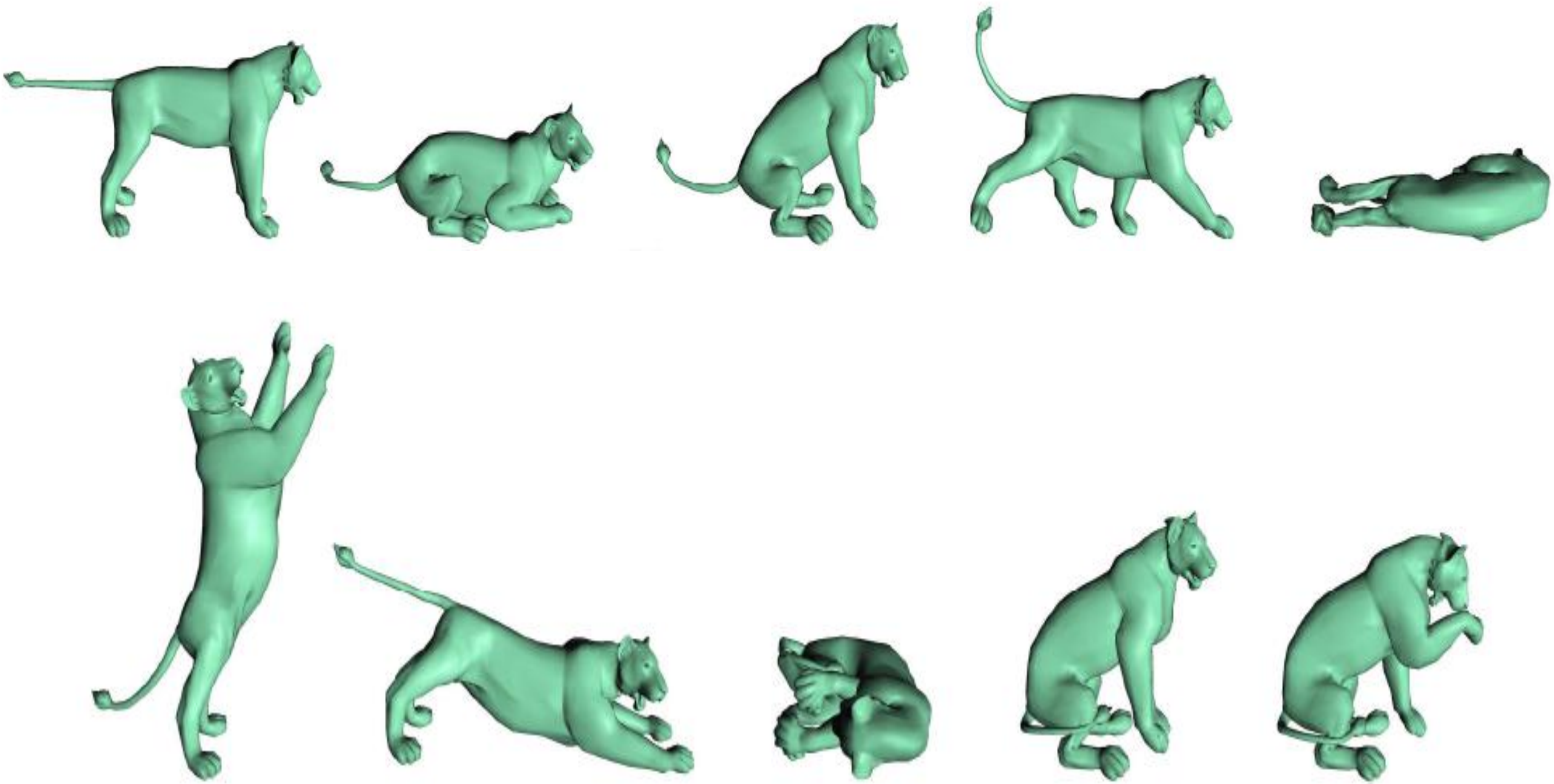
Mesh-Based Inverse Kinematics

Robert W. Sumner   Matthias Zwicker.   Jovan Popovic
MIT

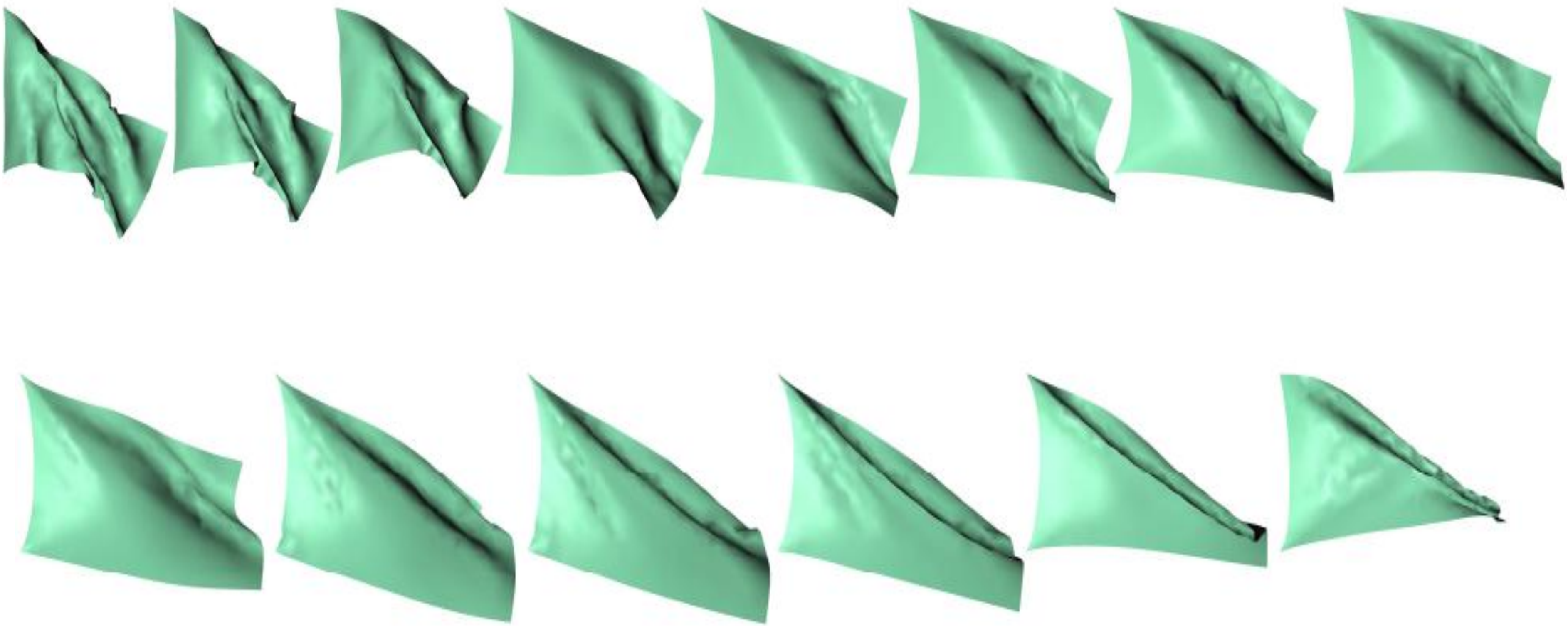
Craig Gotsman
Harvard University

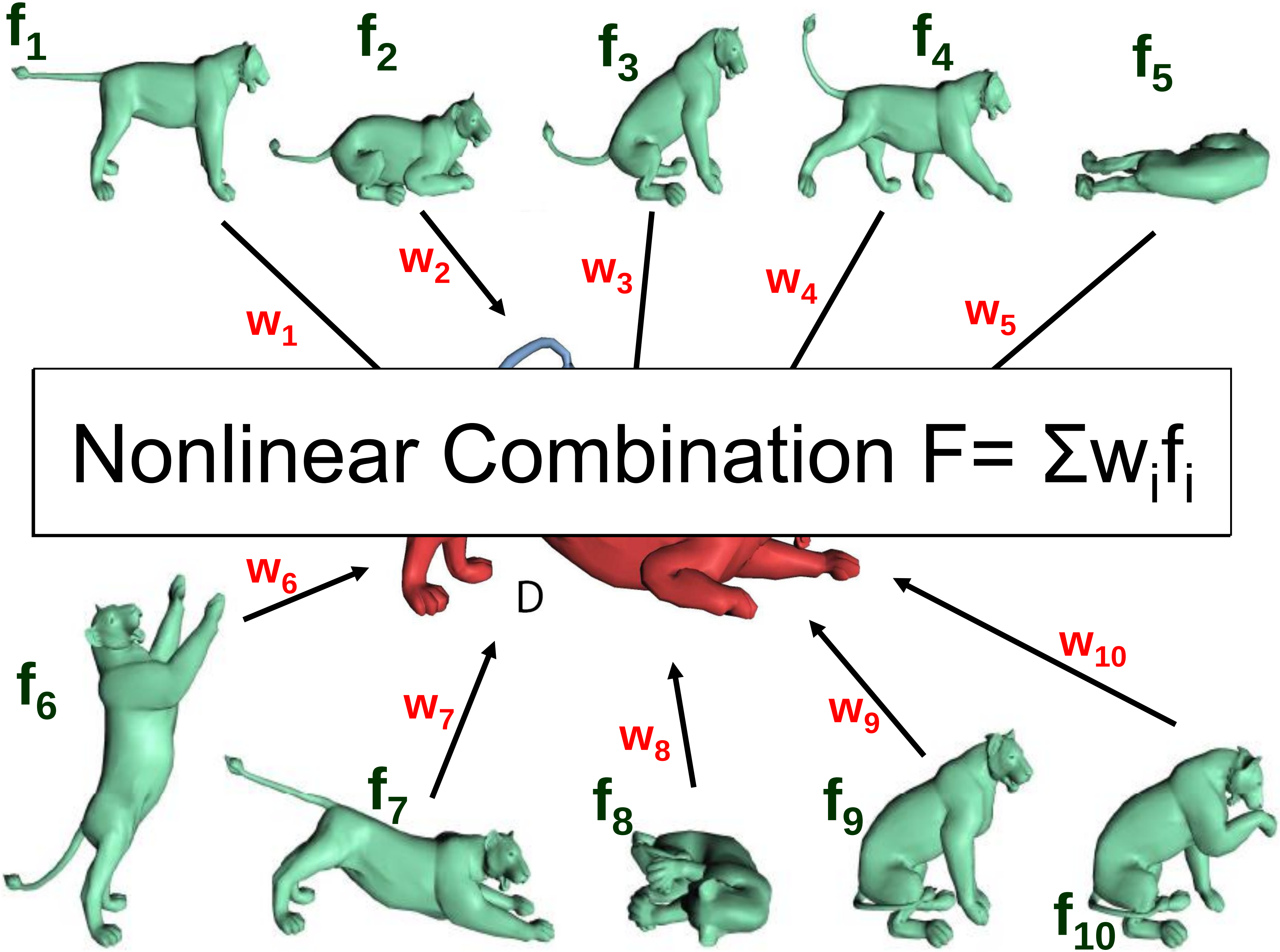

Bob Sumner
The Stata Center
32 Vassar Street 32-D532
Cambridge, MA 02139
(617) 258-5090
sumner@csail.mit.edu

Demo



Demo




f_1
f_2
f_3
f_4
f_5
w_1
w_2
w_3
w_4
w_5
Nonlinear Combination $F= \Sigma w_i f_i$
w_6
D
w_7
w_8
w_9
w_{10}
f_6
f_7
f_8
f_9
f_{10}

What is this paper about?

- New framework – Feature Vectors
  – Traditional Point-based

- Nonlinear mesh interpolation
  – Linear is good?

- Efficient Optimization

Feature Vectors

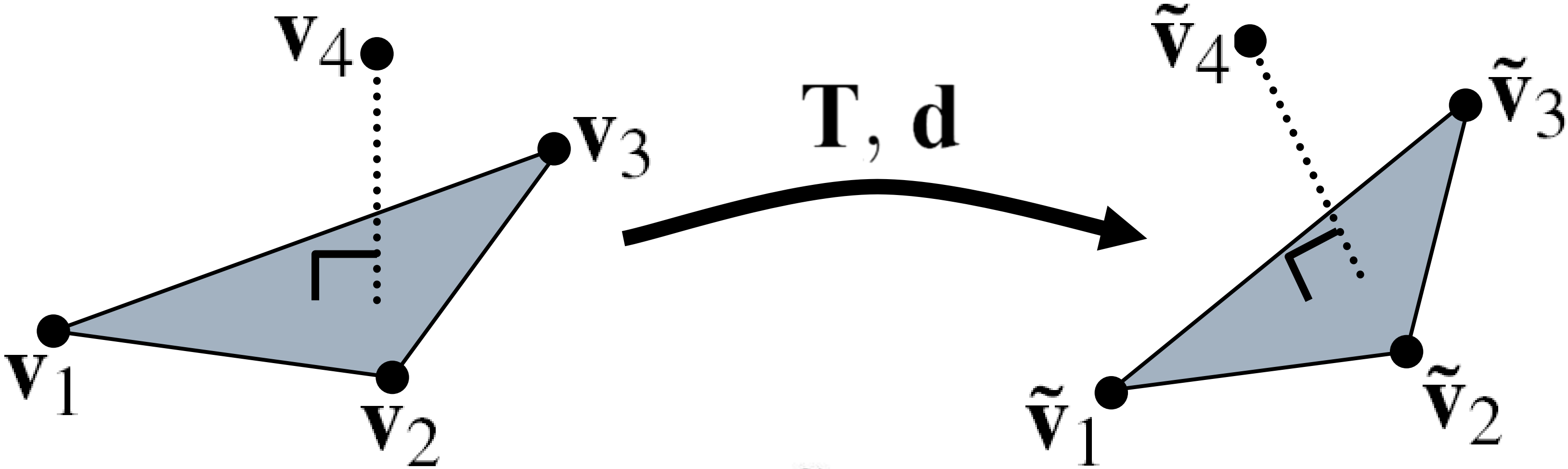
- Feature Vector = a group of deformation gradients.
- What is deformation gradient
  - affine mapping is
  - $\Phi(p) = T*p + d$, T = rotation, scaling, skewing
  - $d\Phi(p) / dp = T$ is deformation gradient

$$\begin{bmatrix} \mathbf{T} \end{bmatrix} \times \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix} + \begin{bmatrix} \mathbf{d} \end{bmatrix}$$

- So given two triangles, how to find T?

Reference | Deformed

$$\mathbf{v}_4 = \mathbf{v}_1 + (\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1) / \sqrt{|(\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1)|}$$



$$\mathbf{T}\mathbf{V} = \tilde{\mathbf{V}}$$

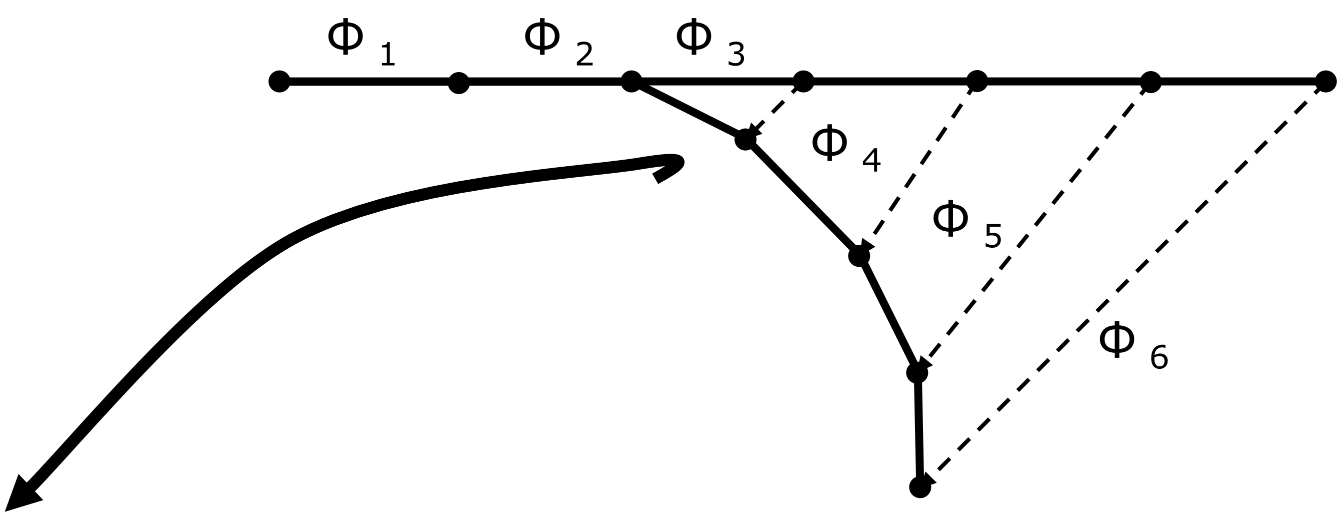
$$\mathbf{T} = \tilde{\mathbf{V}}\mathbf{V}^{-1}$$

$$\mathbf{T}(\mathbf{v}_4 - \mathbf{v}_1) = \tilde{\mathbf{v}}_4 - \tilde{\mathbf{v}}_1$$

$$\mathbf{T}\mathbf{v}_4 + \mathbf{d} = \tilde{\mathbf{v}}_4$$

Feature Vectors

- Deformation gradient of affine mapping
  - $\Phi(\mathbf{p}) = \mathbf{T*p+d}$, T = rotation, scaling, skewing
- Illustration in 2-D curve



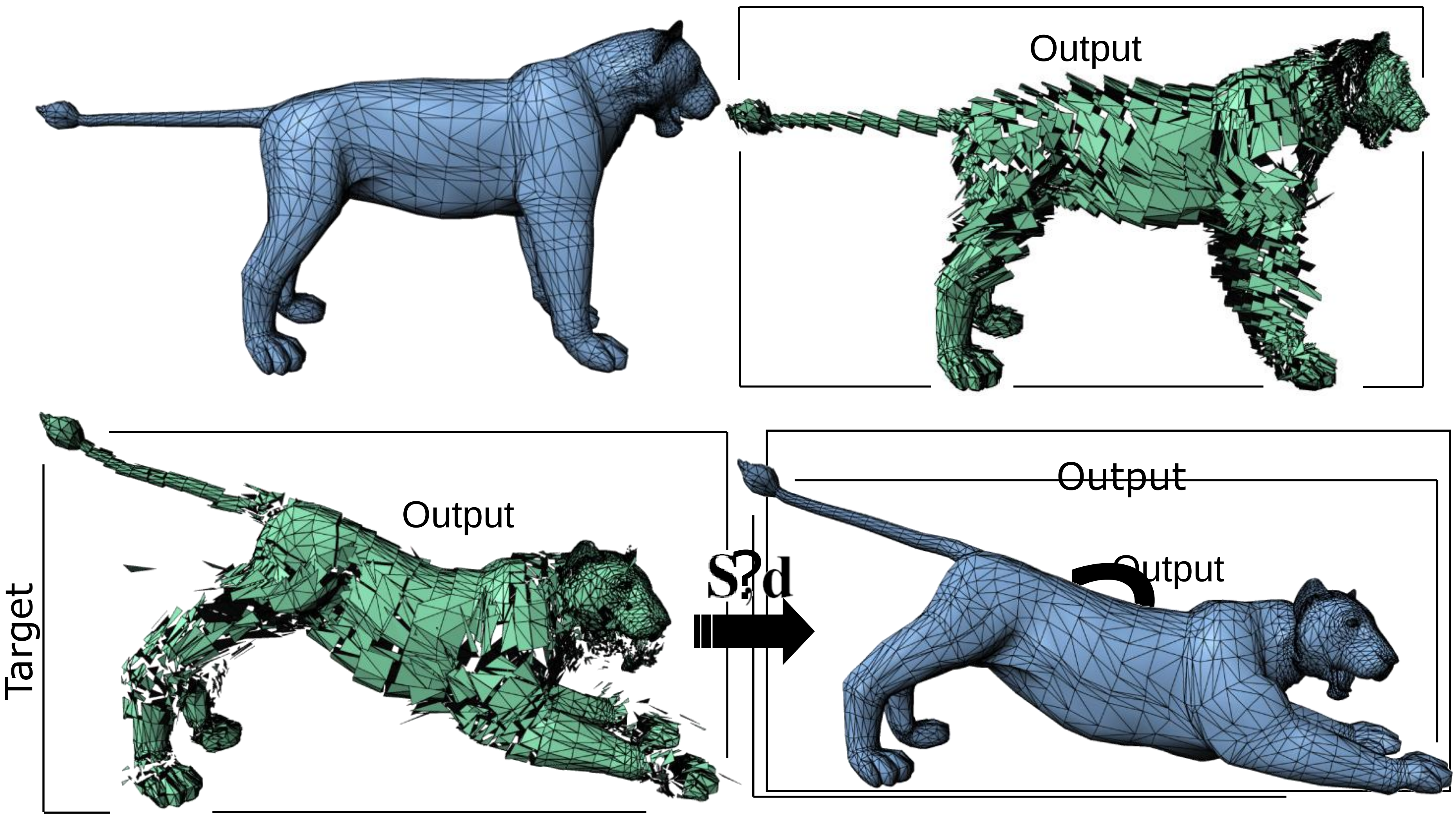
$F = [T_1, T_2, T_3, T_4, T_5, T_6]^T$

$$\mathbf{T} = \tilde{\mathbf{V}}\mathbf{V}^{-1}$$

In Mesh IK

- Yet
- Interpolation among feature vectors

In 3D

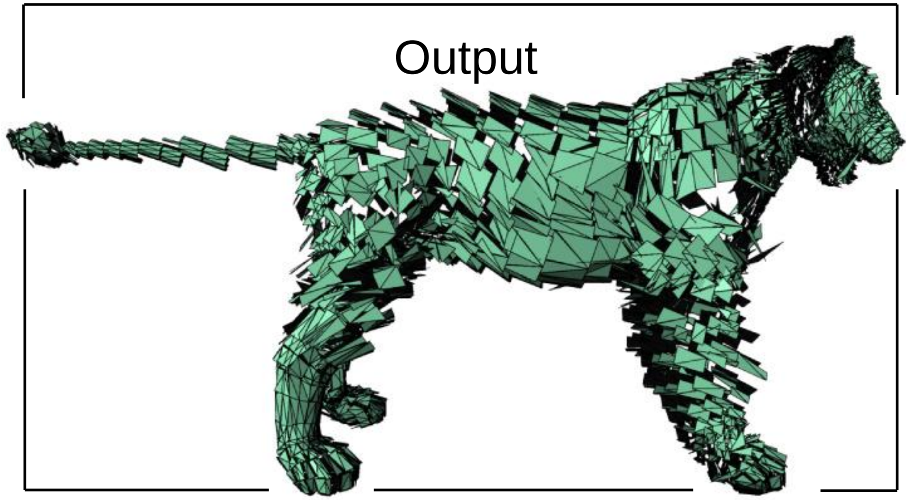


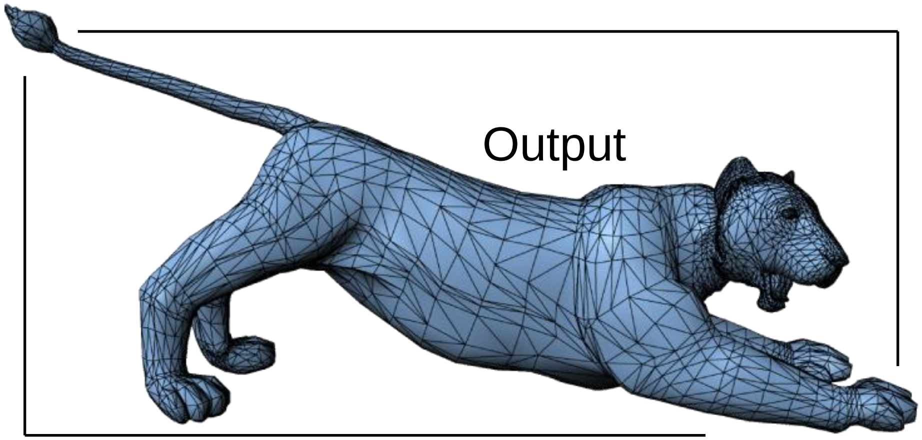
How it works?

- Actually solving for position directly
  - $x = \text{argmin}_x \| f(x) - F \|$
  - f(x) = deformation gradient using x

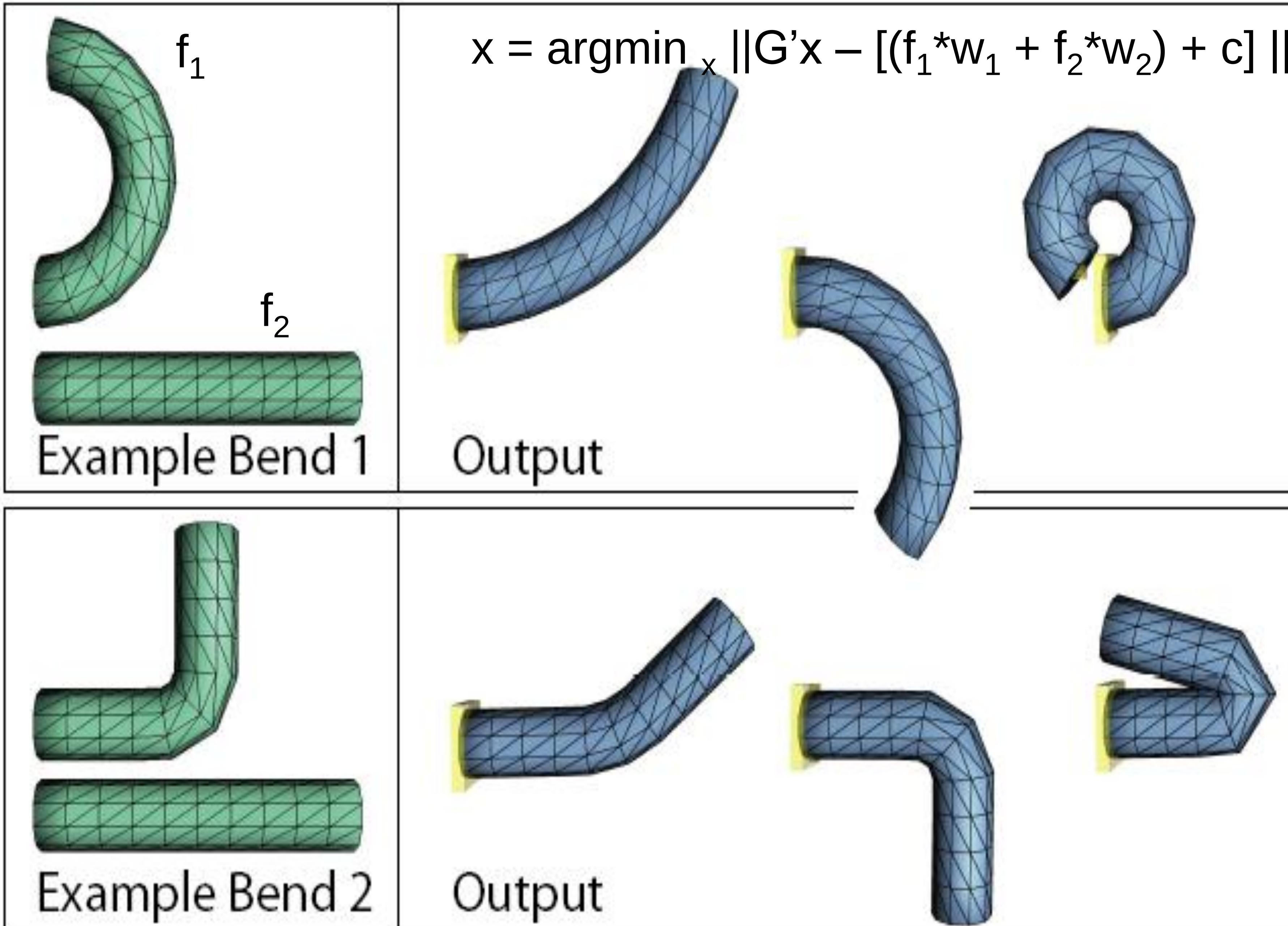
$$\mathbf{T} = \tilde{\mathbf{V}}\mathbf{V}^{-1}$$

  - $x = \text{argmin}_x \|G'x - (F + c)\|$





$$x = \text{argmin}_x \, \|G'x - [(f_1 * w_1 + f_2 * w_2) + c]\|$$

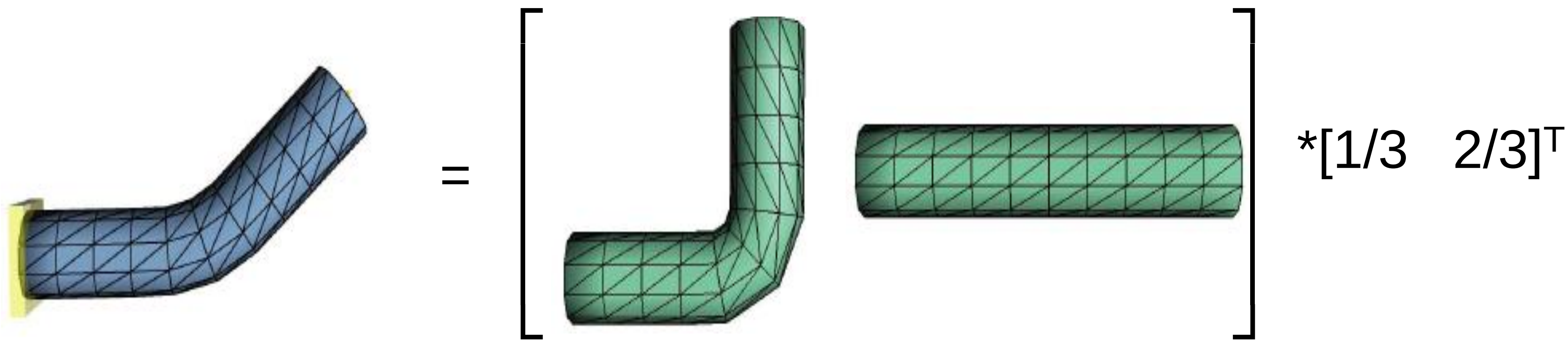


Outline

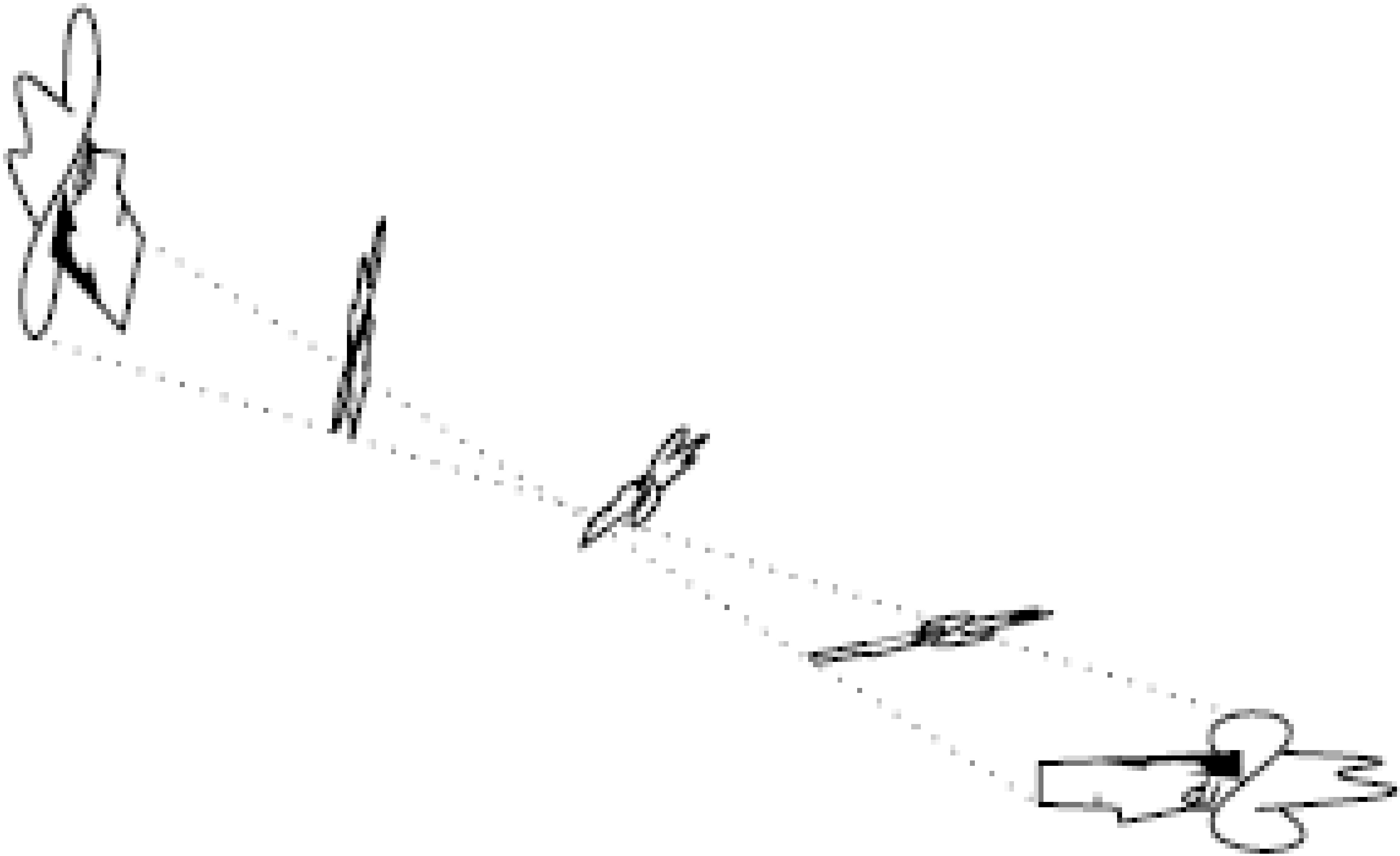
- New framework
- Non-linear mesh interpolation
  - How to interpolate linearly?
  - Polar decomposition
  - Exponential map
- Efficient Optimization

Linear Feature Space

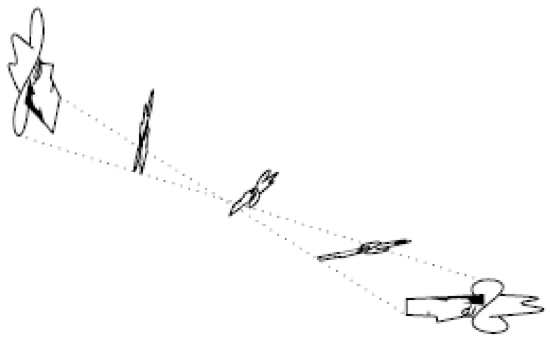
- We can find the desired mesh with feature vector $f_w = M*w$, M is $[f_1, f_2, \dots f_n]$
- $x^*, w^* = \mathrm{argmin} \, || Gx - (Mw + c) ||$
- If set $Mw = d_{avg} + \Sigma w_i d_i$
  - $x^*, w^* = \mathrm{argmin} \, || Gx - (Mw + c) || + k^* ||w||$



NonLinear versus Linear



Alternative: Nonlinear

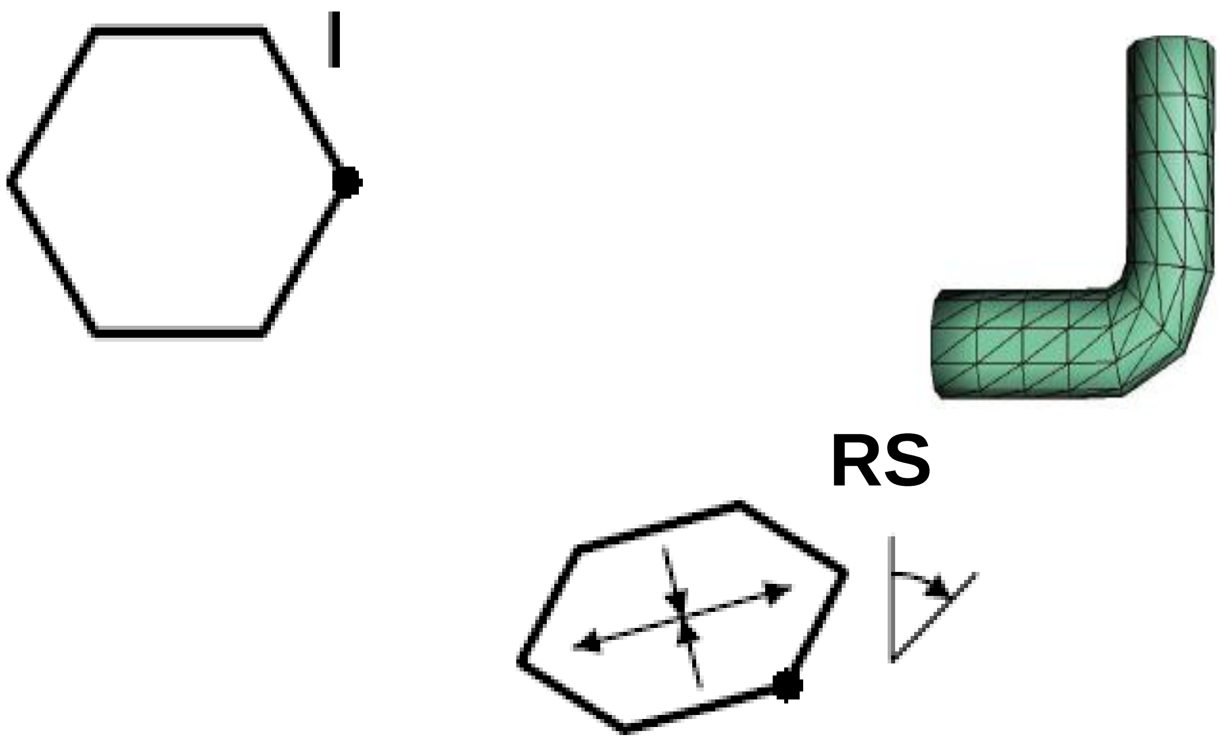


- Polar decomposition
  - rotation & scaling differently

- Exponential map
  - different interpolation

- “Matrix animation and Polar Decomposition” - Shoemake & Duff
- “Linear Combination of Transformation” - Marc Alexa

Polar Decomposition



$$= \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ . \\ . \\ . \\ T_n \end{bmatrix} = \begin{bmatrix} R_1S_1 \\ R_2S_2 \\ R_3S_3 \\ . \\ . \\ . \\ R_nS_n \end{bmatrix}$$

R must be orthogonal **=> SVD : $U\Sigma V^T$, QR : RL**

Exponential Map

T = =>

For $T \times M = T_b \times T_a \times M$ if a = b = 1/2

So ½ T x ½ T = T

=> ½ of $T = T^{½}$

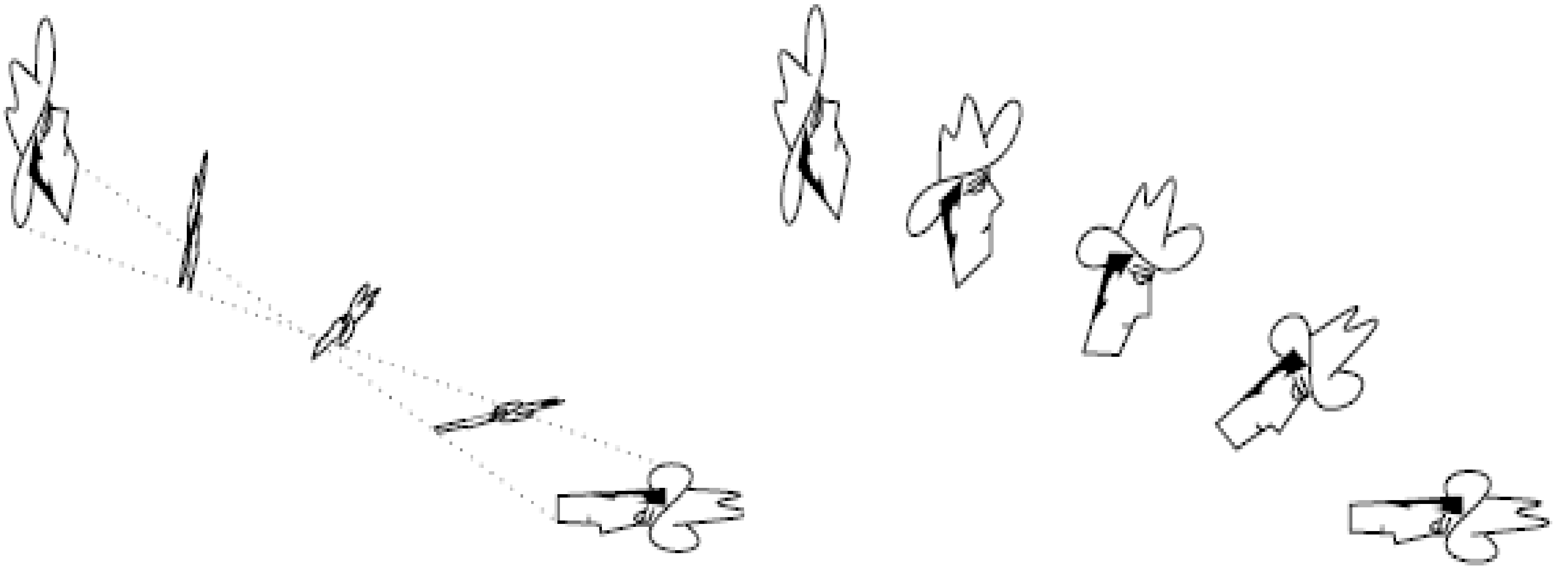
Half T = =>

AxB = BxA? (Commutative)

For $T^{½} \odot T^{½}$ $\odot$ = exp(Log(A) + Log(B))

$T^{½} \odot T^{½}$ = exp(½ Log(T) + ½ Log(T)) = T

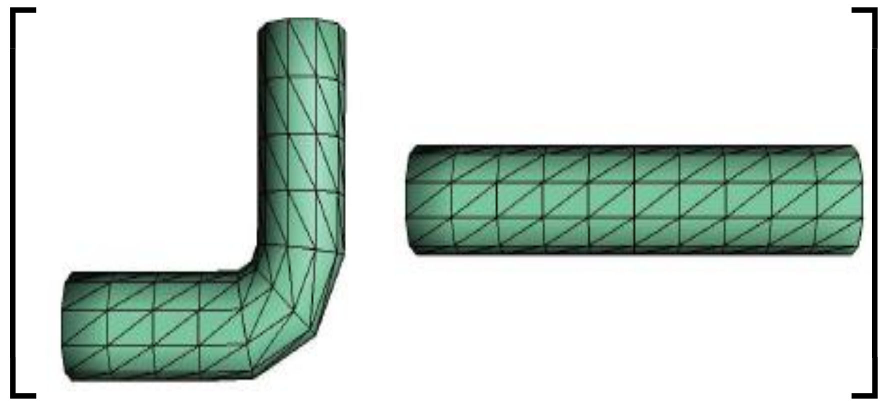
Examples



Nonlinear Feature Space

- Polar decomposition $T_j = R_j * S_{j,}$
- Exponential map
  - $T_j(w) = \exp(\Sigma w_i * \log(R_{ij})) \quad * \quad \Sigma w_i * S_{ij}$

=



$*[1/3 \quad 2/3]^T$

- We can find the desired mesh with feature vector $f_w = M*w = [f_1, f_2, \ldots f_n] * [w_1, w_2, \ldots w_n]^T$
- Defined as $f_w = [T_1(w_1), T_2(w_2), \ldots T_n(W_n)] = M(w)$

Why use exponential map?

1. Easier to find derivatives, with respect to w
   - $T_j(w) = \mathbf{R(w)} * \mathbf{S(w)}$
   - $d_{wk}T_j(w) = \mathbf{dR(w)} * \mathbf{S(w)} + \mathbf{R(w)} * \mathbf{dS(w)}$

2. Then why do we need derivatives?
   - $x^*, w^* = \text{argmin} \, || Gx - [M(w)+c] ||$
   - Gauss-Newton Algorithm
   - $M(w + \delta) = M(w) + d_w M(w) * \delta$

Gauss-Newton Method

- For k-th iteration:
- $\delta_k, x_{k+1} = \text{argmin}||Gx - d_w M(w_k)*\delta - (M(w_k)+c)||$
- $w_{k+1} = w_k + \delta_k$
- $A^T A * [x, \delta]^T = A^T(M(w_k) + c)$
- $A = \begin{bmatrix} \mathbf{G} & & & \mathbf{| - J_1} \\ & \mathbf{G} & & \mathbf{| - J_2} \\ & & \mathbf{G} & \mathbf{| - J_3} \end{bmatrix}$ $\quad J_i = d_w M(w)$
- Take about a minute or longer to solve

Outline

- New framework
- Non-linear mesh interpolation
- Efficient Optimization
  - Specialized Cholesky-Factorization

Optimized Solver

$A^T A * [x, \delta]^T = A^T(M(w_k) + c)$

- General Cholesky or QR factorization might not suffice
- The structure of $A^T A$ in previous normal equation is well defined

$$A = \begin{bmatrix} \mathbf{G} & & & \mathbf{| -J_1} \\ & \mathbf{G} & & \mathbf{| -J_2} \\ & & \mathbf{G} & \mathbf{| -J_3} \end{bmatrix}$$

$$A^T A = \begin{bmatrix} \mathbf{G^T G} & & & \mathbf{-G^T J_1} \\ & \mathbf{G^T G} & & \mathbf{-G^T J_2} \\ & & \mathbf{G^T G} & \mathbf{-G^T J_3} \\ \mathbf{-J_1^T G} & \mathbf{-J_2^T G} & \mathbf{-J_3^T G} & \mathbf{\Sigma J_i^T J_i} \end{bmatrix}$$

Precomputation

$A^TA * [x, \delta]^T = A^T(M(w_k) + c)$

$$A^TA = \begin{bmatrix} \mathbf{G^TG} & & & \mathbf{-G^TJ_1} \\ & \mathbf{G^TG} & & \mathbf{-G^TJ_2} \\ & & \mathbf{G^TG} & \mathbf{-G^TJ_3} \\ \mathbf{-J_1^TG} & \mathbf{-J_2^TG} & \mathbf{-J_3^TG} & \mathbf{\Sigma J_i^TJ_i} \end{bmatrix}$$

Make U such that $U^TU = A^TA$

$$U = \begin{bmatrix} \mathbf{R} & & & \mathbf{-R_1} \\ & \mathbf{R} & & \mathbf{-R_2} \\ & & \mathbf{R} & \mathbf{-R_3} \\ & & & \mathbf{R_s} \end{bmatrix} \qquad A = \begin{bmatrix} \mathbf{G} & & & \mathbf{|\ -J_1} \\ & \mathbf{G} & & \mathbf{|\ -J_2} \\ & & \mathbf{G} & \mathbf{|\ -J_3} \end{bmatrix}$$

where $R^TR = G^TG$, this R can be pre-computed

Solving for

$$A^TA = \begin{bmatrix} \mathbf{G^TG} & & & \mathbf{-\ G^TJ_1} \\ & \mathbf{G^TG} & & \mathbf{-\ G^TJ_2} \\ & & \mathbf{G^TG} & \mathbf{-\ G^TJ_3} \\ \mathbf{-\ J_1^TG} & \mathbf{-\ J_2^TG} & \mathbf{-\ J_3^TG} & \mathbf{\Sigma J_i^TJ_i} \end{bmatrix} =$$

$$U^TU = \begin{bmatrix} \mathbf{R^TR} & & & \mathbf{-\ R^TR_1} \\ & \mathbf{R^TR} & & \mathbf{-\ R^TR_2} \\ & & \mathbf{R^TR} & \mathbf{-\ R^TR_3} \\ \mathbf{-\ R_1^TR} & \mathbf{-\ R_2^TR} & \mathbf{-\ R_3^TR} & \mathbf{\Sigma R_i^TR_i + R_s^TR_s} \end{bmatrix}$$

1. Solve R_i ,where $R^TR_i = G^TJ_i$, $1 <= i <= 3$
2. Solve R_s ,where $R_s^TR_s = \mathbf{\Sigma}\ J_i^TJ_i - R_i^TR_i$
3. The bottleneck for MeshIK

Numerical Result

Mesh	Verts	Tris	Ex	Factor	Solve	Total
Bar	132	260	2	0.000	0.000	0.015
Flag	516	932	14	0.016	0.015	0.020
Lion	5,000	9,996	10	0.475	0.150	0.210
Horse	8,425	16,846	4	0.610	0.105	0.160
Elephant	42,321	84,638	4	13.249	0.620	0.906