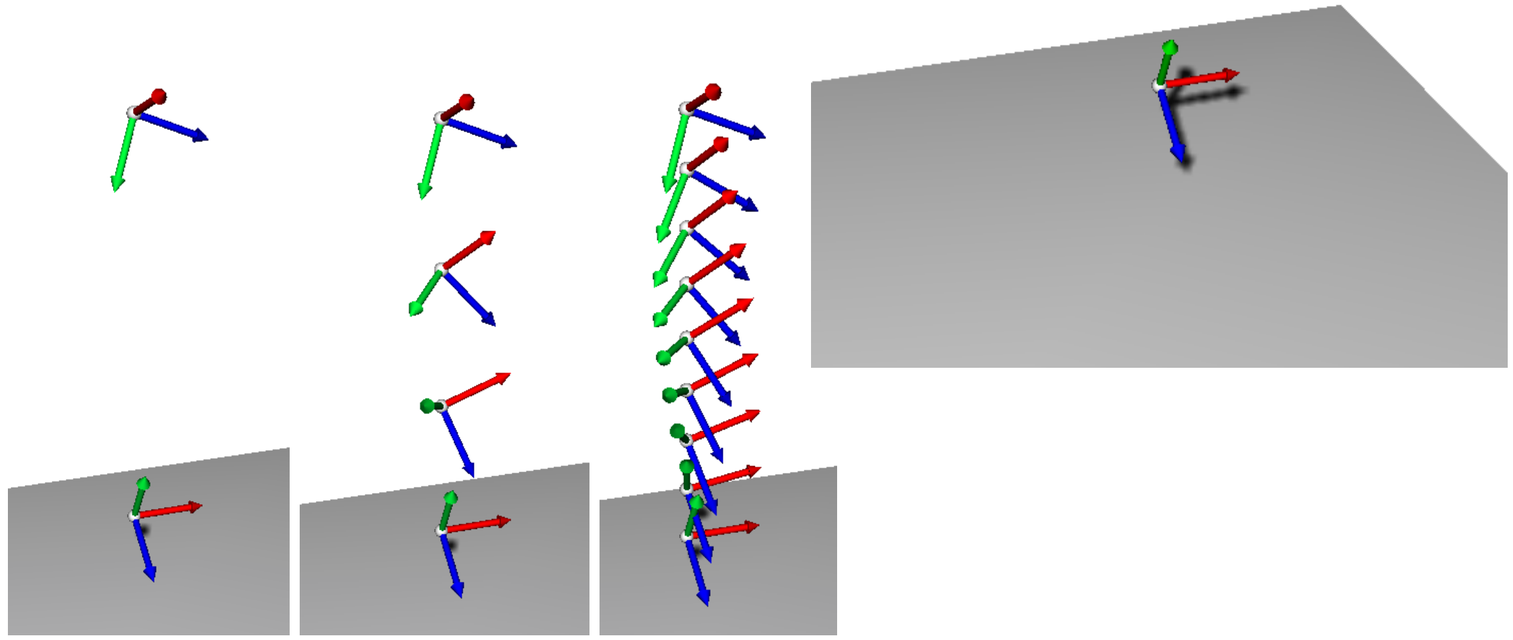
3D objects have orientation $r$ and position $p$.

$\Rightarrow$ Need to interpolate both, usually handled separately.

Given two key-positions $(p_1, r_1)$ and $(p_2, r_2)$ *in-betweens* computed at time $t \in [0, 1]$ as

- Linear interpolation of positions $p(t) = (1 - t)\, p_1 + t\, p_2$
- Interpolate rotation with SLERP on quaternions
  - Convert $(r_1, r_2) \rightarrow (q_1, q_2)$
  - Compute $q(t) = SLERP(q_1, q_2, t)$
  - Convert back $q(t) \rightarrow r(t)$





# Interpolating rigid motion - Comparison

*Matrix interpolation*

*Euler angle interpolation*

*Quaternion interpolation*

# Handling affine transformation : Polar Decomposition

Key-pose can be given as matrix M containing rotation, but also scaling and shearing.

$\Rightarrow$ Cannot convert $M$ to quaternion (not a rotation).

$\Rightarrow$ Separate *rotation* component from shearing and scaling component using **polar decomposition**.

[K. Shoemake and T. Duff, Matrix Animation and Polar Decomposition. Graphics Interface 92.]

- Polar decomposition: $\mathrm{M} = \mathrm{R}\,\mathrm{D}$
  - R: Rotation matrix
  - D: Positive semi-definite matrix

- Polar decomposition is obtained from SVD

$\mathrm{SVD}(\mathrm{M}) = W\Sigma V^T$ with $\boxed{\mathrm{R} = \mathrm{W}\mathrm{V}^T}$, $\mathrm{D} = \mathrm{V}\Sigma\mathrm{V}^T$

- Numerically, R can be computed using the following iterative scheme

$$\boxed{R_0 = \mathrm{M}\,,\ R_{i+1} = 0.5\,(R_i + (R_i^{-1})^T)}$$

# Handling affine transformation

Algorithm to interpolate between two general $4 \times 4$ matrix $M_1, M_2$

1. Extract translation $p_1, p_2$ from $M_1, M_2$.
2. Compute $R_1$, $R_2$ ($3 \times 3$ rotation matrices), and $D_1$, $D_2$ ($3 \times 3$ scaling/shearing matrices) from $M_1, M_2$
3. Interpolate linearly position and scaling/shearing $p(t) = (1-t)\, p_1 + t\, p_2$ , $D(t) = (1-t)\, D_1 + t\, D_2$
4. Compute quaternions $q_1$, $q_2$ from $R_1$, $R_2$

   Note $M \to q$ with $q = \left( \frac{M_{zy} - M_{yz}}{2r}, \frac{M_{xz} - M_{zx}}{2r}, \frac{M_{yx} - M_{xy}}{2r}, \frac{r}{2} \right), r = \sqrt{1 + M_{xx} + M_{yy} + M_{zz}}$
5. Compute $q(t) = \mathrm{SLERP}(q_1, q_2, t)$
6. Convert back to matrix $q(t) \to R(t)$
7. Compute final matrix $M(t) = R(t)\, D(t)$ with translation $p(t)$.

# Shape deformation methods

# Spatial / Surface based deformation

Consider a shape defined as $(p_i)_{i \in [0,N-1]}$ positions

## Spatial/volume deformation

Compute spatial deformation

$\forall p \in \mathbb{R}^3 \, , \quad f : p \to f(p)$

Apply $f$ to every positions

$\forall i \in [0, N-1] \, , \; q_i = f(p_i)$

(+) Independant from shape representation
Point sets, Surface, Volume.
(-) Indirect control
Spatial deformation $\to$ shape deformation
(-) Deformation fully defined by spatial position

## Surface-based

Compute deformation only on surface positions $p_i$

Not defined in space

(+) Can integrate/preserve surface properties (neighborood, curvature, etc.)
(-) Sensible to surface representation & connectivity.

# Spatial deformation

How to compute deformation $f$ ?

# Parametric deformation

Use succession of parameterized space varying deformation

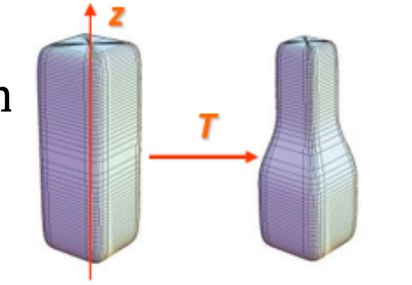
Rotation: Bending, Twisting

Scaling: Squeezing/tapering, Stretch

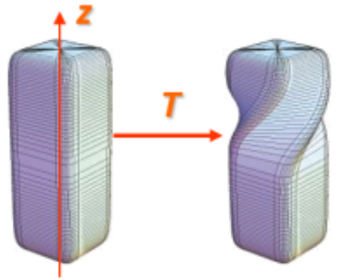
Old idea: First proposed in [Alan H. Barr. Global and Local Deformations of Solid Primitives. SIGGRAPH 1984]

(+) Explicit formulation

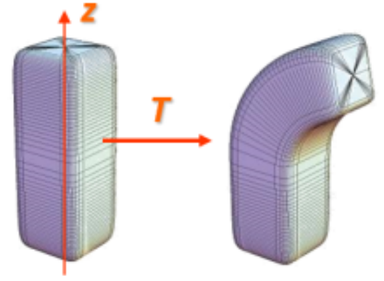
(-) Limited degrees of freedom




$$\begin{pmatrix} s(z) & 0 & 0 & 0 \\ 0 & s(z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$




$$\begin{pmatrix} \cos\theta(z) & \sin\theta(z) & 0 & 0 \\ -\sin\theta(z) & \cos\theta(z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$




$$\begin{pmatrix} \cos\theta(z) & 0 & -\sin\theta(z) & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta(z) & 0 & \cos\theta(z) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

# Local deformers

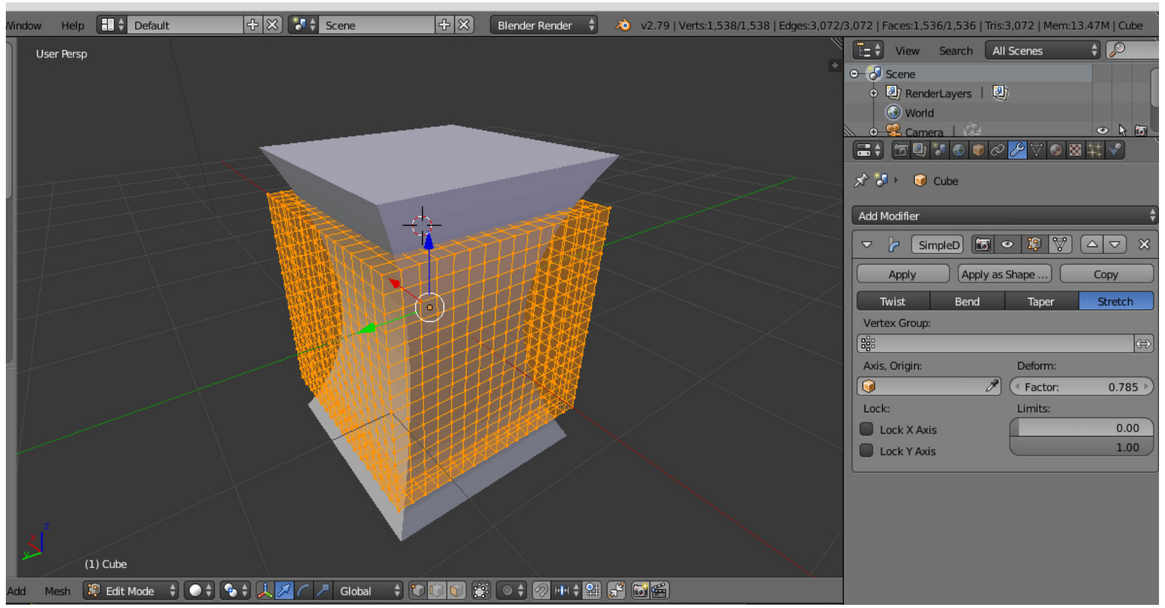
Led to variety of interactive loca **deformers** in 3D modeling softwares

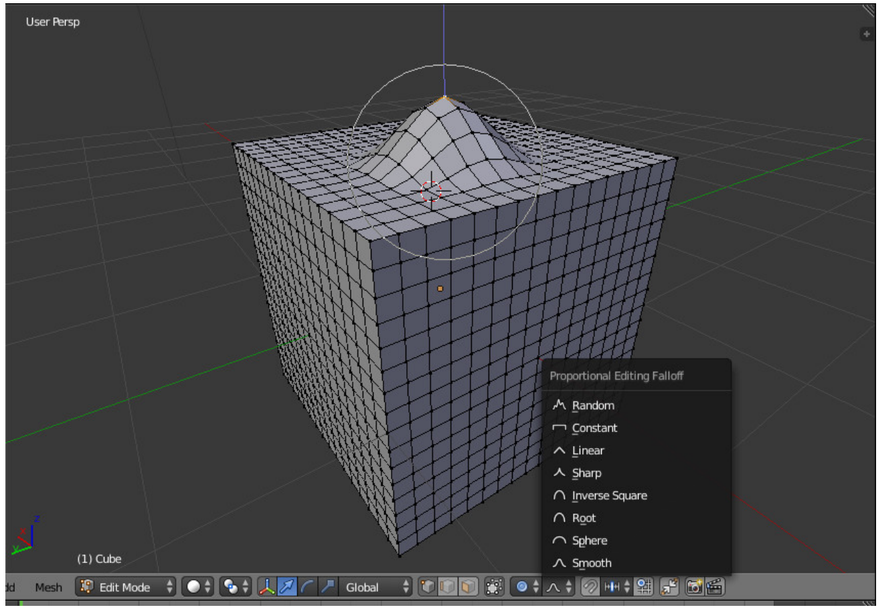
Example: Applying a translation $t$ only locally

$$q_i = p_i + w_i(p_i)\, t \quad , \quad w_i = \exp(-\|p_i/\sigma\|^2)$$

$t$ can be controled interactively

*and other parameters (rotation axis, angle, etc) as well.*



# Deformers example cases

How to model this interactive deformer ?


Input:

- $c$: Selected position on the surface
- $t_{2D}$: 2D displacement coordinates of the mouse on screen while dragging
- $r$: Circle radius

Computation:

For all positions of the surface $p$

...

# Deformers example cases

How to model this interactive deformer ?

# Deformers example cases

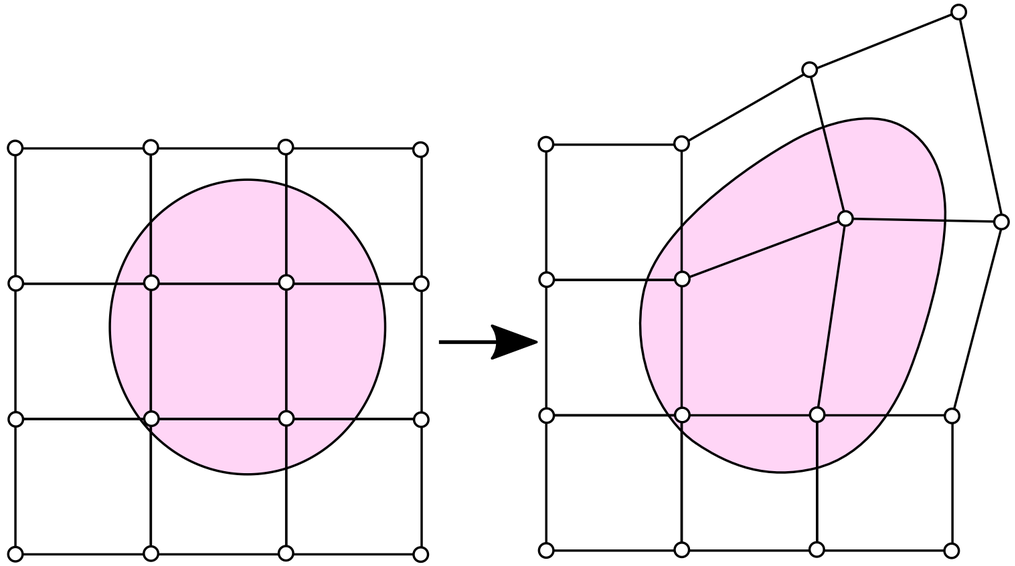
How to model this interactive deformer ?


# Free Form Deformation

FFD / Grid Based Deformation

- Embeed shape within a rectangular grid (lattice)
- Deform the grid $\rightarrow$ space deformation map

How to compute the space deformation ?



Kestrel Moon

# Idea: Approximation function

*Bezier, BSpline, NURBS, etc.*

Reminder for 1D curve $c$ approximating points $(p_i)_{i=0..N}$ (control polygon)

$$c(u) = \sum_{i=0}^{N} b_i(u)\, p_i$$

$$b_i(u) = \binom{N}{i} u^i \, (1-u)^{N-i} \;, \quad u \in [0,1] \text{ (Bernstein Polynomial)}$$

For a 2D surface $S$ approximating points $p_{ij}$

$$S(u,v) = \sum_i \sum_j b_i(u)\, b_j(v)\; p_{ij} \text{ (tensor-product)}$$

For a 3D volume $V$ approximating points on the lattice $p_{ijk}$

$$V(u,v,w) = \sum_i \sum_j \sum_k b_i(u) b_j(v) b_k(w)\; p_{ijk}$$

$\Rightarrow$ Use V as a spatial deformation on vertex coordinates $(u,v,w)$, use $p_{ijk}$ as grid.

# Free Form Deformation

Given

- $g_{ijk}$: position of the grid coordinates
- $p_m = (x_m, y_m, z_m) \in [0,1]^3$ : Initial vertex coordinates (expressed with respect to the grid)

$$\Rightarrow q_m = \sum_i \sum_j \sum_k b_i(x_m)\, b_j(y_m)\, b_k(z_m)\; g_{ijk}$$


Note:

$w^m_{ijk} = b_i(x_m) b_j(y_m) b_k(z_m)$

is a scalar weights independant of the deformation

$\rightarrow$ can be precomputed

$$\Rightarrow q_m = \sum_{ijk} w^m_{ijk}\; g_{ijk}$$

# FFD limits and extensions

FFD introduced in 1986

*[ Thomas Sederberg, Scott Parry. Free-Form deformation of solid geometric models, SIGGRAPH 1986. ]*

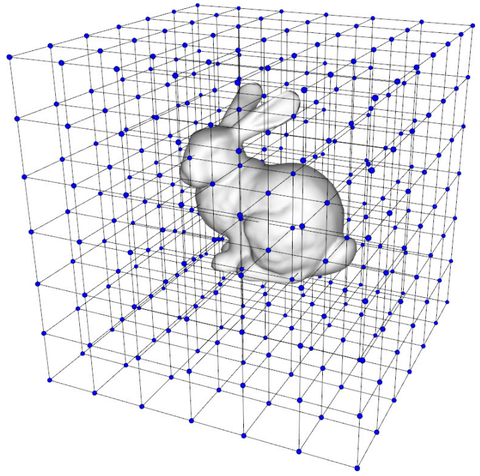
Initially described with Bezier polynomials

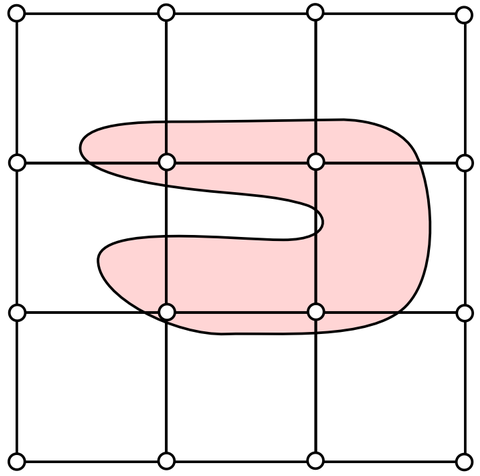
But extends to Spline with any basis function

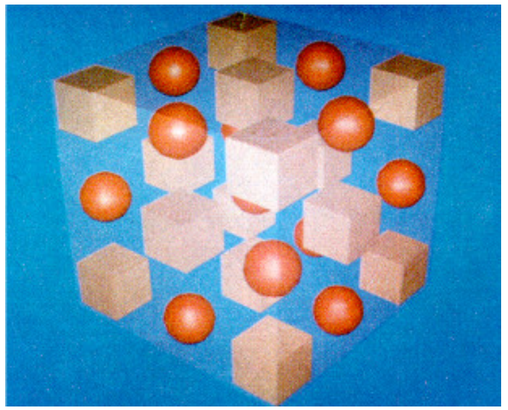
*BSpline, NURBS, etc*

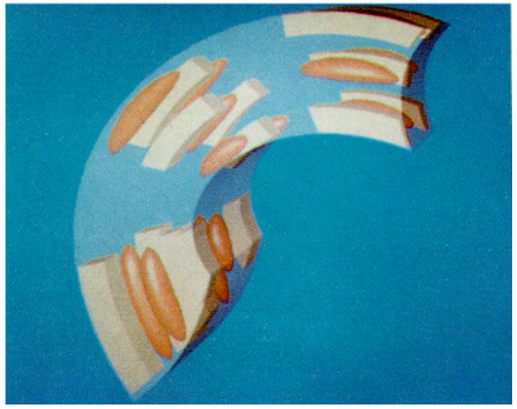
Limitations

- Large grid is hard to manipulate
- Doesn't take into account shape morphology





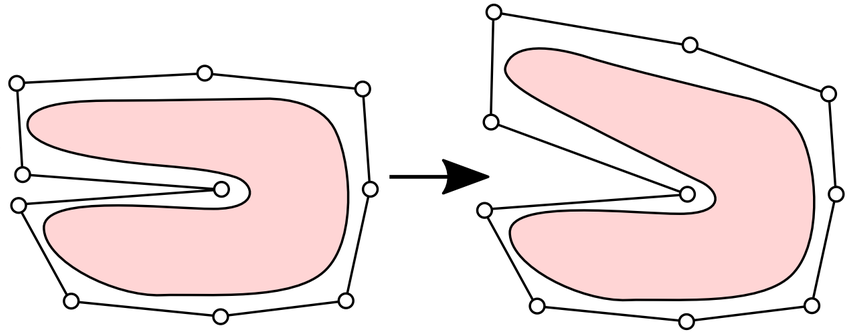




# Cage-based Deformation

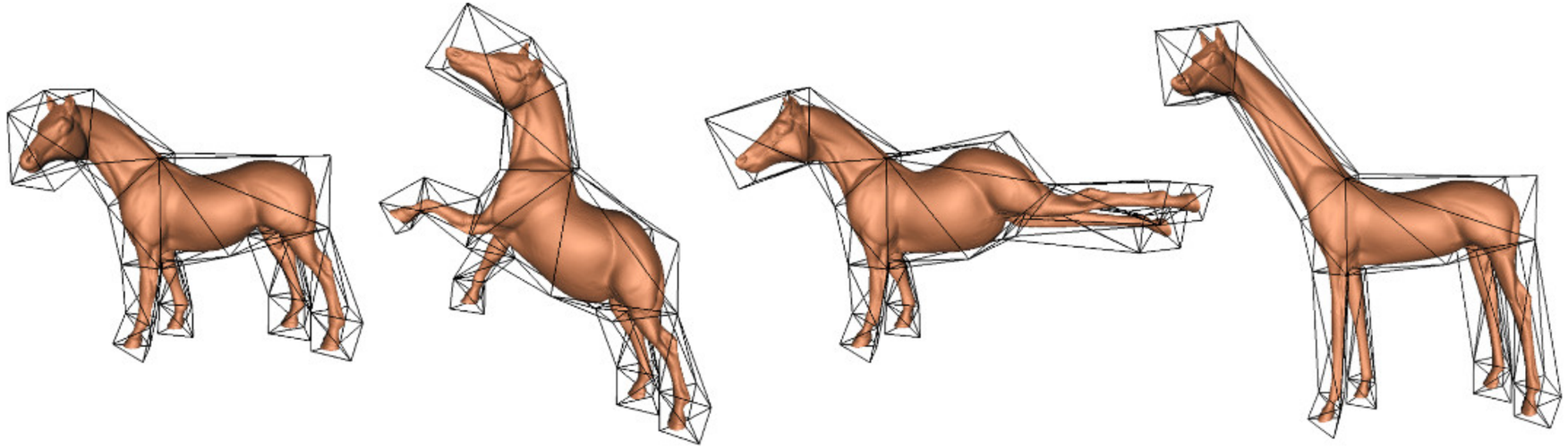
Use a cage surrounding the shape

(+) Adapts deformation locality to shape morpholog/topology



Introduced for 3D meshes in 2005

*[ Tao Ju, Scott Schaefer, Joe Warren. Mean value coordinates for closed triangular meshes. SIGGRAPH 2005 ]*



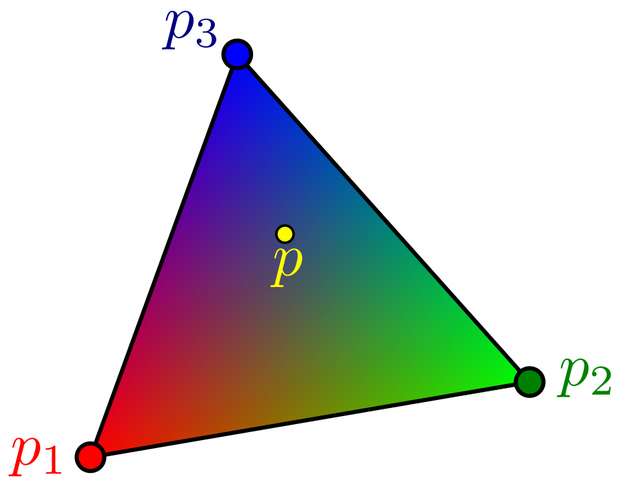
How to compute the spatial deformation from arbitrary cage ?

# Barycentric coordinates

## Reminder barycentric coordinates for Triangle

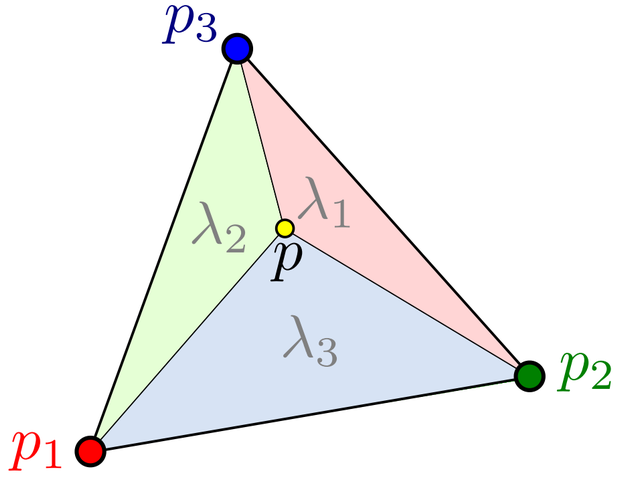
Triangle $(p_1\ p_2\ p_3)$ - barycentric coordinates $(\lambda_1, \lambda_2, \lambda_3)$

$$p \in (p_1\ p_2\ p_3) \Rightarrow p = \lambda_1\, p_1 + \lambda_2\, p_2 + \lambda_3\, p_3$$
$$\lambda_1 + \lambda_2 + \lambda_3 = 1\,, \quad (\lambda_1, \lambda_2, \lambda_3) \in [0,1]^3$$




$\lambda_1 = \text{area}(p_3\ p\ p_2)/\mathcal{A}$
$\lambda_2 = \text{area}(p_1\ p\ p_3)/\mathcal{A}$
$\lambda_3 = \text{area}(p_2\ p\ p_1)/\mathcal{A}$
with, $\mathcal{A} = \text{area}(p_1\ p_2\ p_3)$
and, $\text{area}(A\ B\ C) = \frac{1}{2}\|(p_2 - p_1) \times (p_3 - p_1)\|$

*Note. Similar for tetrahedron:* $p = \lambda_1\, p_1 + \lambda_2\, p_2 + \lambda_3\, p_3 + \lambda_4\, p_4$

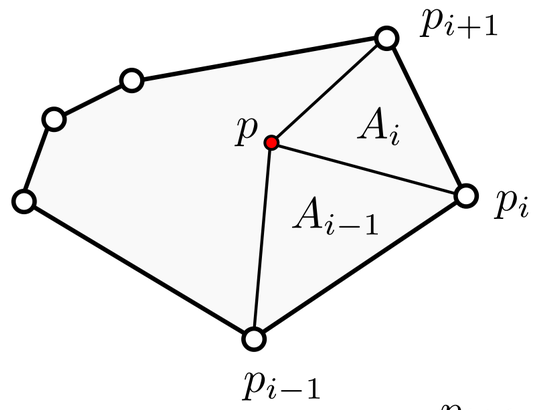



**Idea:** Generalize barycentric coordinates to arbitrary cages $p = \sum_i \lambda_i\, p_i = \sum_i \omega_i\, p_i / \sum_i \omega_i$

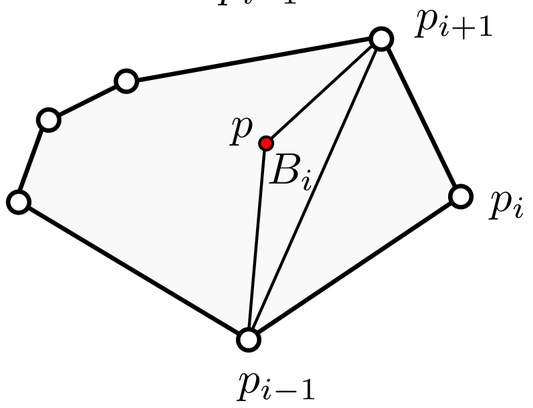
# Wachspress Coordinates

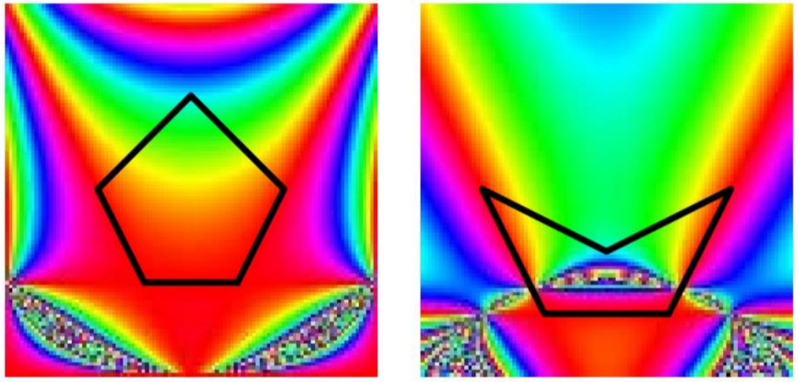
$$w_i = \frac{A_{i+1} + A_i - B_i}{A_{i+1}\, A_i}$$

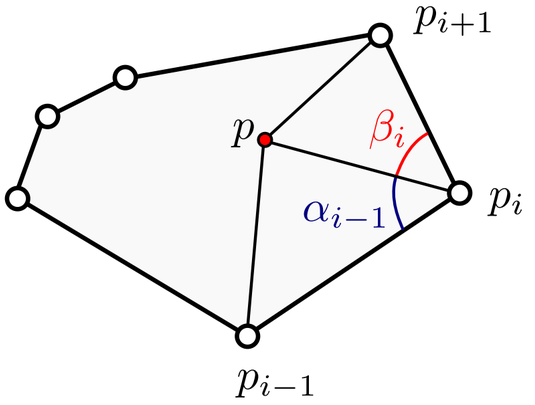
$$w_i = \frac{\cot(\alpha_{i-1}) + \cot(\beta_i)}{\|p_i - p\|}$$




Introduced in 1975, Eugene Wachspress, A Rational Finite Element Basis.






(-) Limited to 2D

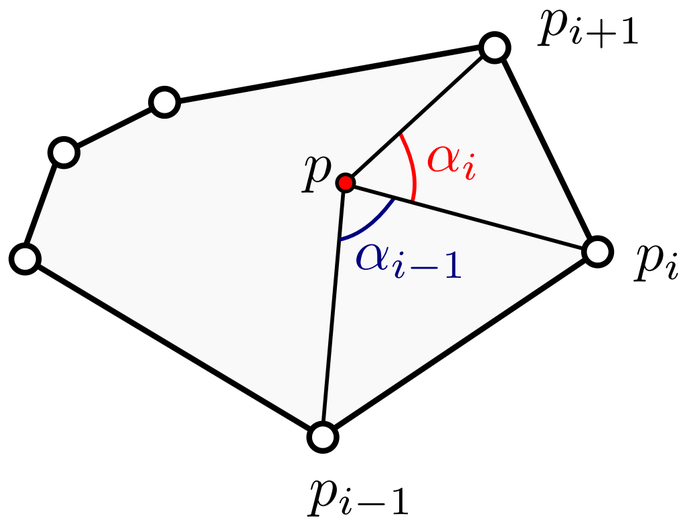
(-) Limited to convex polygons (negative weights, poles)

# Mean Value

## In 2D

$$\omega_i = \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{\|p_i - p\|}$$

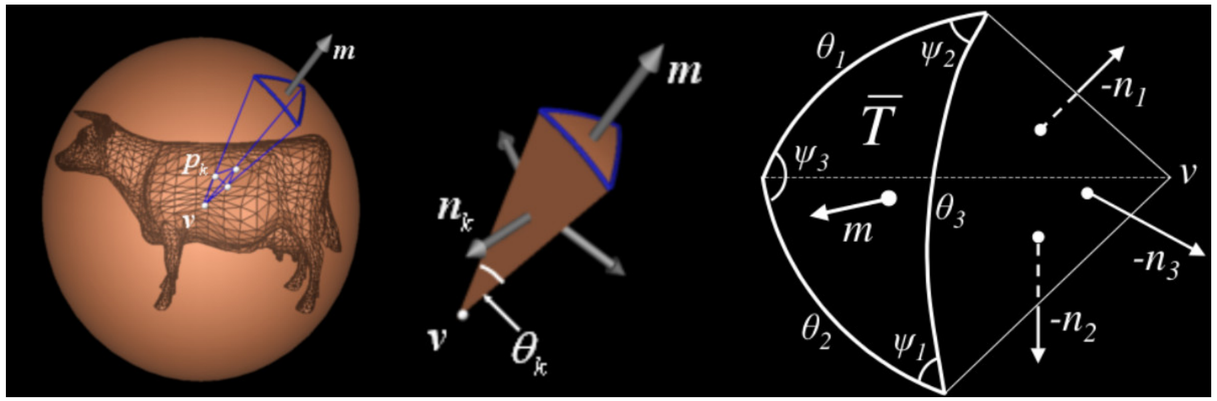
[Michael S. Floater. Mean value coordinates. CAGD, 2003]




## In 3D

$$\omega_i = \frac{n_i \cdot m}{n_i \cdot (p_i - v)}$$

$$m = \frac{1}{2} \sum_i \theta_i \, n_i$$




- [Michael S. Floater, Géza Kos, Martin Reimers. Mean value coordinates in 3D. CAGD, 2005]
- [Tao Ju, Scott Schaefer, Joe Warren. Mean Value Coordinates for Closed Triangular Meshes. SIGGRAPH 2005]

# Mean Value - Idea in 2D

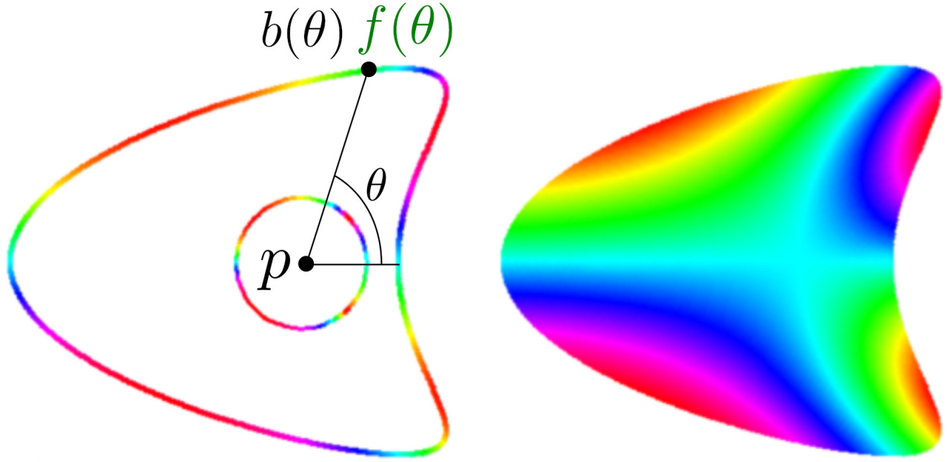
Look for an interpolant.

Given a 2D boundary curve $b(\theta)$, with values $f(\theta)$

Mean-value associated to point $p$:

$$\hat{f}\,(p) = \frac{\displaystyle\int_{\theta=0}^{2\pi} \kappa(\theta)\, f(\theta)\, \mathrm{d}\theta}{\displaystyle\int_{\theta=0}^{2\pi} \kappa(\theta)\, \mathrm{d}\theta}\,, \quad \kappa(\theta) = 1/\|b(\theta) - p\|$$

*Averaged value projected around a unit circle, weighted by the inverse distance.*

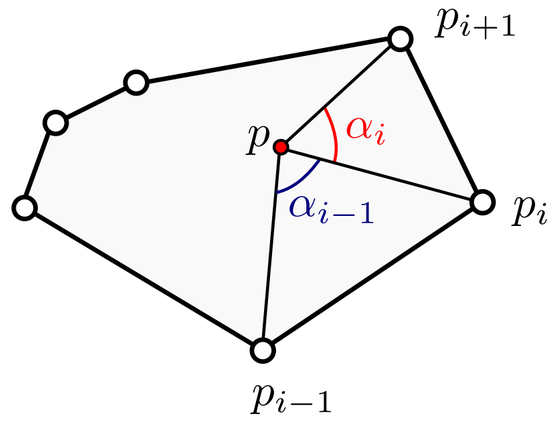



For a piecewise linear boundary

Over the edge $p_i\, p_{i+1}$

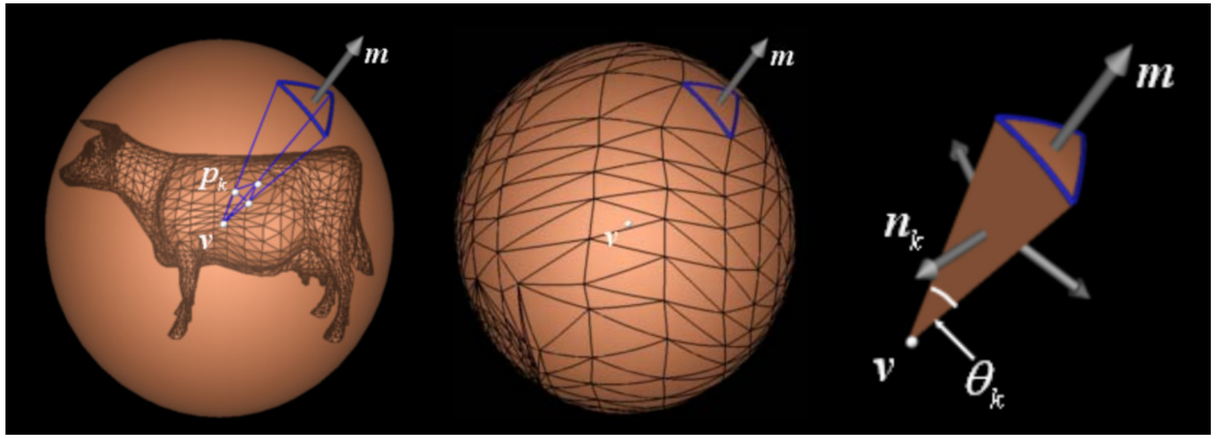
$$\int_{\theta=0_1}^{\theta=\theta_2} \kappa(\theta)\, f(\theta) = \cdots = \left( \frac{f(p_i)}{\|p_i - p\|} + \frac{f(p_{i+1})}{\|p_{i+1} - p\|} \right) \tan\left( \frac{\theta_2 - \theta_1}{2} \right)$$

$$\Rightarrow \omega_i = \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{\|p_i - p\|}$$



# Mean Value - Idea in 3D

In 3D: Integrate over a triangulated sphere

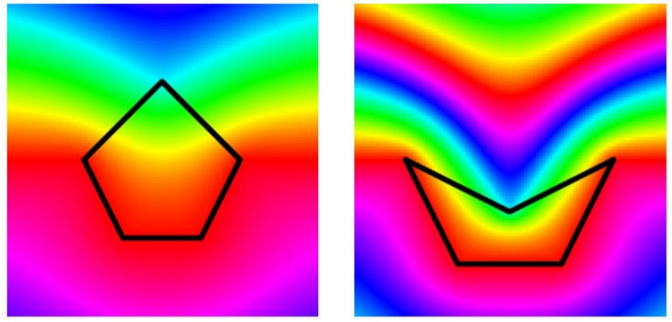



*[ Mean Value Coordinates for Closed Trianular Meshes. T. Ju et al. SIGGRAPH 2005 ]*

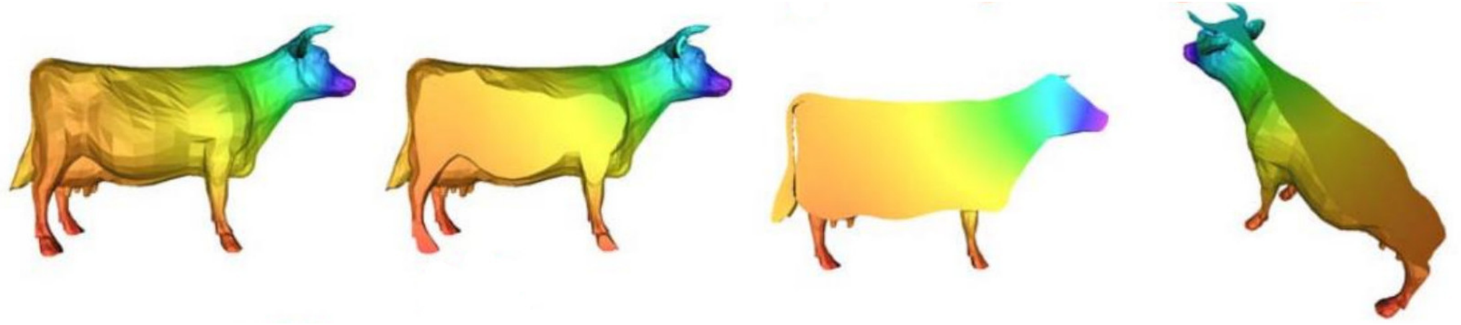
*// Robust evaluation on a triangular mesh*
for each vertex $p_j$ with values $f_j$
    $d_j \leftarrow \|p_j - x\|$
    if $d_j < \varepsilon$ return $f_j$
    $u_j \leftarrow (p_j - x)/d_j$
$totalF \leftarrow 0$
$totalW \leftarrow 0$
for each triangle with vertices $p_1, p_2, p_3$ and values $f_1, f_2, f_3$
    $l_i \leftarrow \|u_{i+1} - u_{i-1}\|$  *// for* $i = 1,2,3$
    $\theta_i \leftarrow 2\arcsin[l_i/2]$
    $h \leftarrow (\sum \theta_i)/2$
    if $\pi - h < \varepsilon$
        *// x lies on t, use 2D barycentric coordinates*
        $w_i \leftarrow \sin[\theta_i] d_{i-1} d_{i+1}$
        return $(\sum w_i f_i)/(\sum w_i)$
    $c_i \leftarrow (2\sin[h]\sin[h-\theta_i])/(\sin[\theta_{i+1}]\sin[\theta_{i-1}]) - 1$
    $s_i \leftarrow \text{sign}[det[u_1, u_2, u_3]]\sqrt{1 - c_i^2}$
    if $\exists i, |s_i| \leq \varepsilon$
        *// x lies outside t on the same plane, ignore t*
        continue
    $w_i \leftarrow (\theta_i - c_{i+1}\theta_{i-1} - c_{i-1}\theta_{i+1})/(d_i \sin[\theta_{i+1}] s_{i-1})$
    $totalF += \sum w_i f_i$
    $totalW += \sum w_i$
$f_x \leftarrow totalF/totalW$

# Mean Value - Results

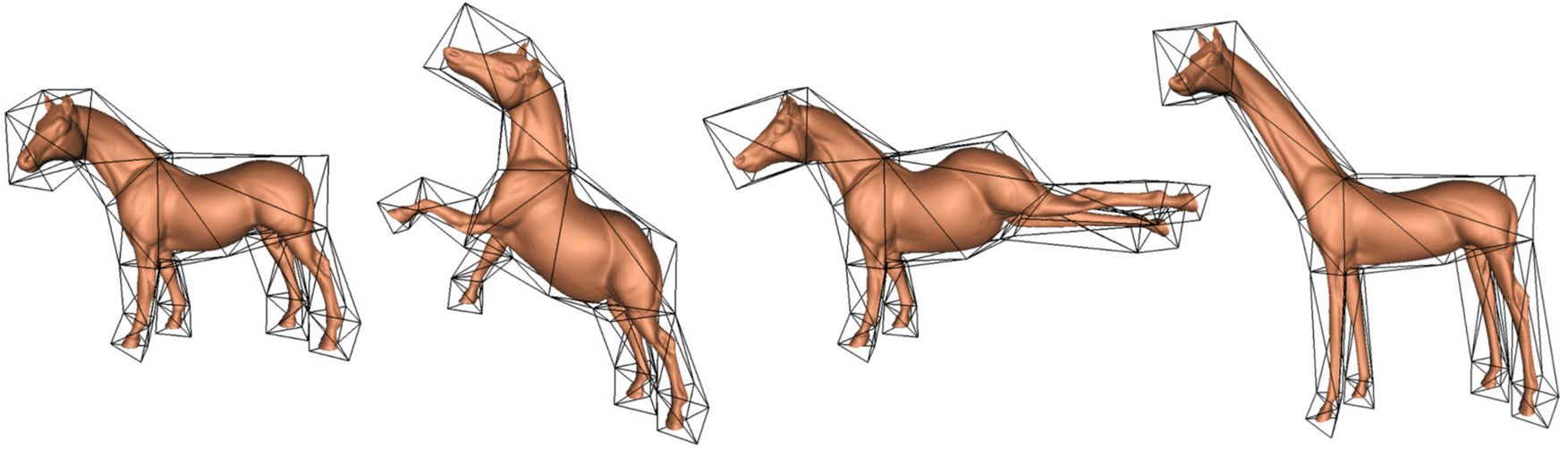
**Coordinates in 2D**



**Coordinates in 3D**



**Application to shape deformation**



# Harmonic Coordinates

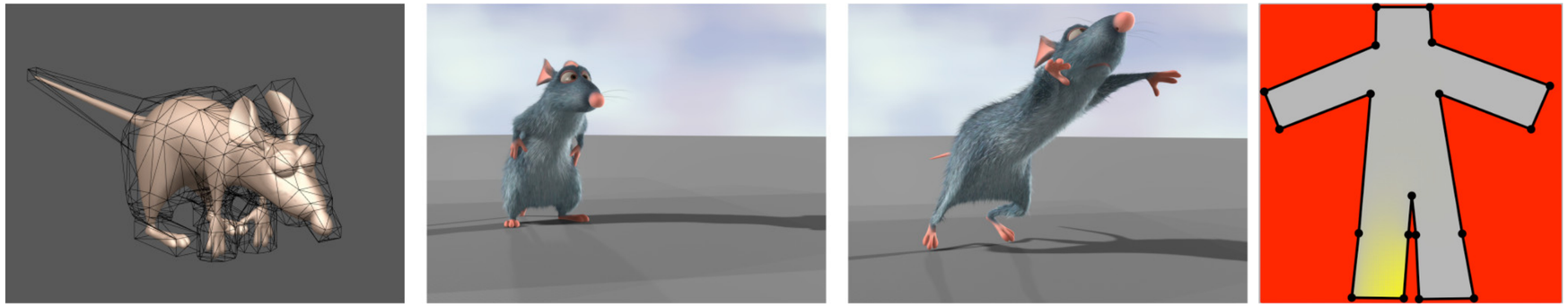
$\omega$ solution of the harmonic equation

$$\triangle \omega(p) = 0$$

$$\omega_i(p_j) = \delta_{ij}$$

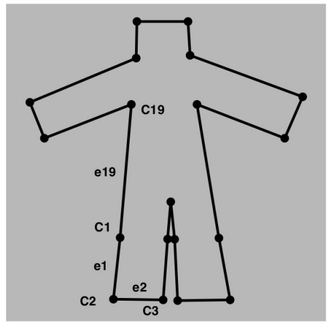
(-) No-closed form solution: requires numerical solver.

(+) Positive weights, even for concave cages

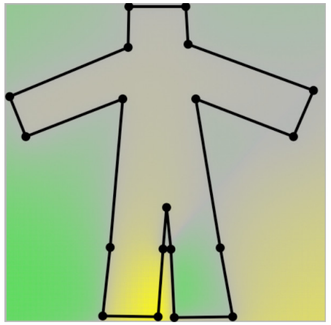


[Pushkar Joshi, Mark Meyer, Tony DeRose. Harmonic Coordinates for Character Articulation. SIGGRAPH 2007.]

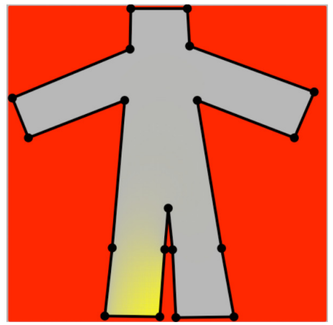
# Harmonic Coordinates VS Mean Value Coordinates



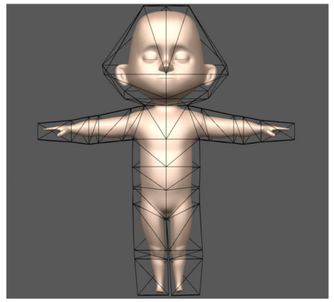

Bind



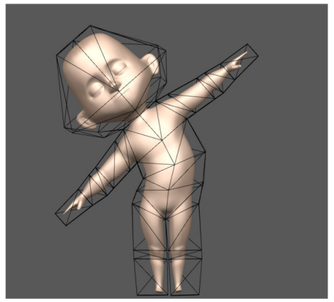
Mean Value



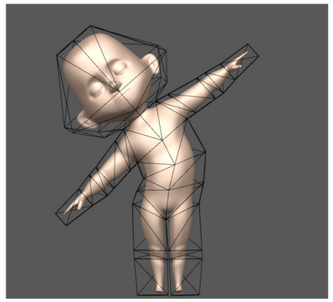
Harmonic



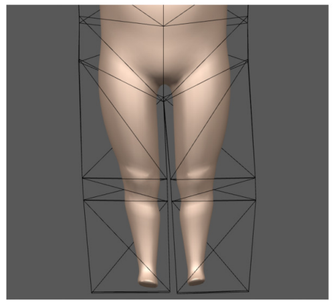
(a)

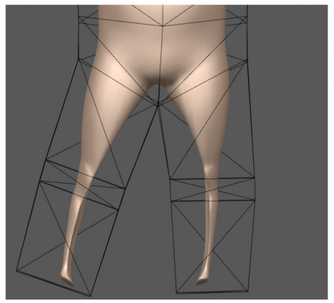


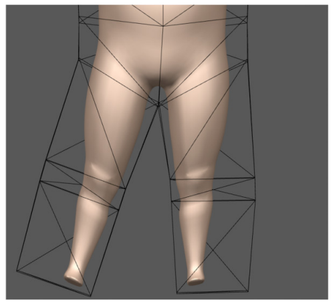
(b)



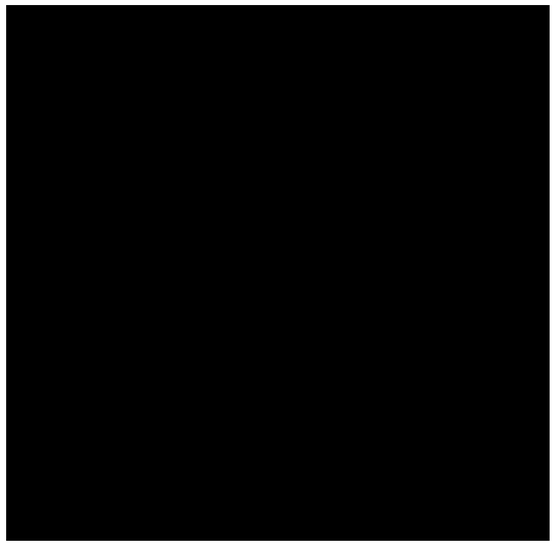
(c)







0:00



# Extensions

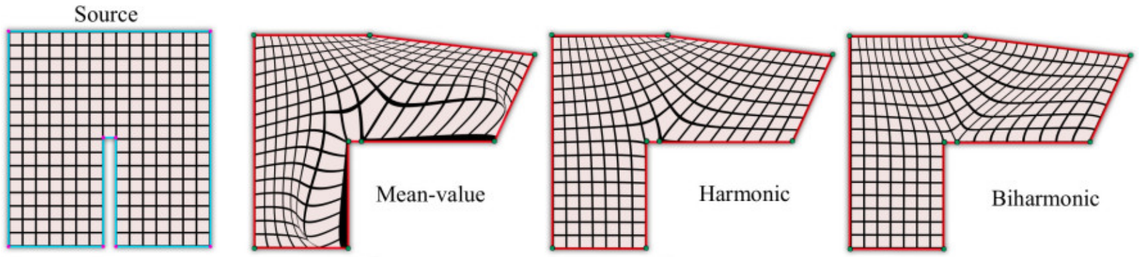
## Other coordinates

### Green coordinates

*[ Green Coordinates. Y. Lipman et al. SIGGRAPH 2008 ]*

### Bi-Harmonic Coordinates

*[ Biharmonic Coordinates. O. Weber et al. CGF 2012 ]*

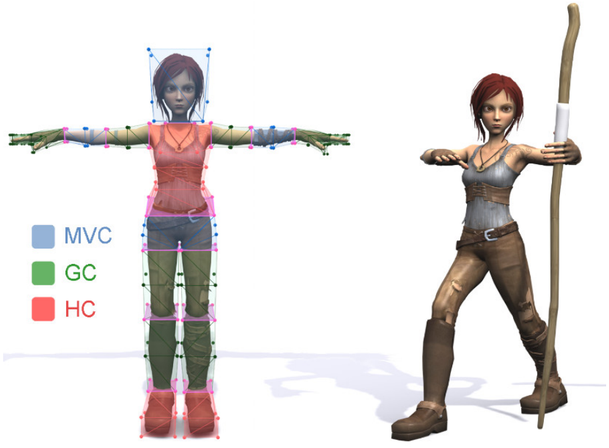



### Multi-cage

*[ *Cages: A multilevel, multi-cage-based system for mesh deformation. F. Garcia et al. ACM TOG 2013 ]*

### MVC on quad meshes

*[ Mean value coordinates for quad cages in 3D. J.-M. Thiery et al. SIGGRAPH Asia 2018 ]*




## Cage creation

### Building good cage is non-trivial task

#### Multiple works to avoid manual creation

*[ Skeleton Based Cage Generation Guided by Harmonic Fields. S. Casti et al. Computer & Graphics 2019 ]*

*[ CageR: Cage-based Reverse Engineering of Animated 3D Shapes. J.-M. Thiery et al. CGF 2012. ]*

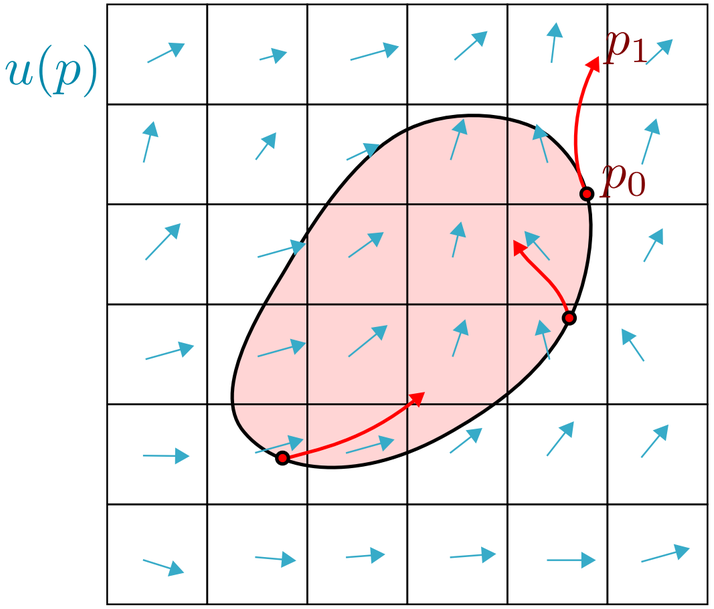
# Vector field deformation

Vector field in space as deformation speed $u : (x, y, z) \to (u_x(x, y, z), u_y(x, y, z), u_z(x, y, z))$

At arbitrary time $t$, $p'(t) = u(p(t))$

Applying deformation = integrate vertex position along streamlines

$$p_1 = p_0 + \int_t u(p(t))\,\mathrm{d}t$$




(+) If $\mathrm{div}(u) = 0 \Rightarrow$ constant volume deformation, no-intersection.

(-) Arbitrary vector fields hard to define and control.

*Note: All physically-based deformation can be seen as vector-field deformations*

# Building divergence free vector field

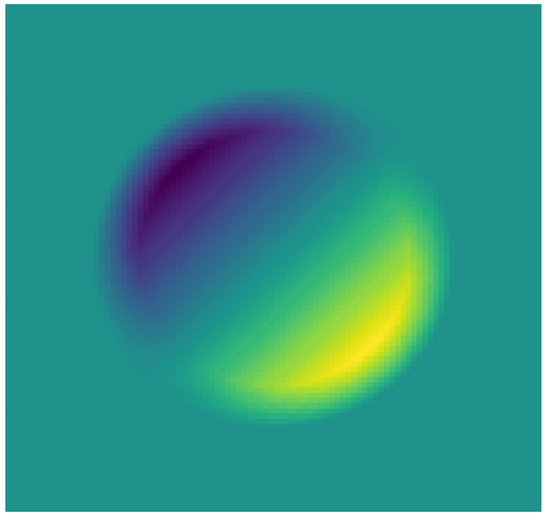
Example: Define 2 scalar fields $(a, b)$

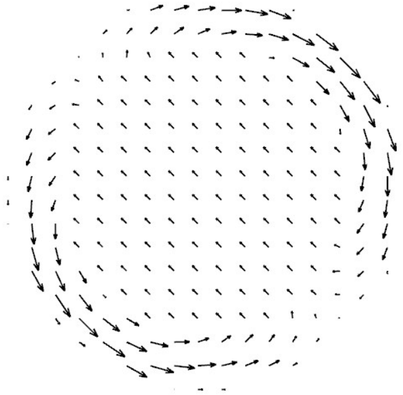
Build vector field $u = \nabla a \times \nabla b$

$$\operatorname{div}(u) = \operatorname{div}(\nabla a \times \nabla b)$$

$$\operatorname{div}(u) = \underbrace{\operatorname{curl}(\nabla a)}_{=0} \cdot \nabla b - \nabla a \cdot \underbrace{\operatorname{curl}(\nabla b)}_{=0}$$

$$\operatorname{div}(u) = 0 \Rightarrow \text{divergence free vector field}$$



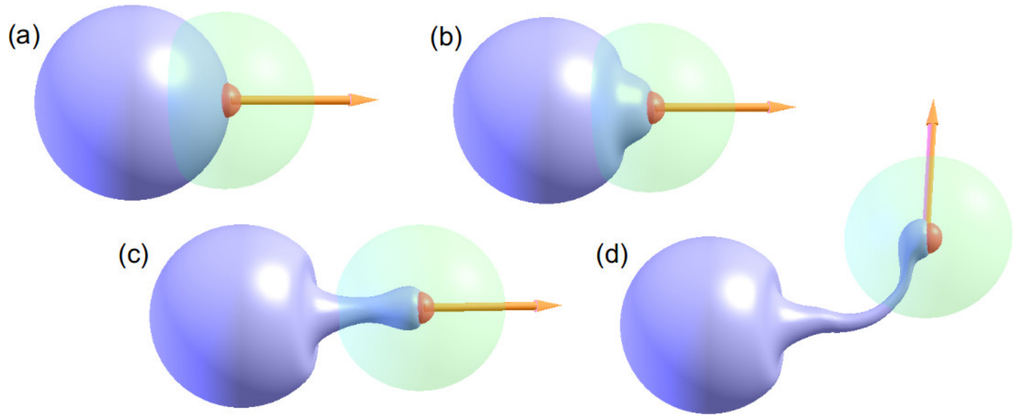


# Divergence free vector field example

Example from [Vector Field Based Shape Deformations. W. von Funck et al. SIGGRAPH 2006]

$$a(p) = \begin{cases} a_0(p) & \|p\| < r_1 \\ (1-\alpha(p))\, a_0(p) & r_1 \le \|p\| \le r_2 \\ 0 & \text{otherwise} \end{cases}$$

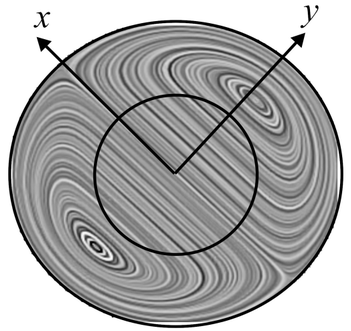
*Similarily with* $b(p)$

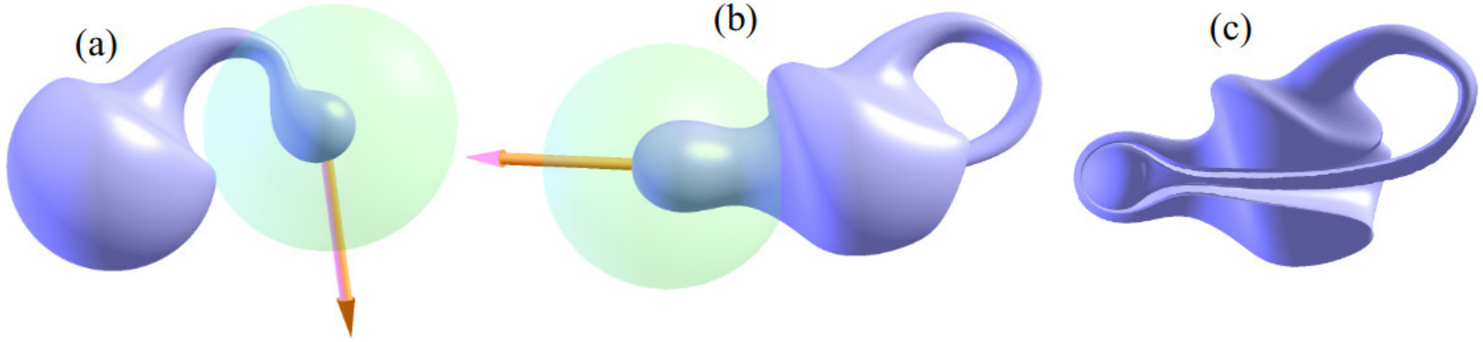



Translation: $a_0(p) = v_1 \cdot (p - p_0), b_0(p) = v_2 \cdot (p - p_0)$

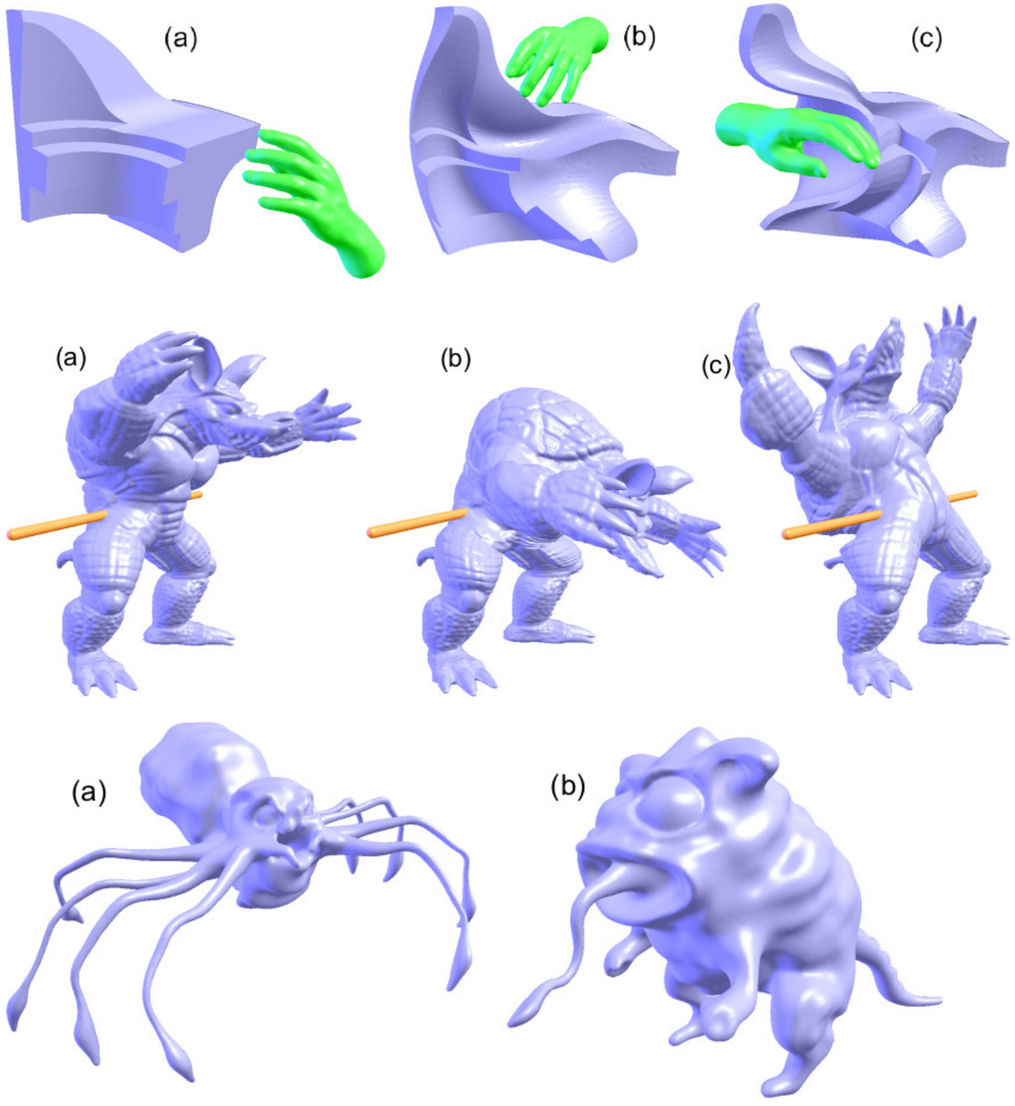
$v_1, v_2$ arbitrary orthogonal unit vector.

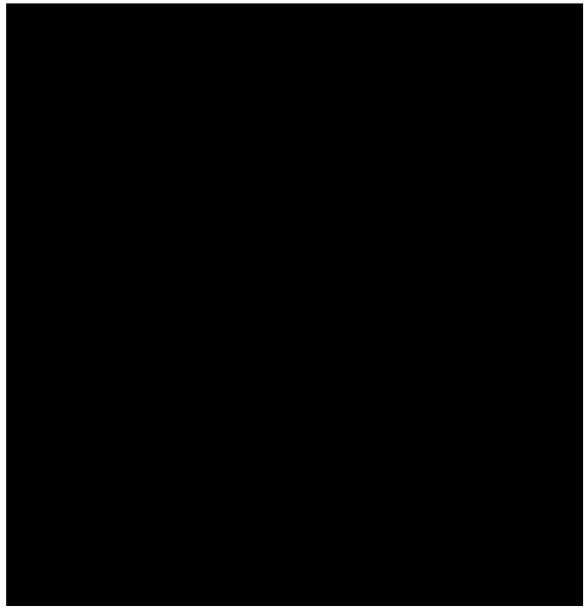
Bending: $a_0(p) = v_1 \cdot (p - p_0), b_0 = \|v_1 \times (p - p_0)\|^2$



# Example of results



# Case of implicit surfaces

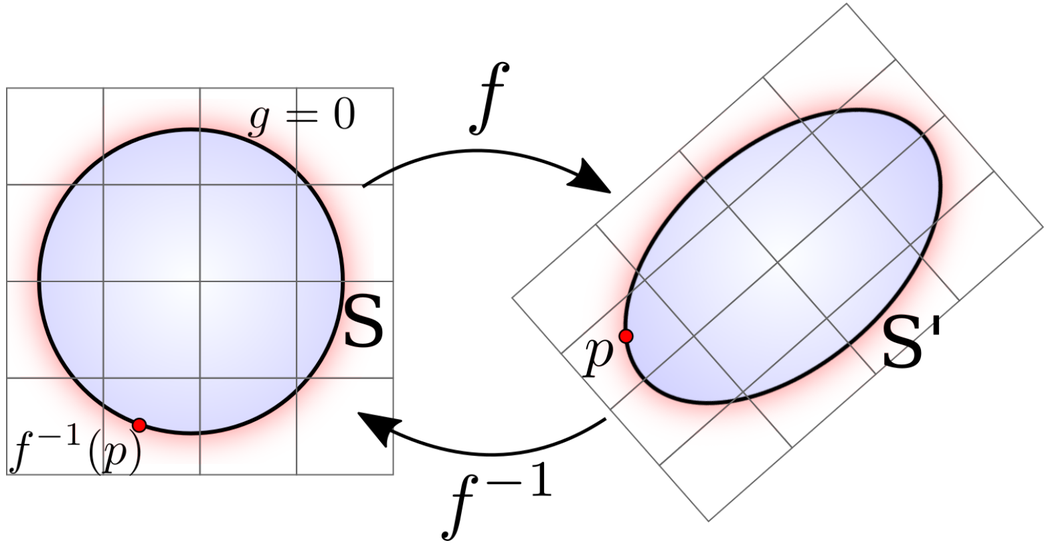
Undeformed implicit surface $S = \{p \in \mathbb{R}^3 \mid g(p) = 0\}$

Given a volume deformation $f : p \to f(p)$

The implicit surface $S'$ deformed by $f$ is defined by

$$S' = \{p \in \mathbb{R}^3 \mid (g \circ f^{-1})(p) = 0\}$$

$p \in S' \Rightarrow f^{-1}(p) \in S \Rightarrow g(f^{-1}(p)) = 0$




Given a vector field $u$

The implicit surface $S'$ advected by $u$ is defined as

$$\frac{\partial g}{\partial t} + u \cdot \nabla g = 0$$

*Note: Only velocity normal to the surface changes the shape*

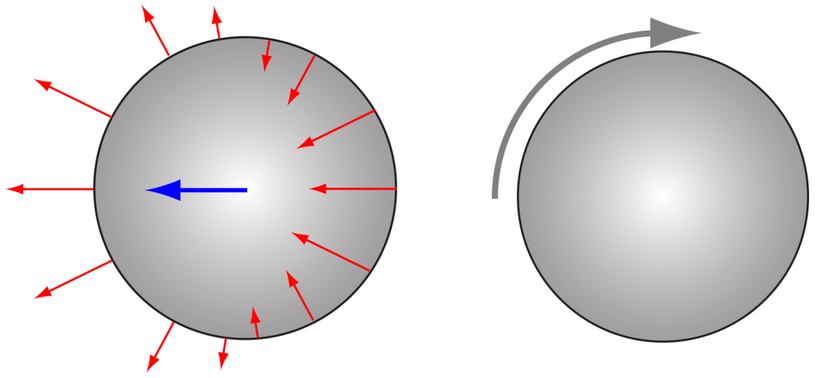
$g$ is a space-time function $g(p, t)$

$g(p + u, t + \mathrm{d}t) = g(p, t) = 0$

$\Rightarrow$ The differential $\mathrm{d}g = 0$

$\Rightarrow \frac{\partial g}{\partial t}\,\mathrm{d}t + \nabla g \cdot \mathrm{d}p = 0$

$\Rightarrow \frac{\partial g}{\partial t} + \nabla g \cdot \underbrace{\frac{\mathrm{d}p}{\mathrm{d}t}}_{u} = 0$



*[On the Velocity of an Implicit Surface]*

# Surface based deformation

# Laplacian/Differential Coordinates Editing

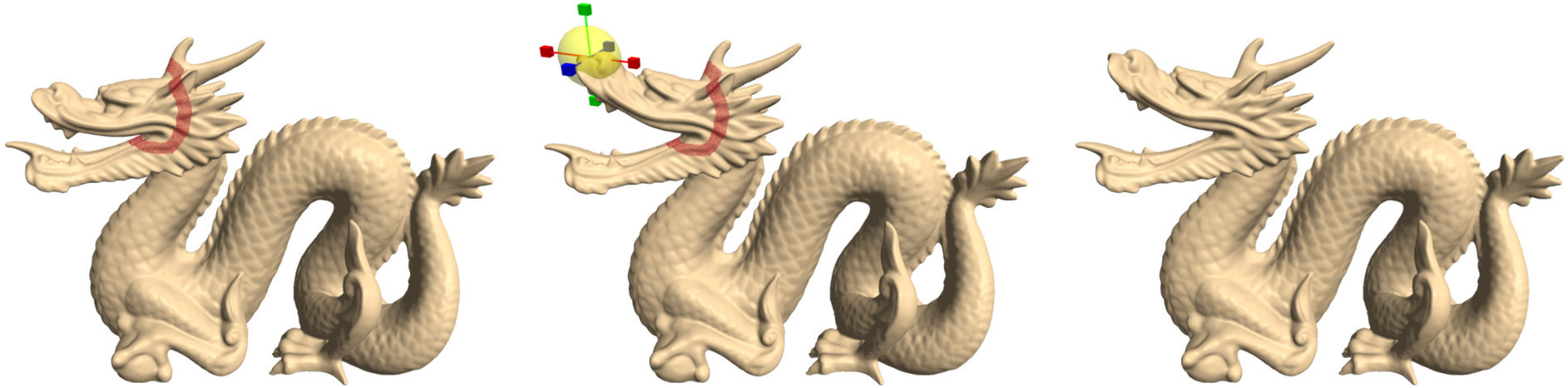
Initialization: Encode vertex position as *differential coordinates* w/r neighbors

Ex. Laplacian coordinate value $\delta p$

During deformation

- Try to match constrained position for subset of vertices
- While preserving the differential coordinates

*Local neighborhoods appearance*



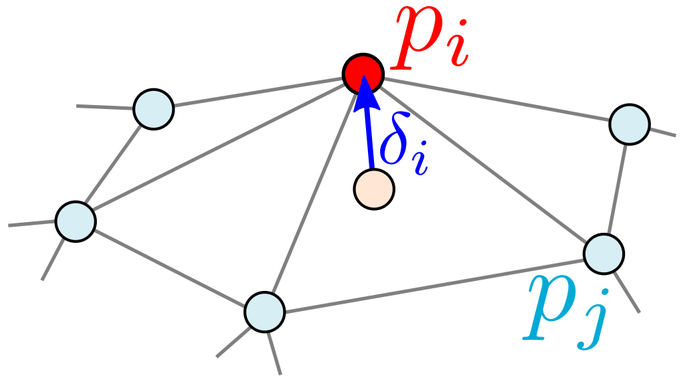
# Laplacian of the coordinates

- For a scalar/vector function $f$ defined in $\mathbb{R}^2$ with parameter $(u, v)$: $\triangle f = \frac{\partial^2 f}{\partial u^2} + \frac{\partial^2 f}{\partial v^2} = \mathrm{div}(\mathrm{grad}(f))$
- Laplacian on a manifold: Called Laplace-Beltrami operator.
- Differential coordinates $\delta$: Laplace-Beltrami operator applied on coordinates functions $f = (x, y, z)$ itself.

Common approximation in the discrete case

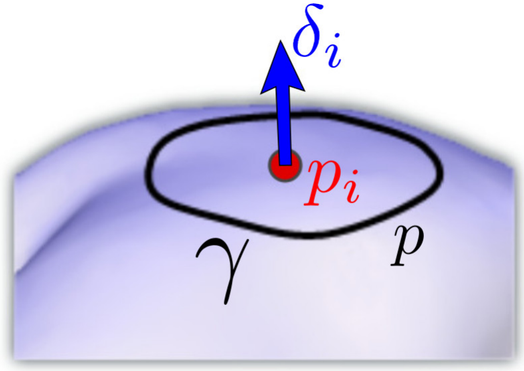
$$\delta_i = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} (p_i - p_j) = p_i - \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} p_j$$

$\mathcal{N}_i$: one ring neighborhood




Note: $\delta_i \simeq \lim_{|\gamma| \to 0} \frac{1}{|\gamma|} \int_{p \in \gamma} (p_i - p)\, \mathrm{d}l(p) = \mathrm{H}(p_i)\, n_i$

$\gamma$: small contour around $p_i$
$\mathrm{H}(p_i)$: Mean curvature at $p_i$, $n_i$ normal at $p_i$




$\Rightarrow$ Encode an approximation of the mean curvature

# Laplacian of the coordinates: Approximation

Simplest approximation of the laplacian: $\delta_i = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} (p_i - p_j)$

(-) Depends on the connectivity

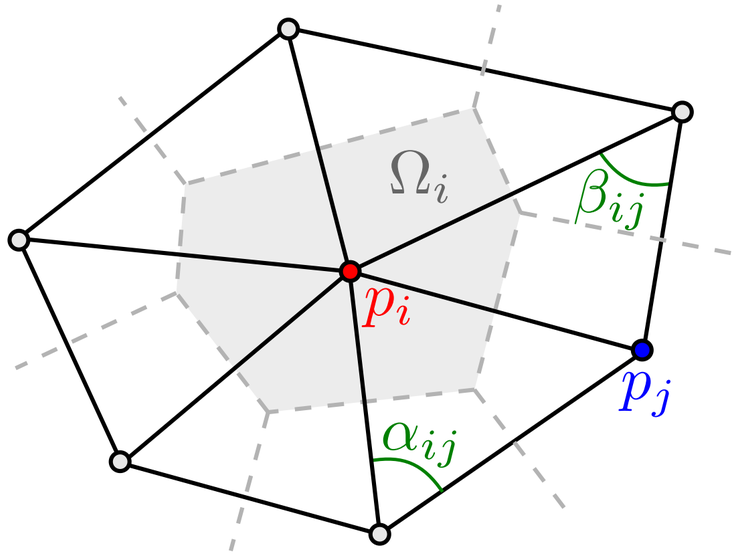
Other possible approximations: $\delta_i = \sum_{j \in \mathcal{N}_i} \omega_{ij} \, (p_i - p_j)$

Ex. Cotangent weights

$$\delta_i = \frac{1}{|\Omega_i|} \sum_{j \in \mathcal{N}_i} \frac{1}{2} (\cot \alpha_{ij} + \cot \beta_{ij}) \, (p_i - p_j)$$

Better properties: independant of the connectivity

Or using mean-value coordinates




No perfect approximation (satisfying all properties of the continuous Laplacian operator)

*[ Discrete Laplace operators: No free lunch. M. Wardetzky et al. SGP 2007 ]*

# Laplacian deformation formulation

Given initial position $(p_i)_{i\in[0,N-1]}$ + set of $N_c$ positional constraints $c_i$, $i \in C$.

Find $q_i$ such that

$\delta(q_i) = \delta(p_i)$ : Same differential coordinates

$\forall i \in C, q_i = c_i$ : Positional constraints

Formulate using quadratic energy w/r to unknown $q$

$$E = \sum_{i=0}^{N-1} \|\delta(q_i) - \delta(p_i)\|^2 + \omega \sum_{i\in C} \|q_i - c_i\|^2$$ , $w$: weight associated to positional constraints.

$$E = \sum_{i=0}^{N-1} \left\| q_i - \frac{1}{|\mathcal{N}_i|} \sum_{j\in\mathcal{N}_i} q_j - \delta(p_i) \right\|^2 + \omega \sum_{i\in C} \|q_i - c_i\|^2$$

# Laplacian deformation least square formulation

$$E = \sum_{i=0}^{N-1} \left\| q_i - \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} q_j - \delta(p_i) \right\|^2 + \omega \sum_{i \in C} \|q_i - c_i\|^2$$

Finding $q$ minimizing $E$ is similar to solve the least square problem

$$q^\star = \arg\min_q \ \|\mathrm{M}\, q - b\|^2$$

$q = (q_0, \cdots, q_{N-1})^T$ : $N$ unknown

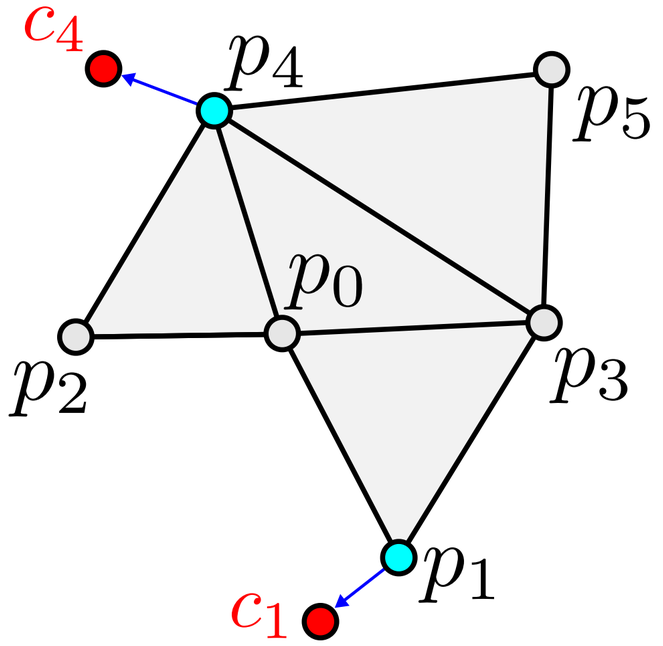
$\mathrm{M} = \begin{pmatrix} \mathrm{D} \\ \hline \mathrm{C} \end{pmatrix}$ with D ($N \times N$) ref. Laplacian weights; C ($N_c \times N$) ref. constraints.

- $\forall i \in [0, N-1]$ , $\mathrm{D}_{ii} = 1$ , $\mathrm{D}_{ij} = -1/|\mathcal{N}_i|$ if $j \in \mathcal{N}_i$
- $\forall k \in [0, N_c - 1]$ , $\mathrm{C}_{ki} = \omega$, with $i$ refering to the vertex index of the $k$th constraint.

$b = (\delta(p_0), \cdots, \delta(p_{N-1}),\ \omega\, c_0, \cdots, \omega\, c_{N_c-1})^T$

# Example of matrix

$$\underbrace{\left(\begin{array}{rrrrrr} 1 & -1/4 & -1/4 & -1/4 & -1/4 & 0 \\ -1/2 & 1 & 0 & -1/2 & 0 & 0 \\ -1/2 & 0 & 1 & 0 & -1/2 & 0 \\ -1/4 & -1/4 & 0 & 1 & -1/4 & -1/4 \\ -1/4 & 0 & -1/4 & -1/4 & 1 & -1/4 \\ 0 & 0 & 0 & -1/2 & -1/2 & 1 \\ \hline 0 & \omega & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \omega & 0 \end{array}\right)}_{\mathrm{M}} \underbrace{\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \end{pmatrix}}_{q} = \underbrace{\left(\begin{array}{c} \delta_0 \\ \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \\ \delta_5 \\ \hline \omega\, c_1 \\ \omega\, c_2 \end{array}\right)}_{b}$$




$$\begin{aligned}
\delta_0 &= p_0 - \tfrac{1}{4}(p_1 + p_2 + p_3 + p_4) \\
\delta_1 &= p_1 - \tfrac{1}{2}(p_0 + p_3) \\
\delta_2 &= p_2 - \tfrac{1}{2}(p_0 + p_4) \\
\delta_3 &= p_3 - \tfrac{1}{4}(p_1 + p_0 + p_4 + p_5) \\
\delta_4 &= p_4 - \tfrac{1}{4}(p_0 + p_2 + p_3 + p_5) \\
\delta_5 &= p_5 - \tfrac{1}{2}(p_3 + p_4)
\end{aligned}$$

# Laplacian deformation least square solution

Need to minimize $||\mathrm{M}\,q - b||^2$

Similar to solve the linear system $(\mathrm{M}^T\,\mathrm{M})\,q = \mathrm{M}^T\,b$ (*Normal equations*)

$\mathrm{M}^T\mathrm{M}$ is sparse (diagonally dominant)

Can use standard sparse solver (CG, Jacobi, Gauss Seidel, etc.)

*Reminder: Never try to compute the inverse matrix ! Only solve the linear system.*

*Special solvers (QR factorization) doesn't even need to compute the normal equations*

System can be solved independently on $x, y, z$ coordinates. Same matrix $\mathrm{M}$

$$\mathrm{M}\,q_x = b_x, \quad \mathrm{M}\,q_y = b_y, \quad \mathrm{M}\,q_z = b_z$$

Changing constraints position $c_i$ only impact rhs $b$

$\rightarrow$ $\mathrm{M}$ can be factorized once, and reused during interactive deformation

Changing constrained vertex impact $\mathrm{M}$

$\rightarrow$ need to be re-factorized

# Solving a dense least square problem

```cpp
#include <iostream>
#include <Eigen/Dense>
int main()
{
    // Build matrix
    Eigen::MatrixXf M;
    // M.resize(N,N)
    // Fill the matrix ... M(i,j) = value
    M = Eigen::MatrixXf::Random(3, 3); // example
    //QR Decomposition (different types are proposed in Eigen)
    auto solver = M.colPivHouseholderQr();


    // Build RHS
    Eigen::VectorXf b;
    // b.resize(N); b(i) = value ...
    b = Eigen::VectorXf::Random(3);


    // Solve linear system
    Eigen::VectorXf x = solver.solve(b);
    std::cout << x << std::endl;
    std::cout <<"Check solution : "<< (M*x-b).norm() << std::endl;
    return 0;
}
```

# Solving a sparse least square problem

```cpp
#include <iostream>
#include <Eigen/Sparse>
int main()
{
    // Build matrix
    Eigen::SparseMatrix<float> M;

    // Resize and reserve space to store coefficients
    //   ex. N rows, 10 coeffs per rows
    int N = 50;
    M.resize(N,N);
    M.reserve(Eigen::VectorXi::Constant(N,10));
    // Fill the matrix ... M.coeffRef(i,j) = value
    for(int k=0; k<N; ++k) {
        M.coeffRef(k,k)=1.0f; M.coeffRef(k,(k+1)%N)=-0.3f;
    }

    // Compress the matrix representation
    M.makeCompressed();
    // Factorization for a solver (here Conj. Gradient)
    Eigen::LeastSquaresConjugateGradient< Eigen::SparseMatrix<float> > solver;
    solver.compute(M);
    solver.setTolerance(1e-6f);

    // Build RHS
    Eigen::VectorXf b;
    b = Eigen::VectorXf::Random(N);

    // Solve linear system
    Eigen::VectorXf x = solver.solve(b);
    // or solver.solveWithGuess(b, x0)
    //   faster with good guess

    std::cout <<"Check solution : "
       << (M*x-b).norm() << std::endl;
    std::cout <<"N iterations : "
       << solver.iterations() << std::endl;

    return 0;
}
```

# Laplacian deformation: Results

Rem. Discrete solution of the Poisson equation with Dirichlet boundaries

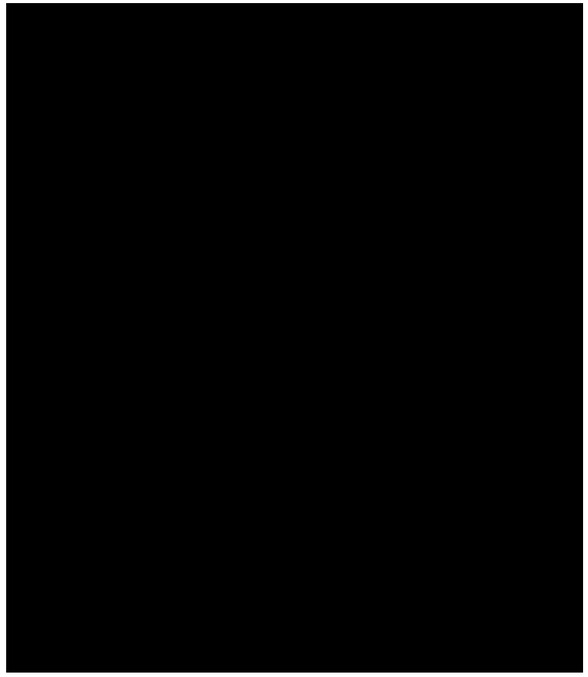
$\triangle f = \varphi$

$f = f^0$ on $\partial\Omega$

In the case of a plane: Laplacian equation $\triangle f = 0$

Solution are membranes

*Surface of minimal mean curvature*



# Laplacian deformation: Results

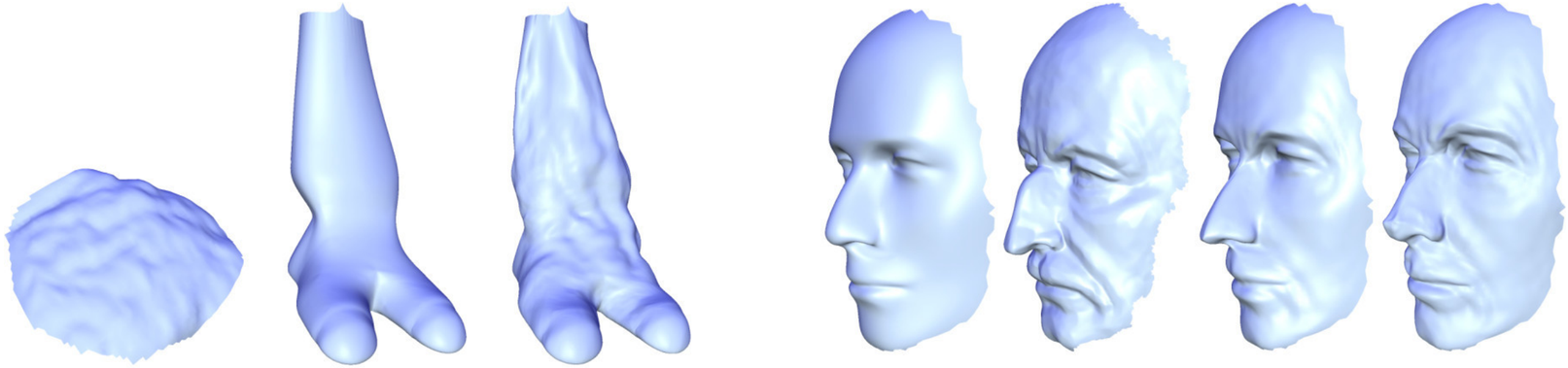

*[Laplacian Surface Editing. O. Sorkine et al. SGP 2004]*

# Extension: Detail transfert

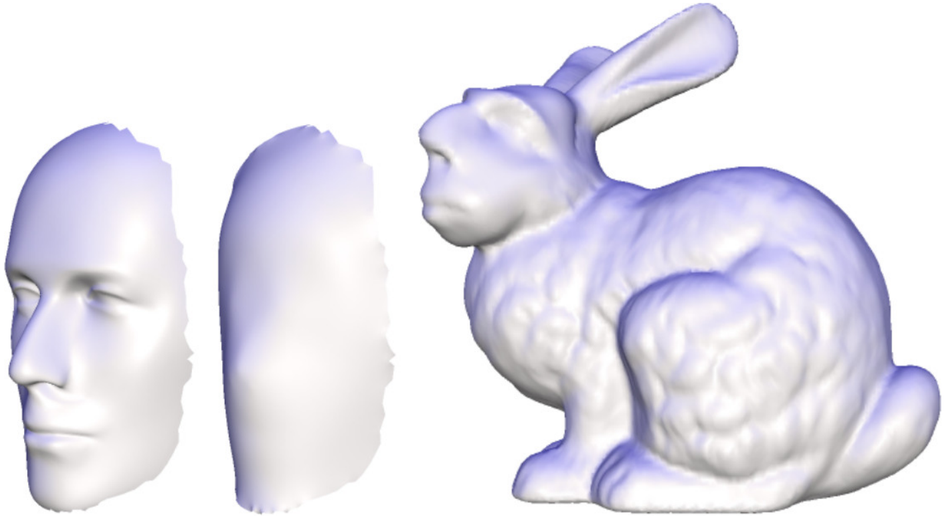
Use laplacian from another surface to transfert details

Requires a per-vertex correspondance

*User defined anchors*







# Laplacian editing: Sketching silhouette deformation


*[A Sketch-Based Interface for Detail-Preserving Mesh Editing. A. Nealen et al. SIGGRAPH 2005]*

# As Rigid As Possible (ARAP)

The Laplacian formulation tries to preserve the surface orientation.

While we could expect surface rotation


**Idea:** Include rotation transformation R in the formulation

$$E = \sum_{i=0}^{N-1} \|\delta(q_i) - R(q_i)\,\delta(p_i)\|^2 + \omega \sum_{i \in C} \|q_i - c_i\|^2$$

$R(q_i)$: best rotation at point $q_i$


Finding the best rotation at a given point

1- Look at surrounding edges at the point $q_i / p_i$

$e_j = q_j - q_i$, $e_j^0 = p_j - p_i$

2- Compute covariance matrix $\sigma = \sum_j e_j \, (e_j^0)^T$

3- Compute rotation using polar decomposition: $R = \text{polar}(\sigma)$

*ARAP*

# As Rigid As Possible (ARAP) Solution

Minimize the following term

$$E = \sum_{i=0}^{N-1} \|\delta(q_i) - R(q_i)\,\delta(p_i)\|^2 + \omega \sum_{i \in C} \|q_i - c_i\|^2$$

*Problem*: $R(q_i)$ is a non linear function of $q_i$

$E$ is non quadratic, cannot use least square.

*Idea*: Interleave and iterate on these two steps

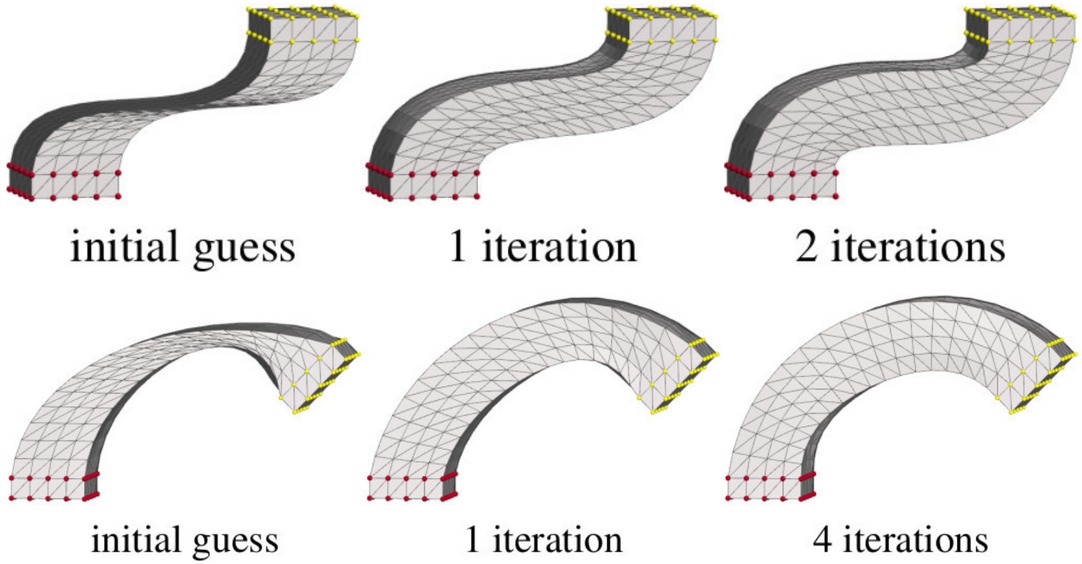
Find optimal rotation $R(q_i)$ with fixed $q_i$

Find $q_i$ optimizing $E$ with fixed $R(q_i)$

*Gain*: Surface rotates during deformation

*Limit*: Slightly more costly

*Iterations + polar decomposition*



# As Rigid As Possible: Results


*[As-Rigid-As-Possible Surface Modeling. O. Sorkine, M. Alexa. SGP 2007]*

# Other approaches

## Gradient representation

*[ Mesh editing with Poisson-based gradient field manipulation. Y.Yu et al. SIGGRAPH 2004]*

## Bi-Laplacien weights

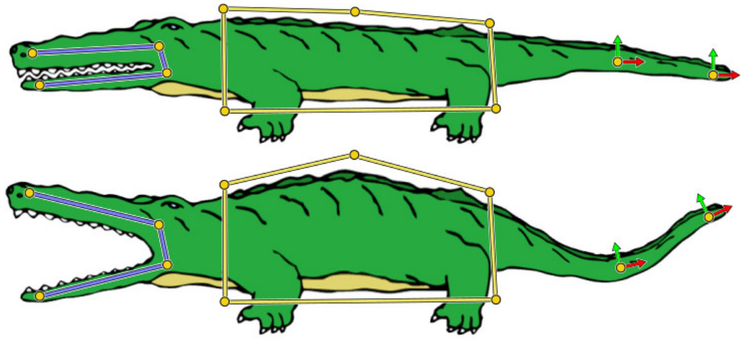
*[ Bounded Biharmonic Weights for Real-Time Deform. A. Jacobson et al. SIGGRAPH 2011 ]*

## Local frame representation

*[ Linear Rotation-invariant Coordinates for Meshes. Y. Lipman et al. SIGGRAPH 2005 ]*

## Deformation propagation

*[ Harmonic Guidance for Surface Deformation. R. Zayer et al. EG 2005 ]*



## State of the art

*[ On Linear Variational Surface Deformation Methods. M. Botsch and O. Sorkine. TVCG 2008]*