



MAX-PLANCK-GESELLSCHAFT

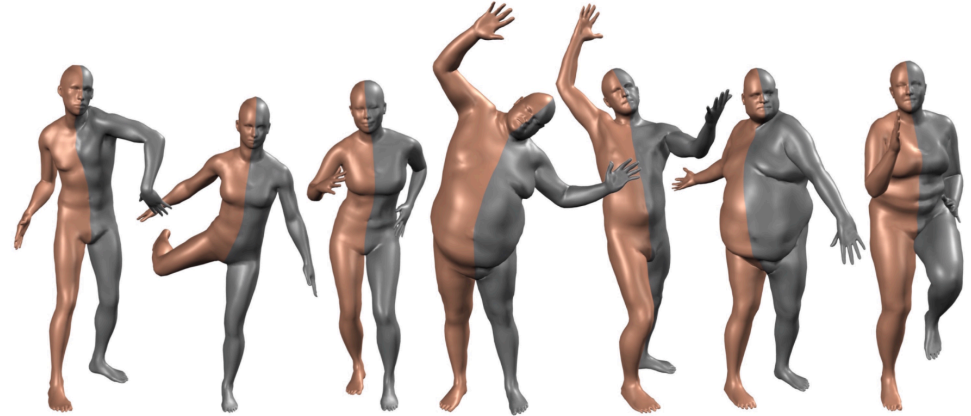
Body Models II



mpi max planck institut
informatik



UNIVERSITÄT
DES
SAARLANDES



Gerard Pons-Moll and Bernt Schiele

Max Planck Institut für Informatik

January 08, 2018

Schedule

Intro - Probabilities

Graphical Models Basics

Factor Graphs & Sum Product

Max-Product and Body Pose

Image Processing

Sampling & Tracking

Parameter Estimation

Junction Tree and Causality

Body Models 1

08.01.2018 **Body Models 2**

15.01.2018 Body Models 3

22.01.2018 Stereo & Graph Cut

29.01.2018 Optical Flow

Our research goal: Virtual humans

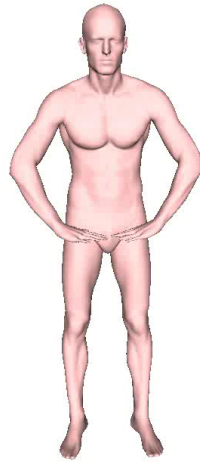


What is a virtual human model?

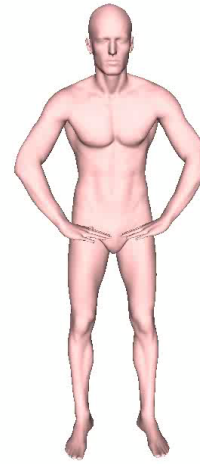
3D scan with
texture



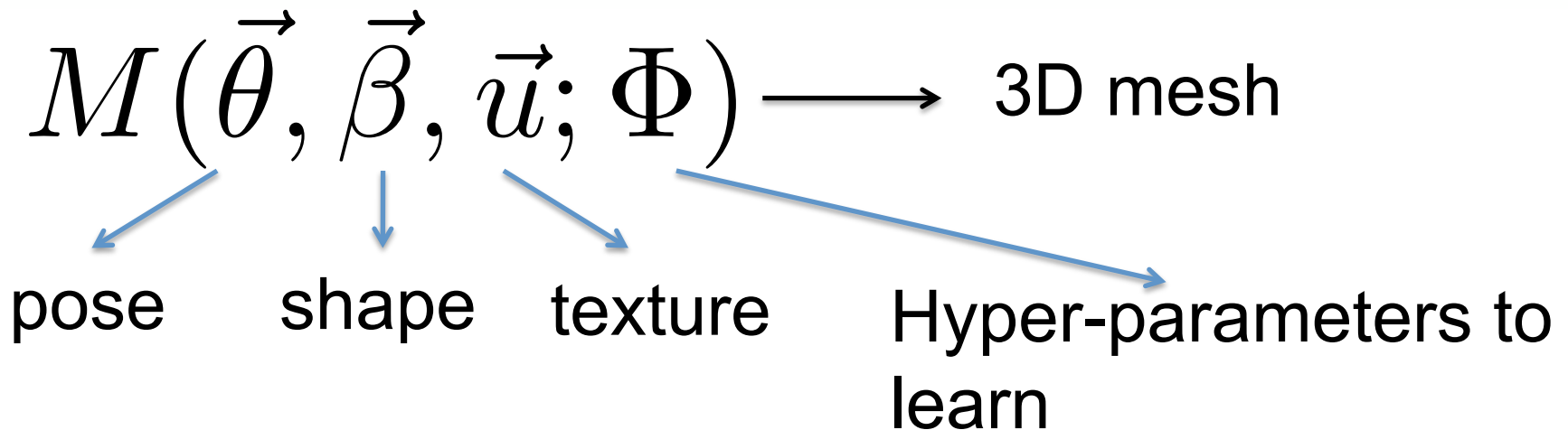
Ground truth
shape



Model



Model with
texture



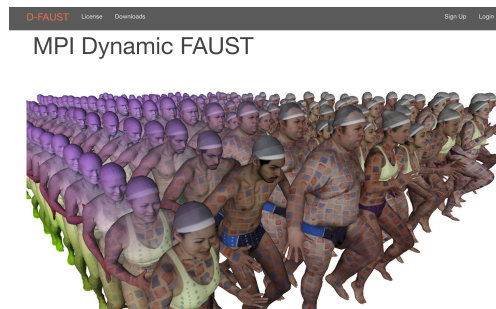
Applications



Tracking from depth



Virtual Reality



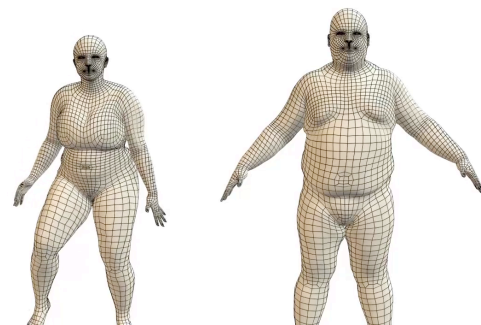
Registration



Cloth modeling and try-on

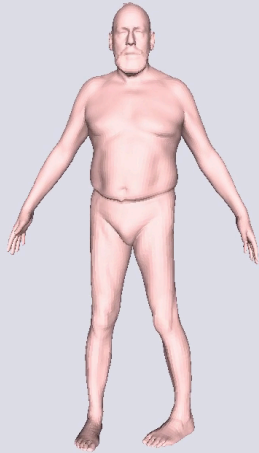


Tracking from images

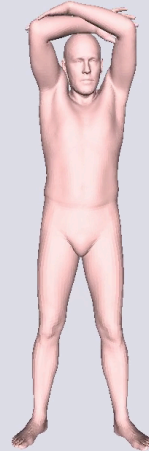


Animation

A Body Model is a function



$$M(\mathbf{0}, \mathbf{X}_{\text{shape}})$$



$$M(\mathbf{X}_{\text{pose}}, \mathbf{0})$$



$$M(\mathbf{X}_{\text{pose}}, \mathbf{X}_{\text{shape}})$$



$$R \cdot M(\mathbf{X}_{\text{pose}}, \mathbf{X}_{\text{shape}})$$



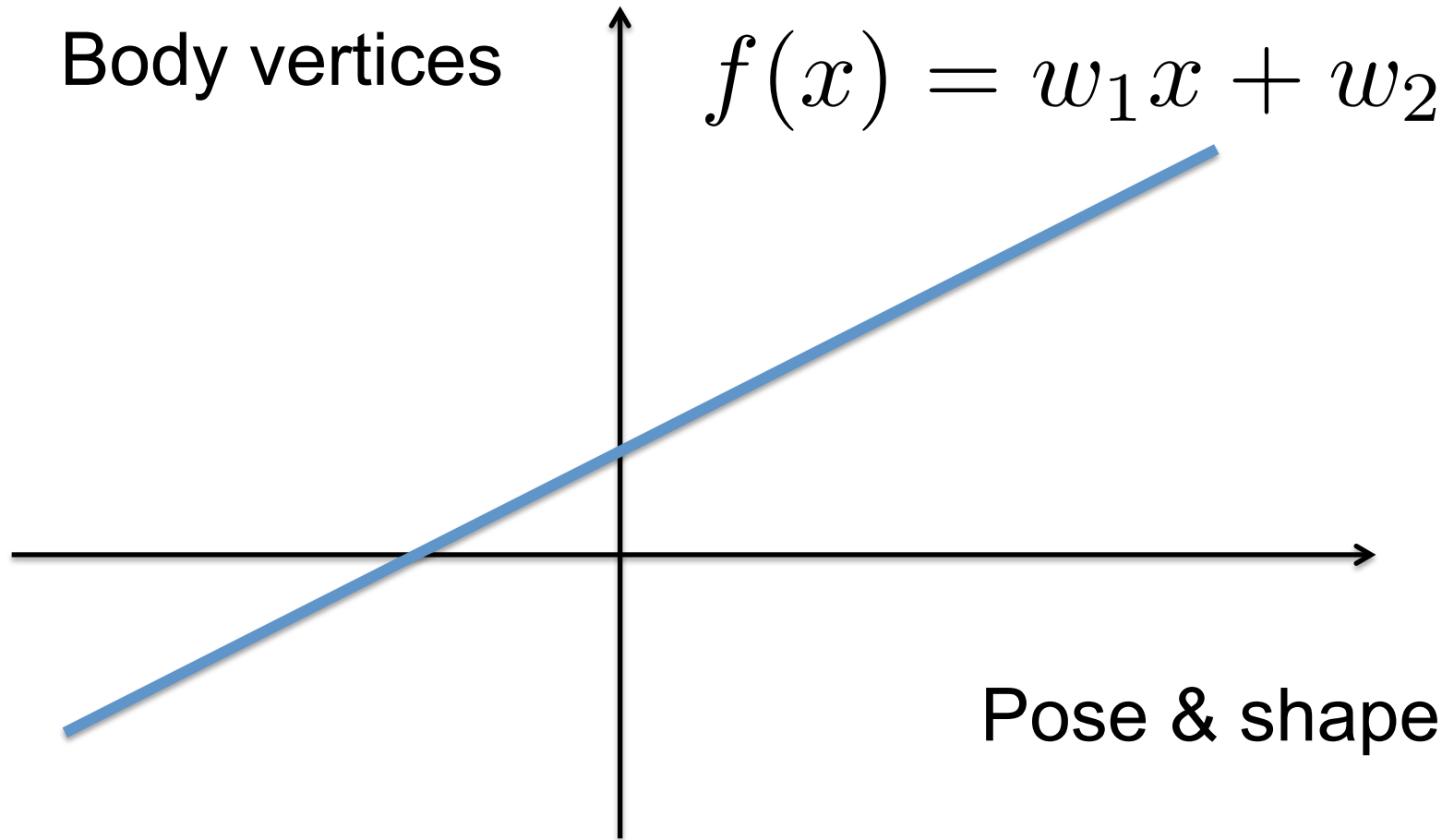
$$M(\mathbf{X}_{\text{pose}}, \mathbf{X}_{\text{shape}})$$



Y

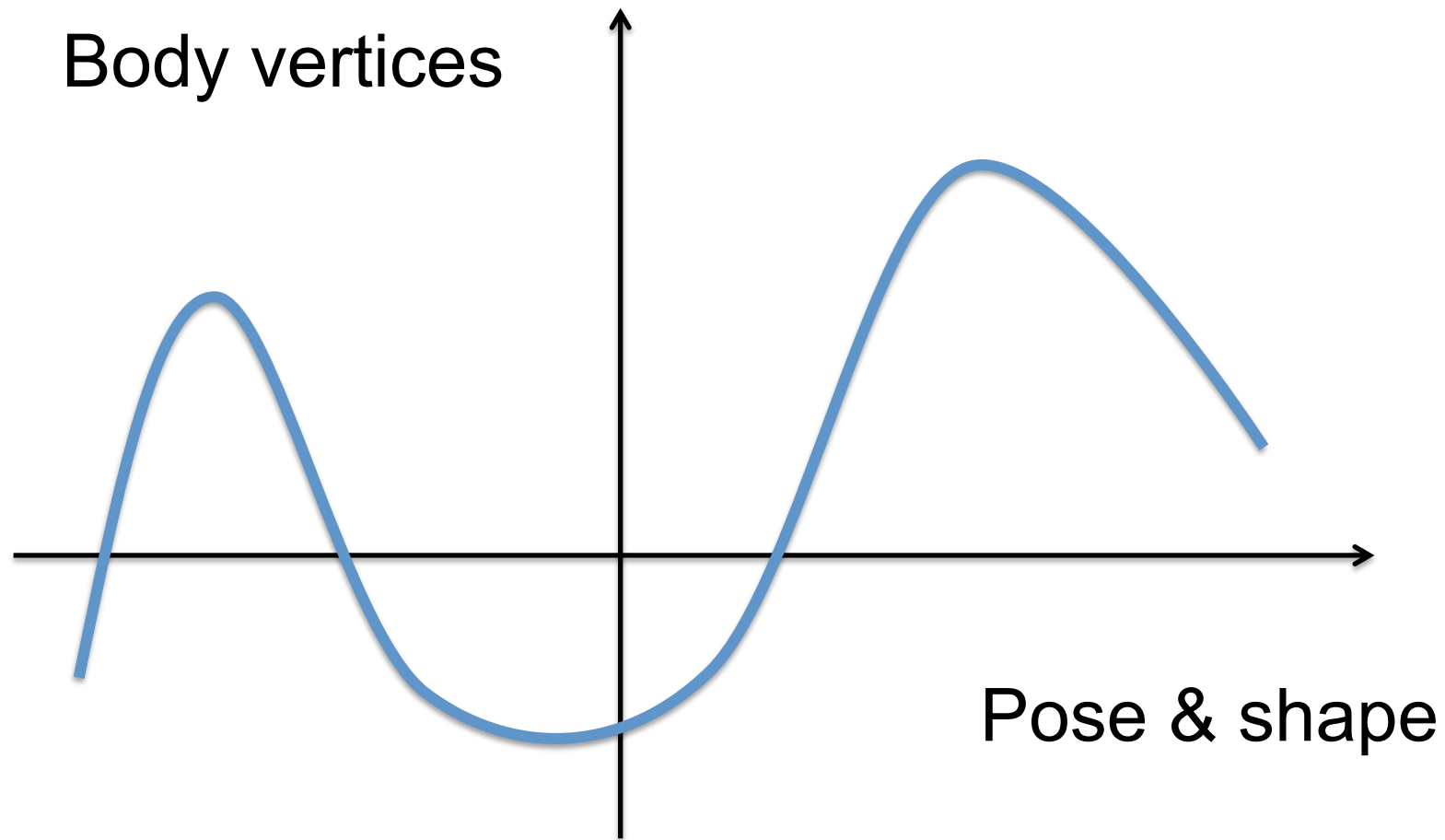
$$\mathbf{X} = \{\mathbf{X}_{\text{pose}}, \mathbf{X}_{\text{shape}}\}$$

What kind of function ?



Linear ?

What kind of function ?



Polynomial ?

Given the function, what \mathbf{w} ?

$$f(x; \mathbf{w}) = w_1 x^3 + w_2 x^2 + w_1 x + w_0$$

$$f(x; \mathbf{w})$$



Input parameters

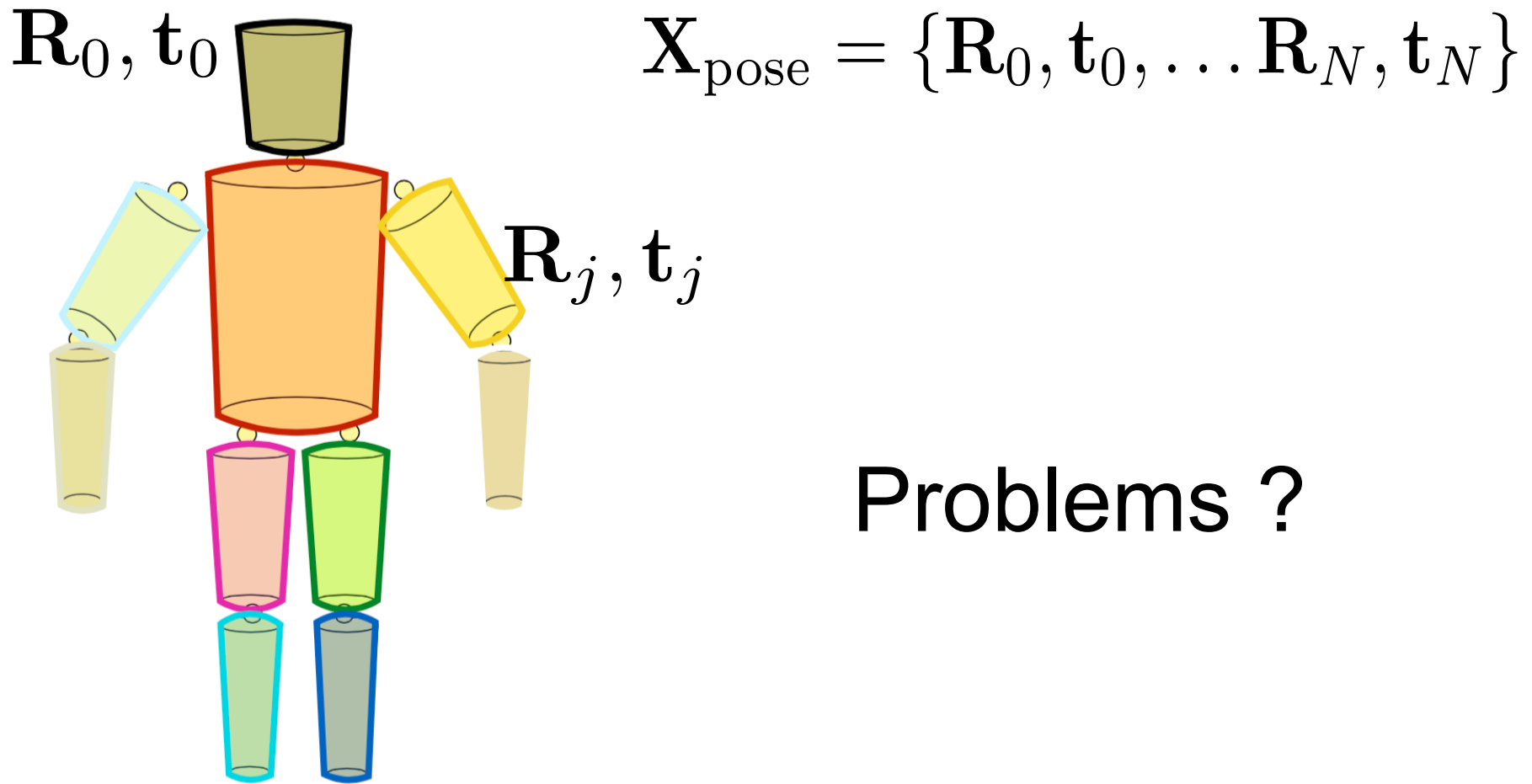
Hyper-parameters

And also why our input \mathbf{X} is
shape and pose ?

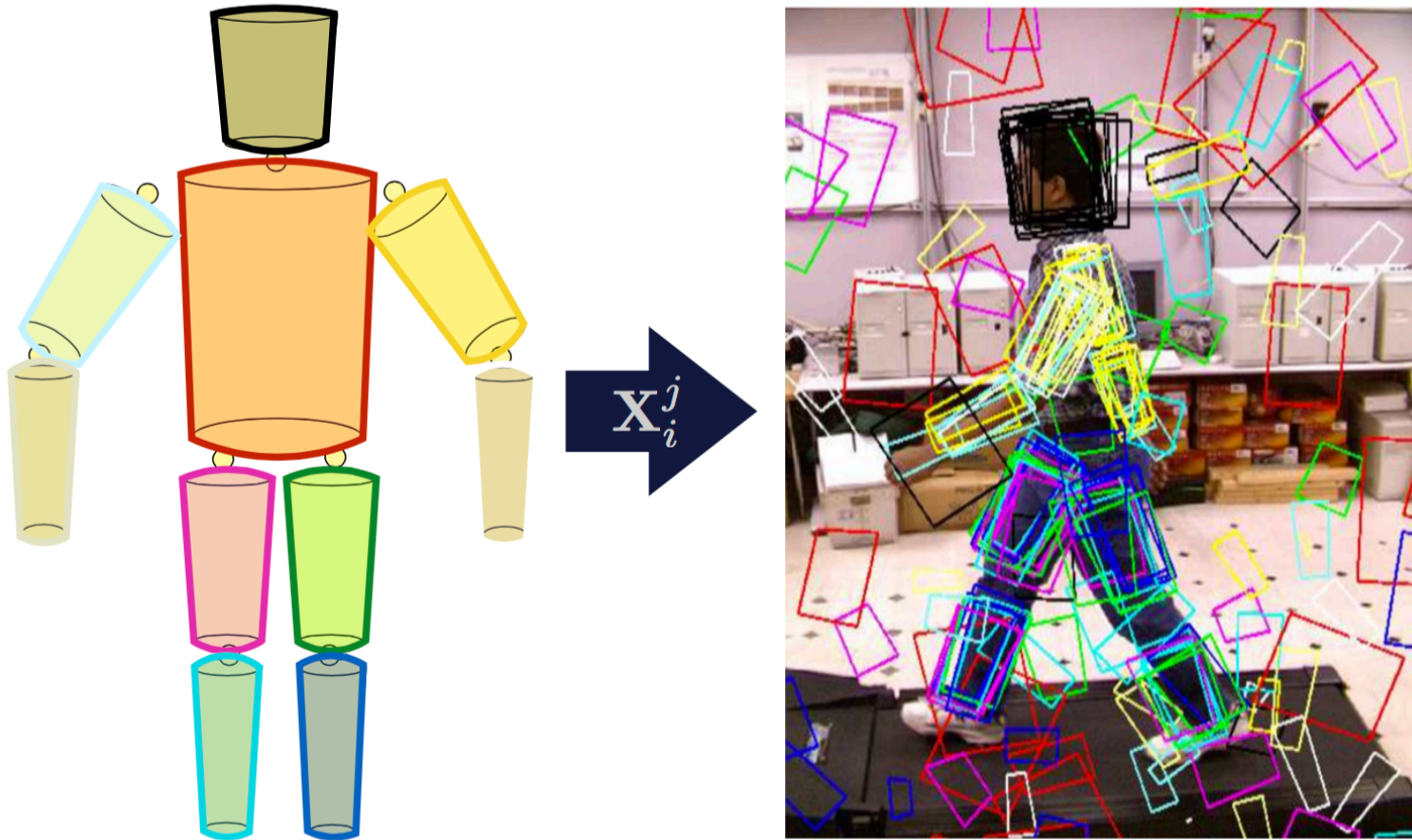
Notation: $\mathbf{X}_{\text{pose}} = \vec{\theta}$ $\mathbf{X}_{\text{shape}} = \vec{\beta}$

How do we parameterize pose ?

- Parameterize every body part separately ?



How do we parameterize pose?



Articulated constraints not satisfied!

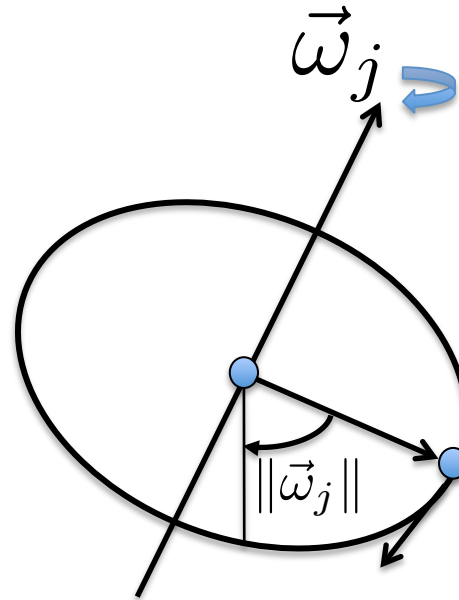
Rotation parameterization

- Rotations are composed of 9 numbers
- **6 additional constraints** to ensure that the matrix is orthonormal
- **Suboptimal** for optimization

Rotation with Exponential Maps

$\|\vec{\omega}_j\|$: Angle of rotation

$\vec{\omega}_j$: scaled axis of rotation

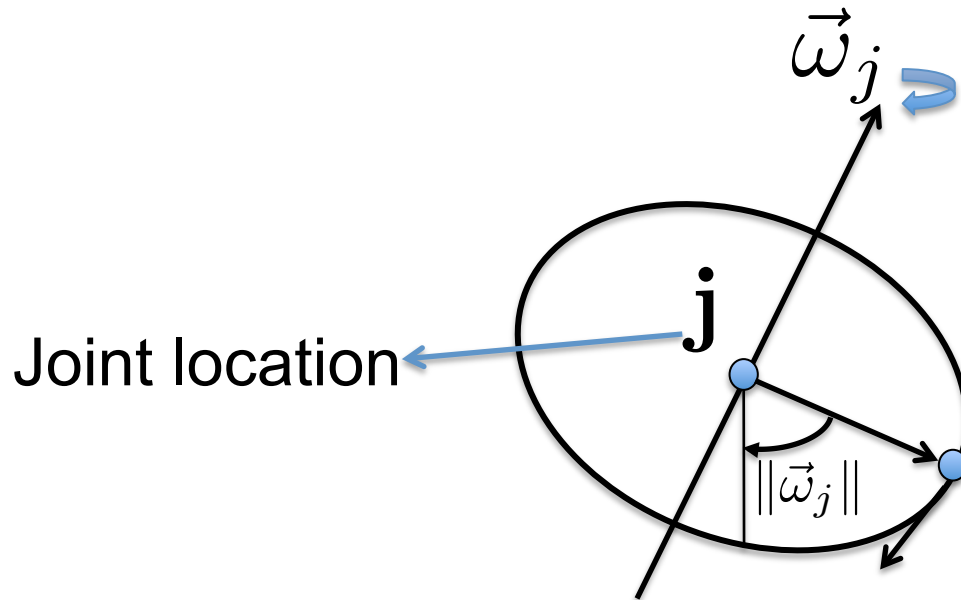


Rotation obtained with Rodrigues formula:

$$\mathbf{R} = e^{\hat{\omega}} = \mathcal{I} + \hat{\omega}_j \sin(\|\vec{\omega}_j\|) + \hat{\omega}_j^2 (1 - \cos(\|\vec{\omega}_j\|))$$

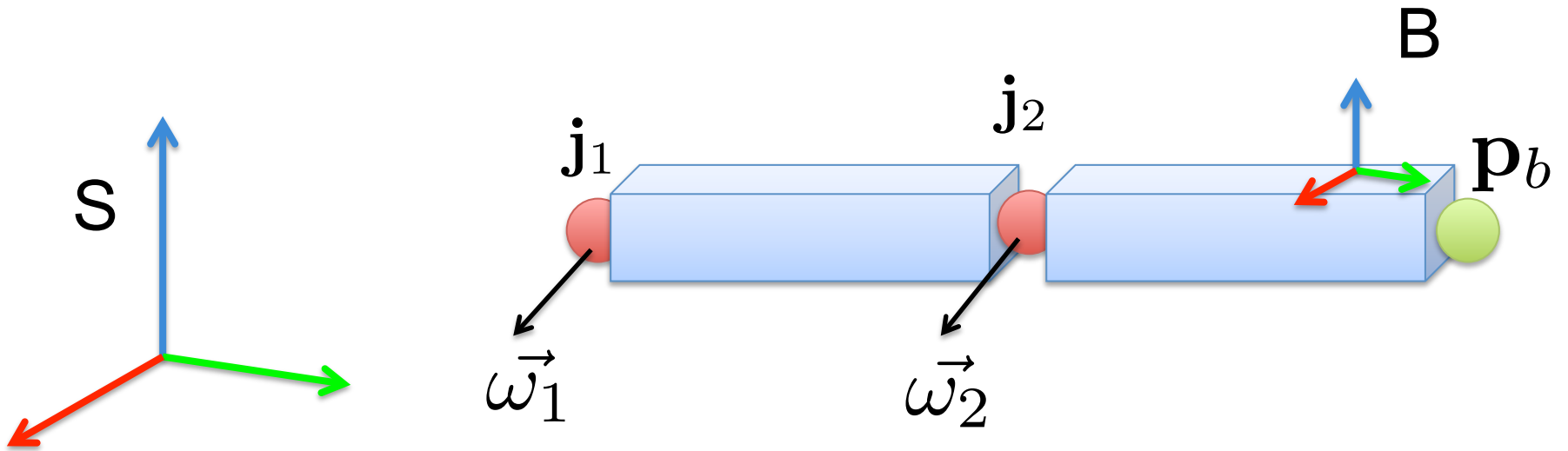
Joint Rigid Body Motion

The transformation associated with a rotational joint is

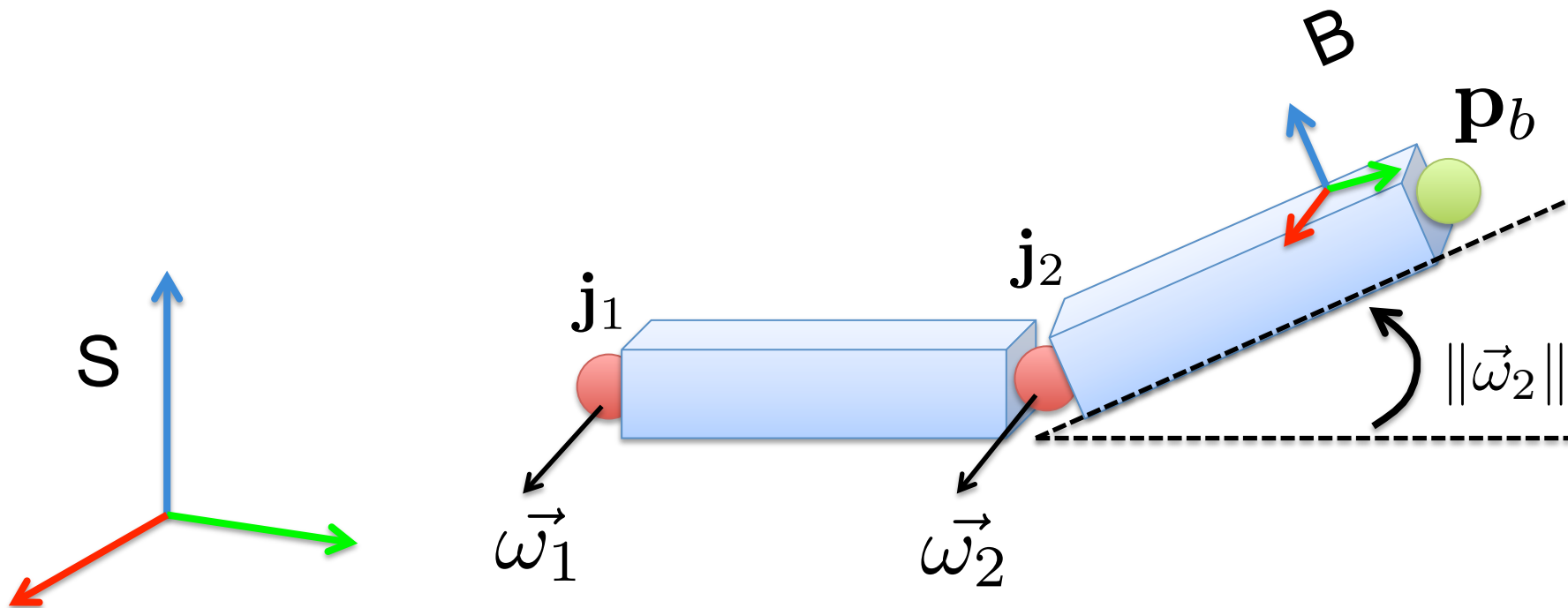


$$G(\vec{\omega}, \mathbf{j}) = \begin{bmatrix} [e^{\vec{\omega}}]_{3 \times 3} & \mathbf{j}_{3 \times 1} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \longrightarrow \text{Rigid Body Motion}$$

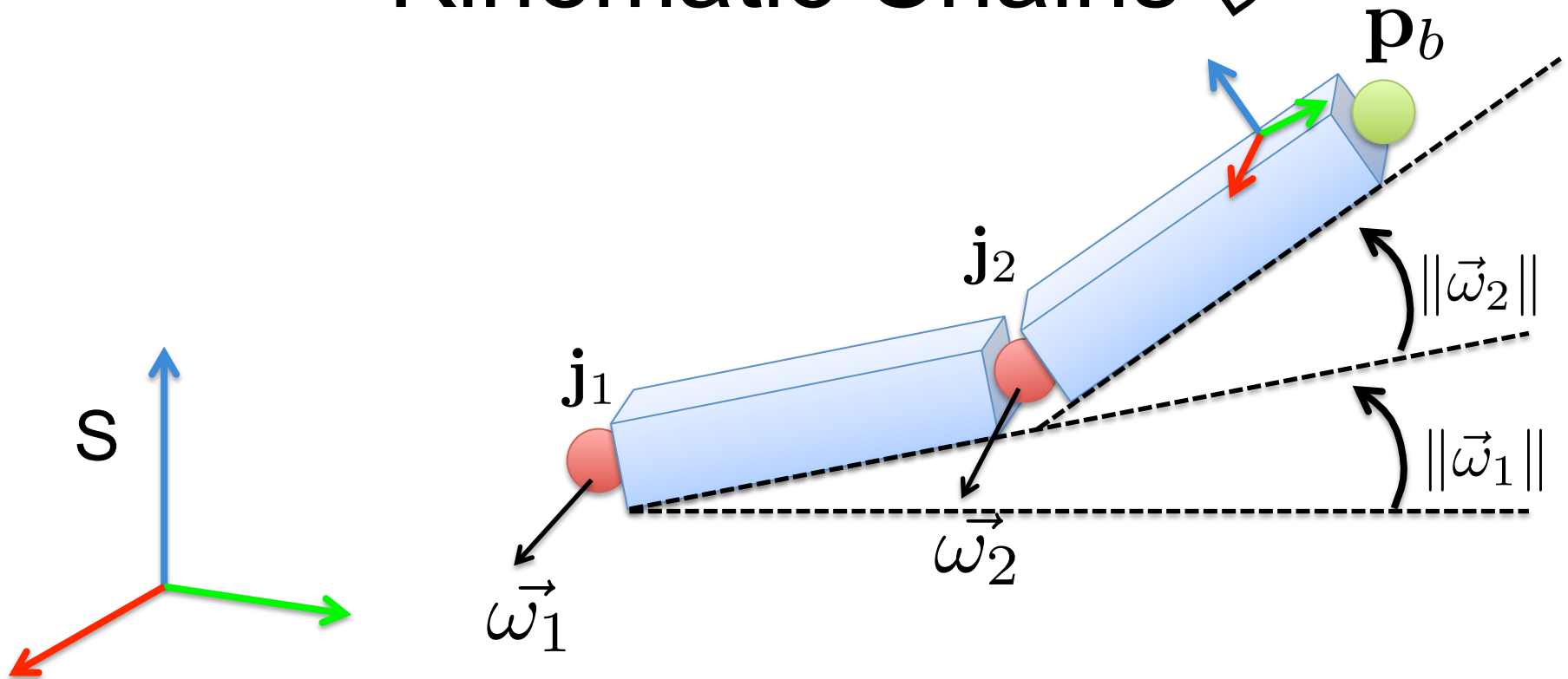
Kinematic Chains



Kinematic Chains



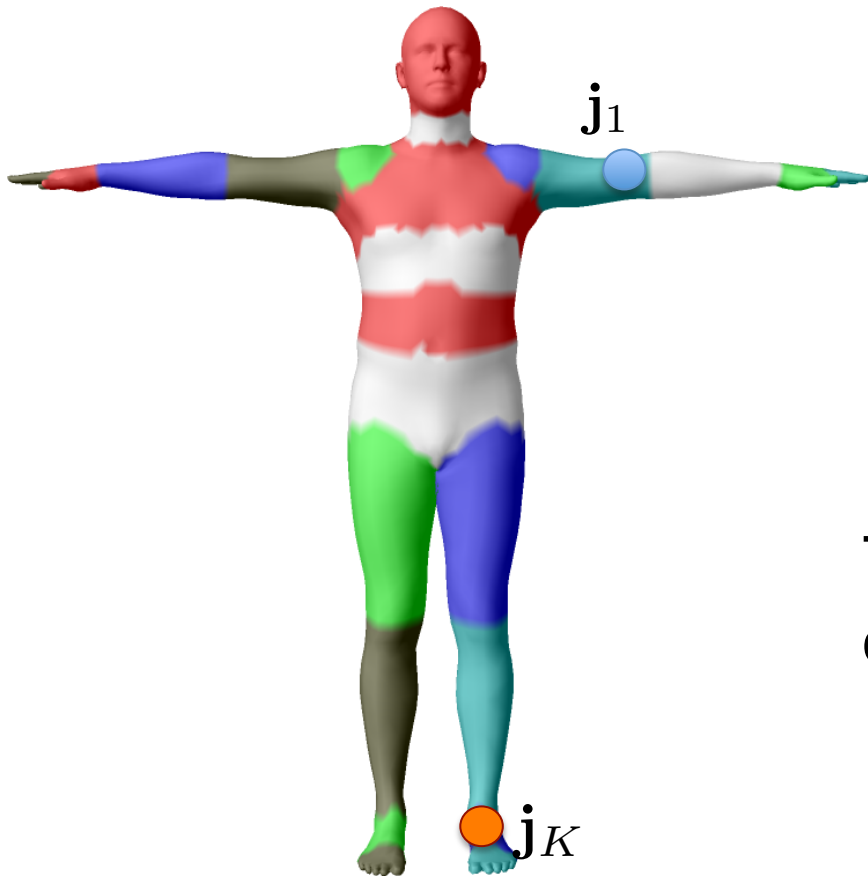
Kinematic Chains \mathcal{B}



The coordinates of the point in the spatial frame are:

$$\bar{p}_s = G(\vec{\omega}_1, \vec{\omega}_2, \mathbf{j}_1, \mathbf{j}_2) = G(\vec{\omega}_1, \mathbf{j}_1) G(\vec{\omega}_2, \mathbf{j}_2) \bar{p}_b$$

Pose Parameters



T

Given a set of joint locations

$$\mathbf{J} = (\underline{j_1}, \dots, \underline{j_K})^T$$

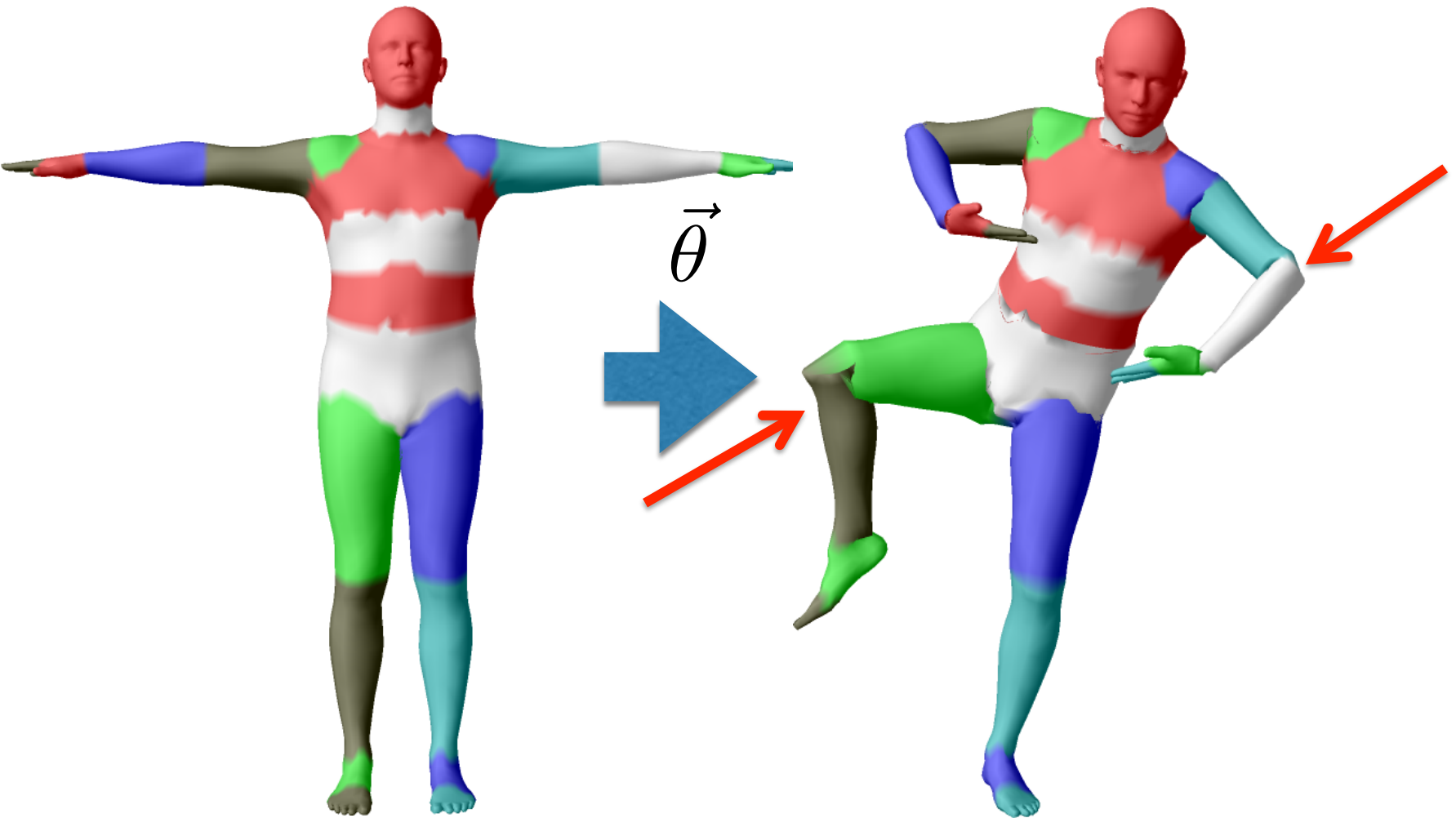
The pose defined as the vector of concatenated part axis-angles

$$\vec{\theta} = (\underline{\vec{\omega}_1}, \dots, \underline{\vec{\omega}_k})^T$$

Pons-Moll & Rosenhahn 2011

Model-based Pose Estimation. Looking at People.

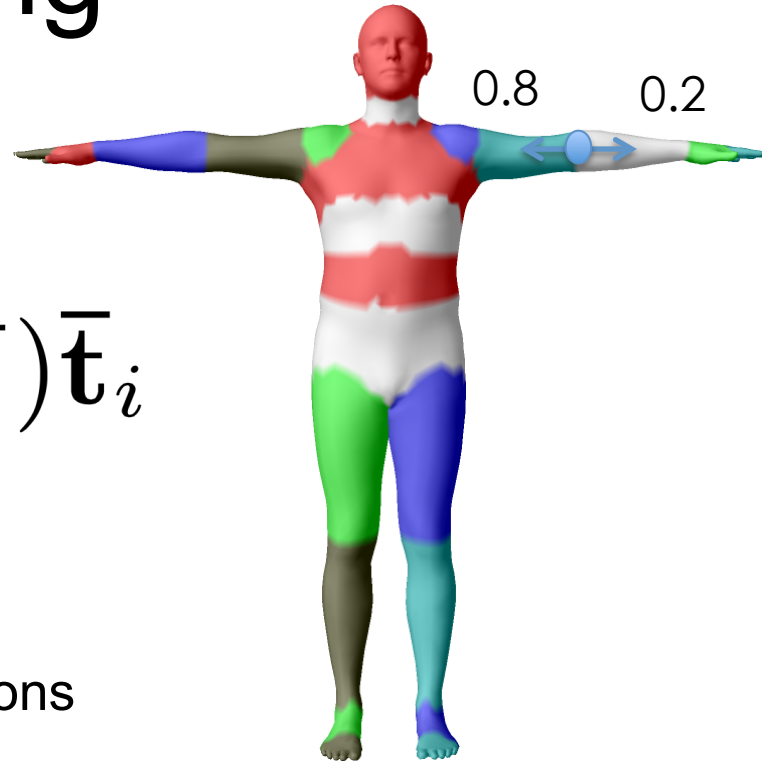
Kinematic Chain Problems



Different poses

- Different poses using no blendweights
>>python visualize_ablated_smpl.py

Linear Blend Skinning



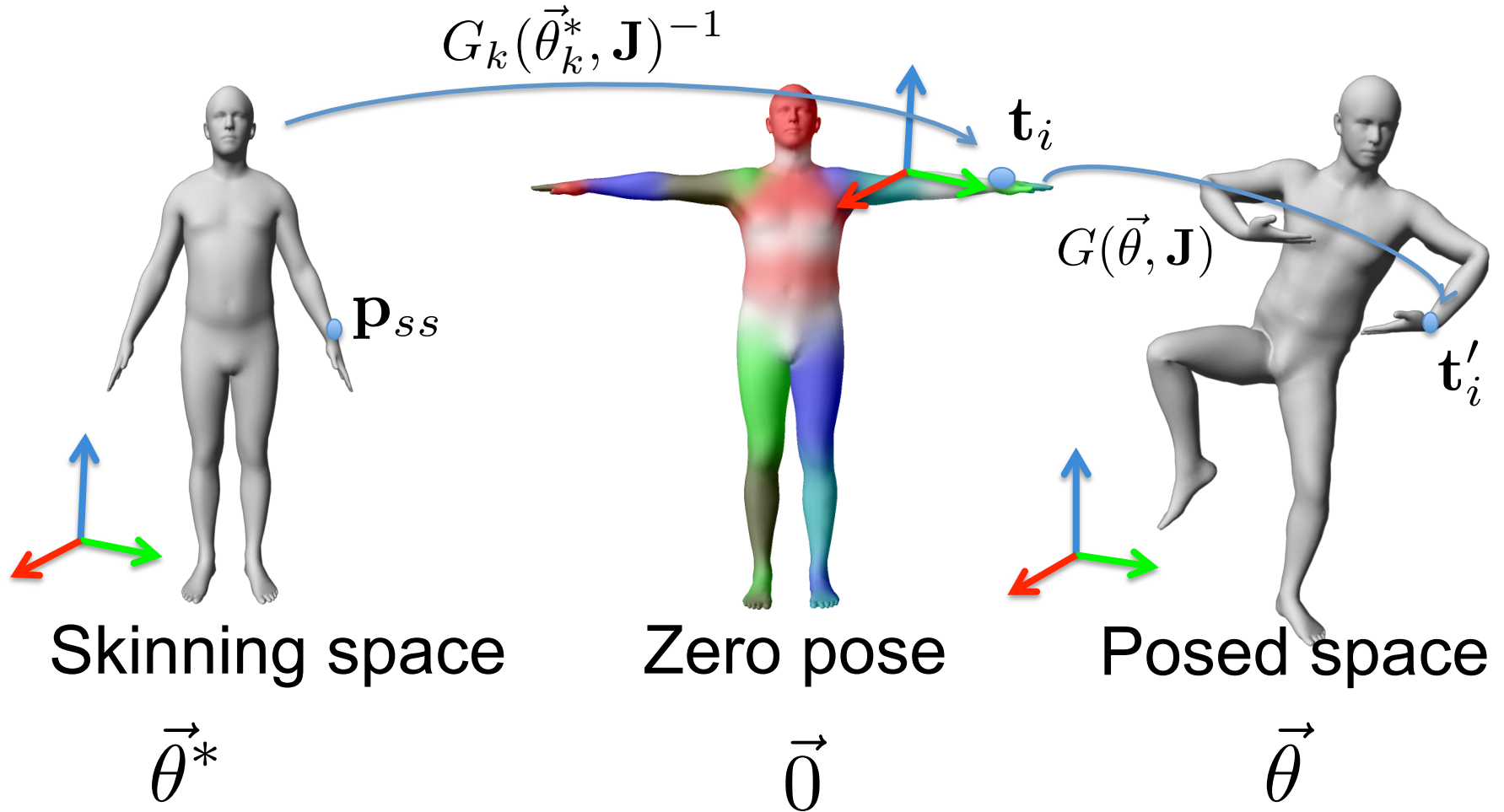
$$\bar{\mathbf{t}}'_i = \sum_{k=1}^K w_{k,i} G'_k(\vec{\theta}, \mathbf{J}) \bar{\mathbf{t}}_i$$

Blend weights

Part transformations

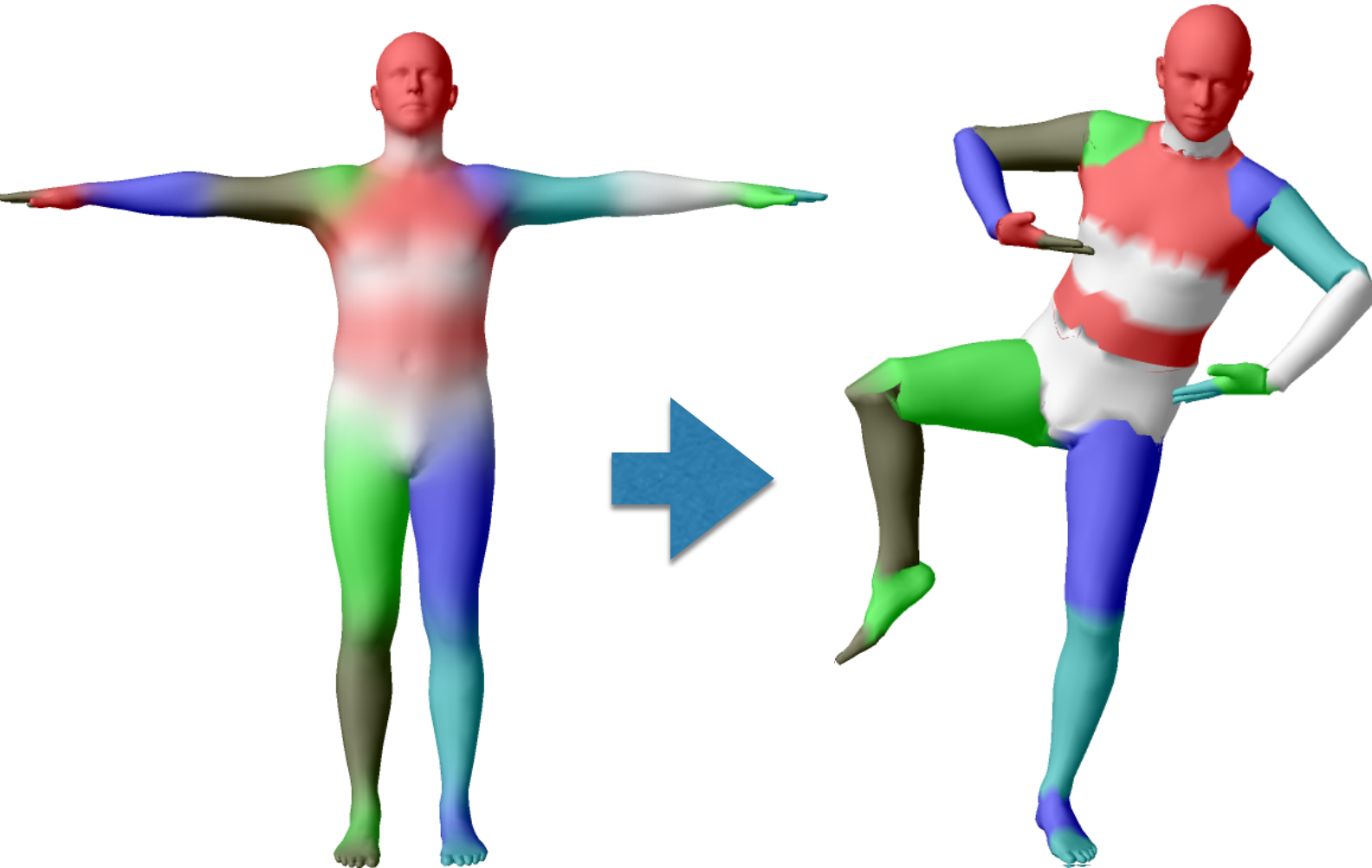
Points transformed as blended linear combination of joint transformation matrices

Binding Matrices



$$\bar{\mathbf{t}}_i = G_k(\vec{\theta}_k^*, \mathbf{J})^{-1} \mathbf{p}_{ss} \quad \rightarrow \quad G'_k(\vec{\theta}, \mathbf{J}) = G_k(\vec{\theta}, \mathbf{J}) G_k(\vec{\theta}^*, \mathbf{J})^{-1}$$

Linear Blend Skinning



Different poses using BW

- Different poses using no blendweights
>>python visualize_ablated_smpl.py

Standard Skinning

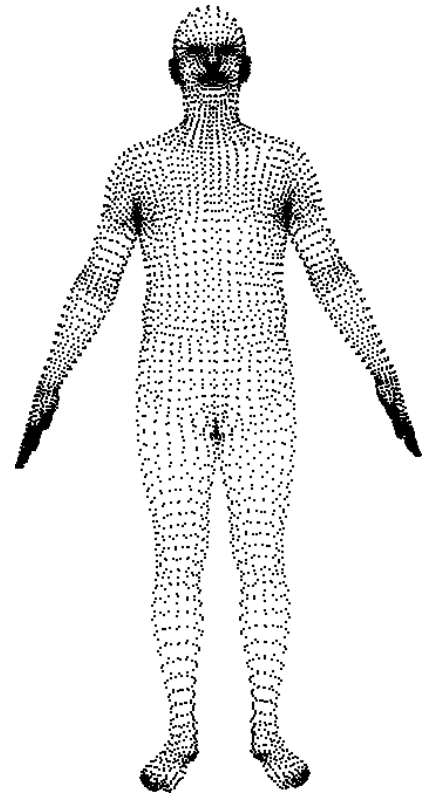
Standard skinning produces vertices from...

– Rest pose vertices: $\mathbf{T} \in \mathbb{R}^{3N}$

– Joint locations: $\mathbf{J} \in \mathbb{R}^{3K}$

– Weights: $\mathcal{W} \in \mathbb{R}^{N \times K}$

– Pose parameters: $\vec{\theta} \in \mathbb{R}^{3K}$



Standard Skinning

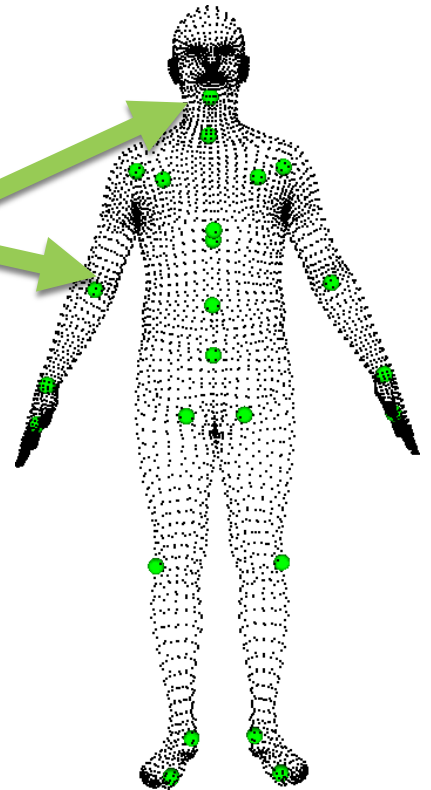
Standard skinning produces vertices from...

– Rest pose vertices: $\mathbf{T} \in \mathbb{R}^{3N}$

– Joint locations: $\mathbf{J} \in \mathbb{R}^{3K}$

– Weights: $\mathcal{W} \in \mathbb{R}^{N \times K}$

– Pose parameters: $\vec{\theta} \in \mathbb{R}^{3K}$



Standard Skinning

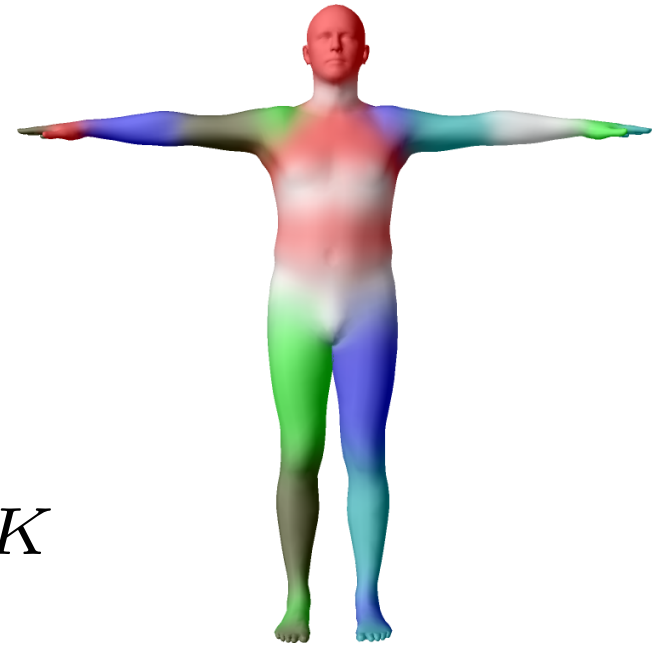
Standard skinning produces vertices from...

– Rest pose vertices: $\mathbf{T} \in \mathbb{R}^{3N}$

– Joint locations: $\mathbf{J} \in \mathbb{R}^{3K}$

– Weights: $\mathcal{W} \in \mathbb{R}^{N \times K}$

– Pose parameters: $\vec{\theta} \in \mathbb{R}^{3K}$



Standard Skinning

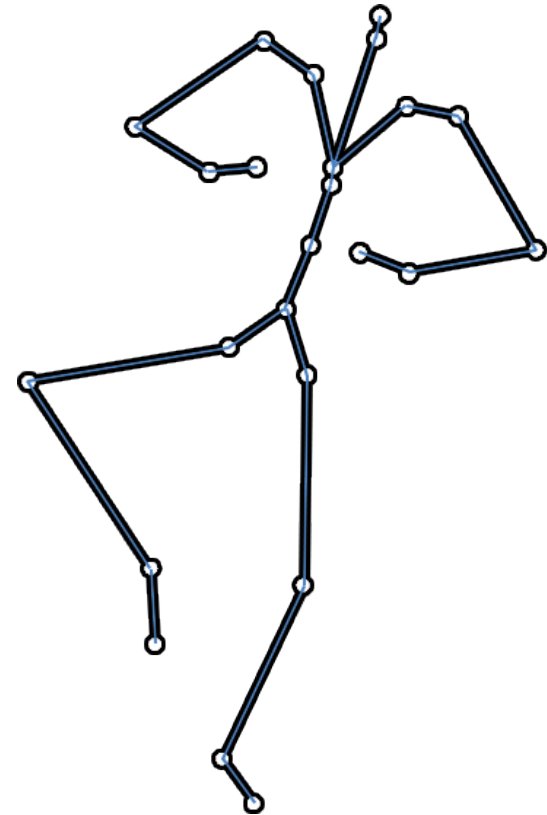
Standard skinning produces vertices from...

– Rest pose vertices: $\mathbf{T} \in \mathbb{R}^{3N}$

– Joint locations: $\mathbf{J} \in \mathbb{R}^{3K}$

– Weights: $\mathcal{W} \in \mathbb{R}^{N \times K}$

– Pose parameters: $\vec{\theta} \in \mathbb{R}^{3K}$

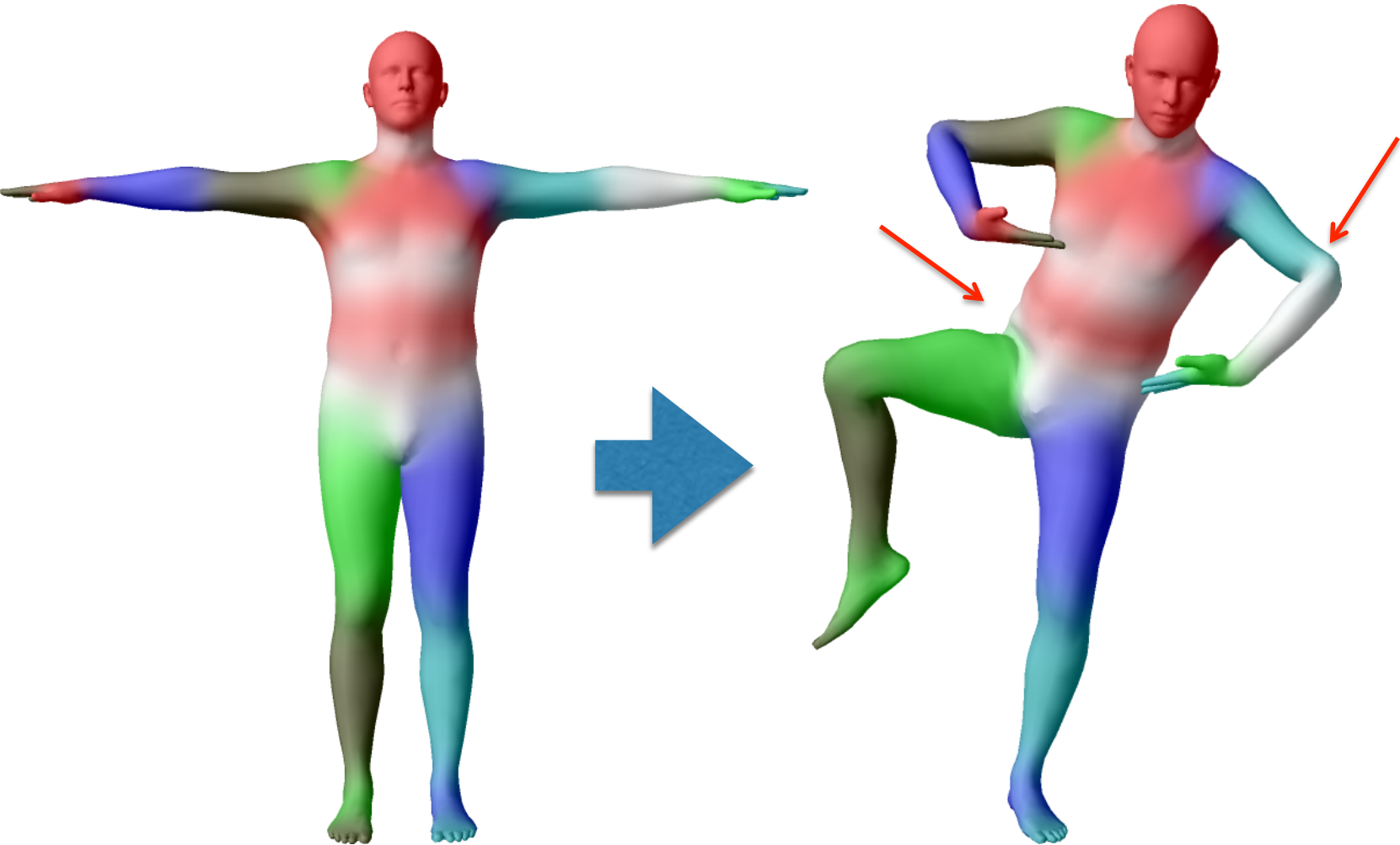


Skinning function

- Rest pose vertices: $\mathbf{T} \in \mathbb{R}^{3N}$
- Joint locations: $\mathbf{J} \in \mathbb{R}^{3K}$
- Weights: $\mathcal{W} \in \mathbb{R}^{N \times K}$
- Pose parameters: $\vec{\theta} \in \mathbb{R}^{3K}$

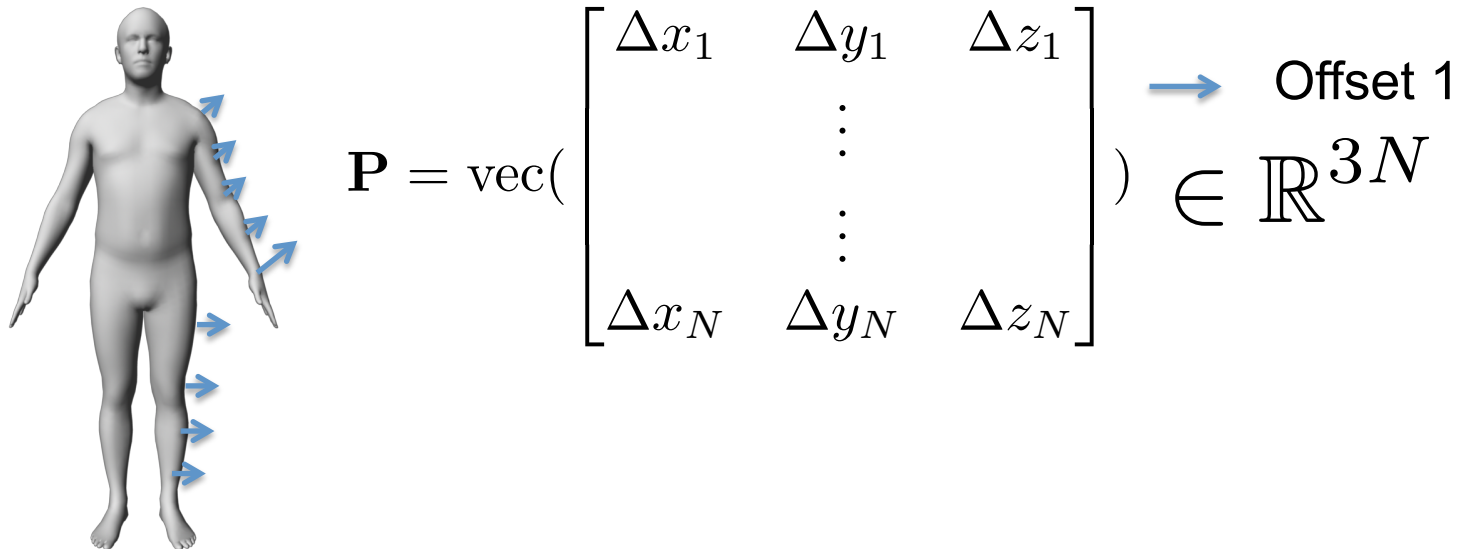
$$W(\mathbf{T}, \mathbf{J}, \mathcal{W}, \vec{\theta}) \mapsto \text{vertices}$$

LBS problems



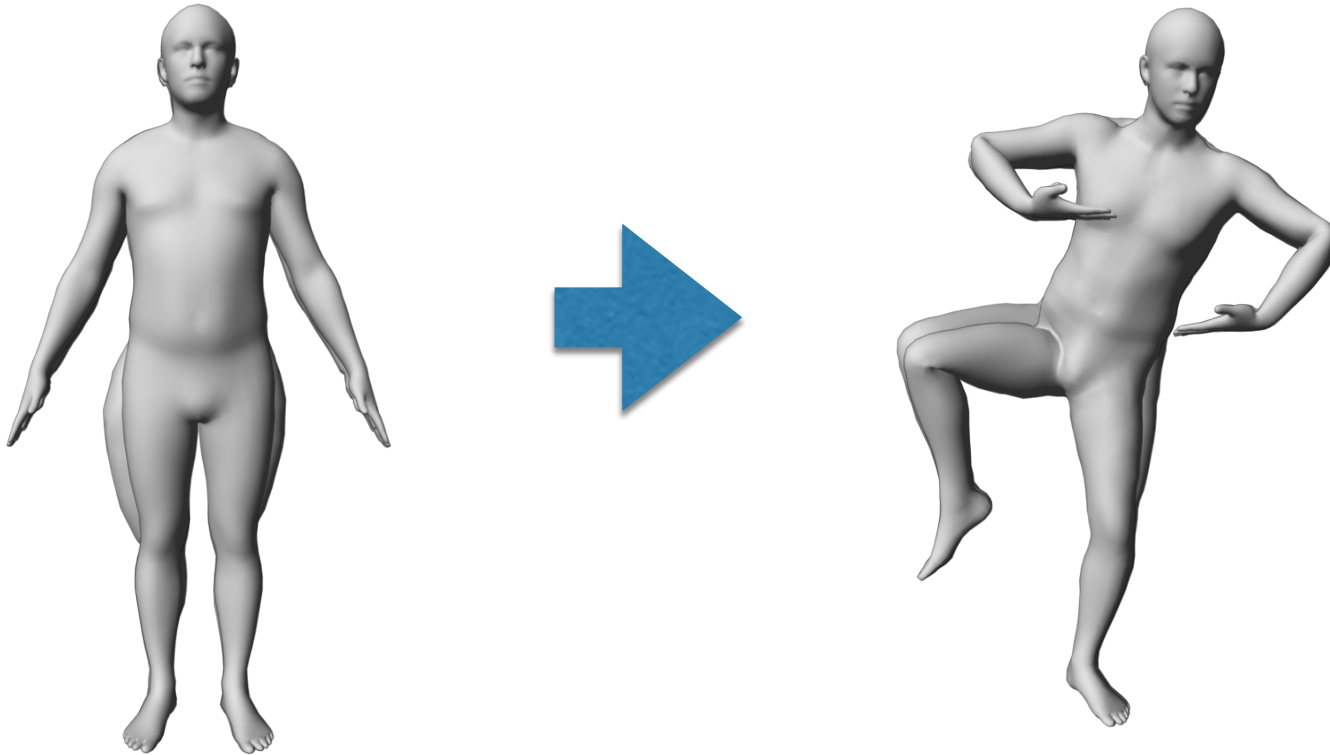
Solution: Blend Shapes

- A **blend shape** is a set of vertex displacements in a rest pose
 - Pose blend shapes: correct for LBS problems



Pose Blend Shapes

- **With** blend shape correction

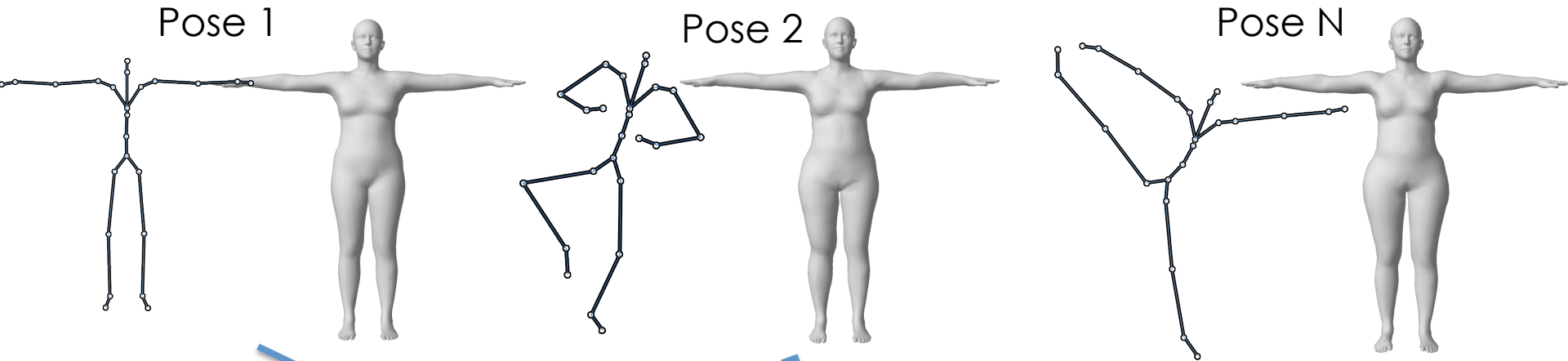


How to predict Blend Shapes ?

- Animators sculpt it manually!
- Time consuming, does not scale

Can we leverage training data ?

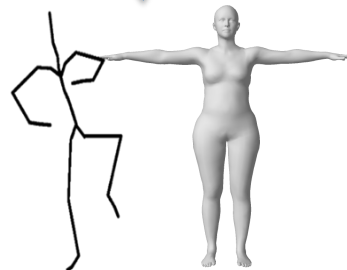
Scattered Data Interpolation



$$\lambda_i \propto K(\vec{\theta}', \vec{\theta}^i) = \exp\left(-\frac{\|\vec{\theta}' - \vec{\theta}^i\|^2}{\tau}\right)$$

$$B_P(\vec{\theta}') = \sum_i \lambda_i(\vec{\theta}') \mathbf{P}_i$$

Query pose



J.P.Lewis et.al. 2000

Problems Scattered Data Interpolation

- 1) Computationally expensive (need to find closest poses in a database)
- 2) Does not extrapolate very well to novel poses

Problems

- If we don't use scattered data interpolation, how do we define pose blend shapes ?

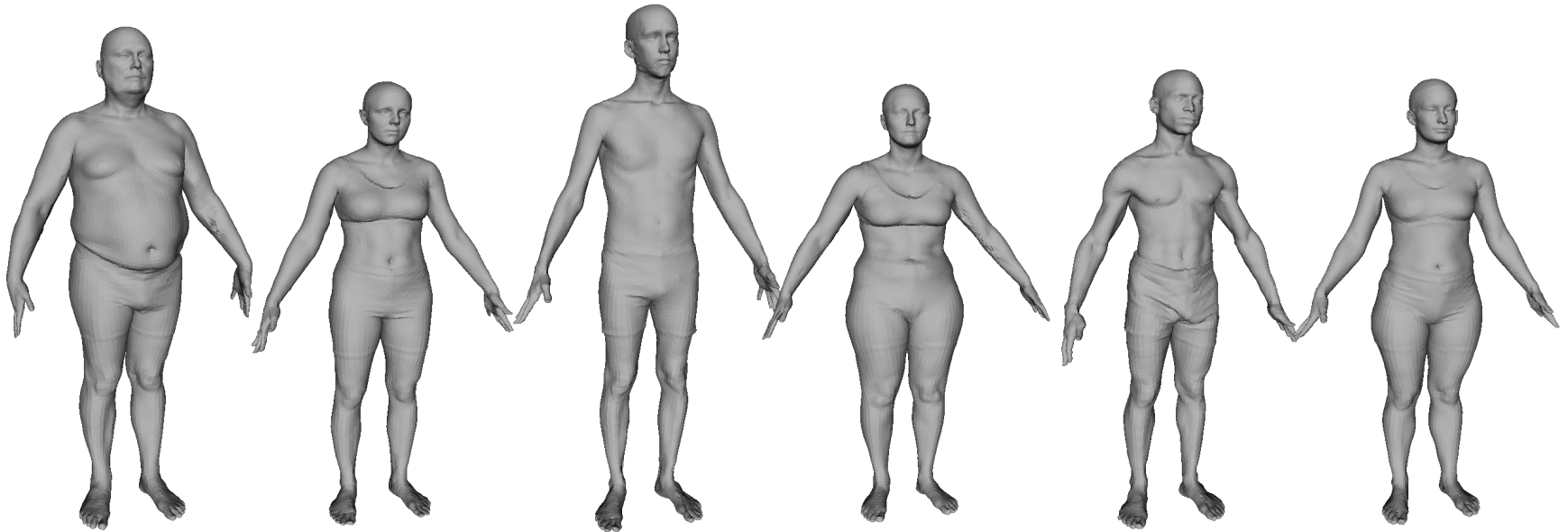
$$B_P(\vec{\theta}')$$

- How to set the skinning parameters ?

$$\mathbf{T} \in \mathbb{R}^{3N} \quad \mathbf{J} \in \mathbb{R}^{3K} \quad \mathcal{W} \in \mathbb{R}^{N \times K}$$

More Problems

How do we model shape identity variations ?



SMPL



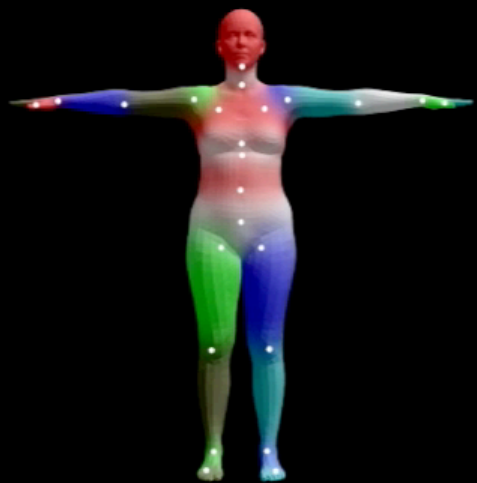
SMPL Model Results

SMPL Philosophy

We aim for the simplest possible model while having state-of-the-art performance

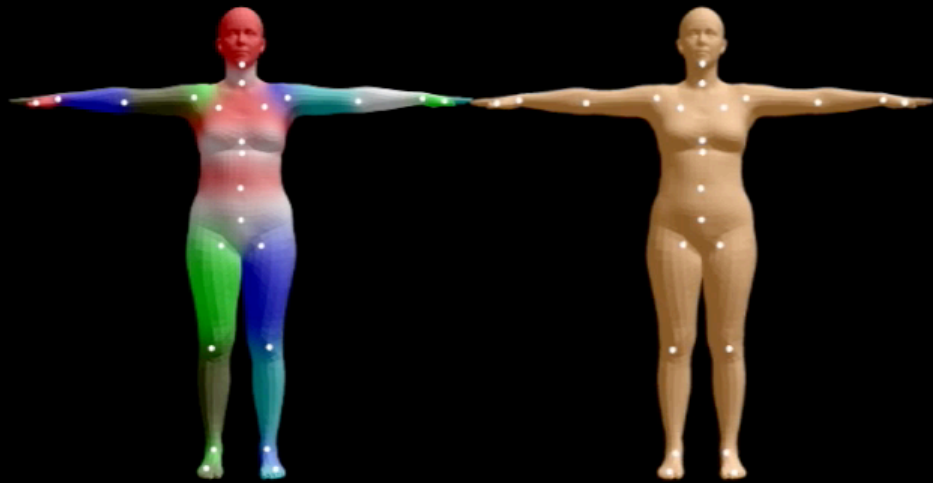
- Makes training easier
- Enables compatibility

Pipeline



Template Mesh

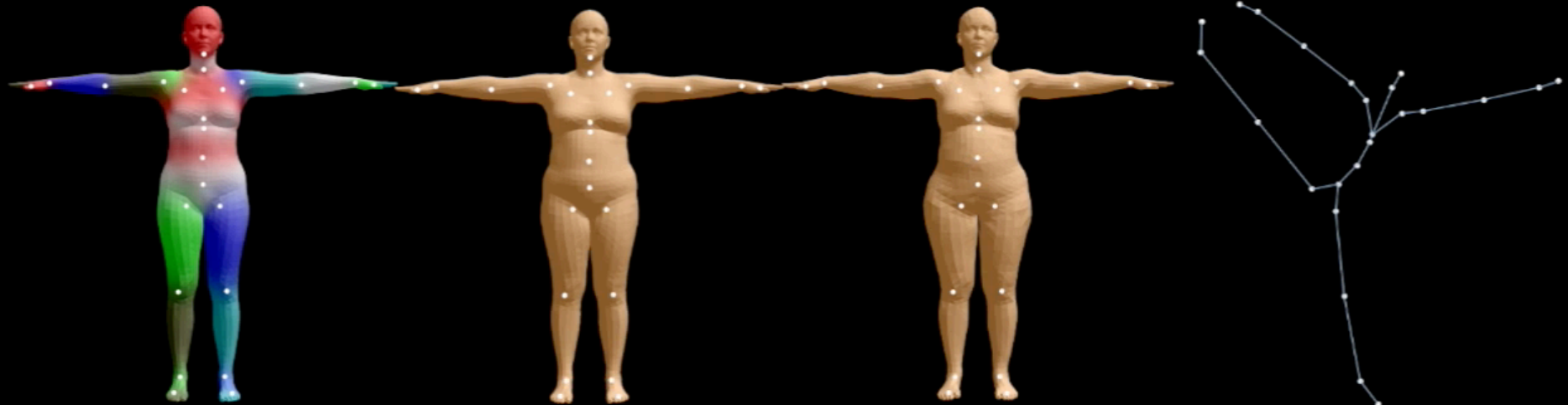
Pipeline



Template Mesh

Shape
Blend Shapes

Pipeline



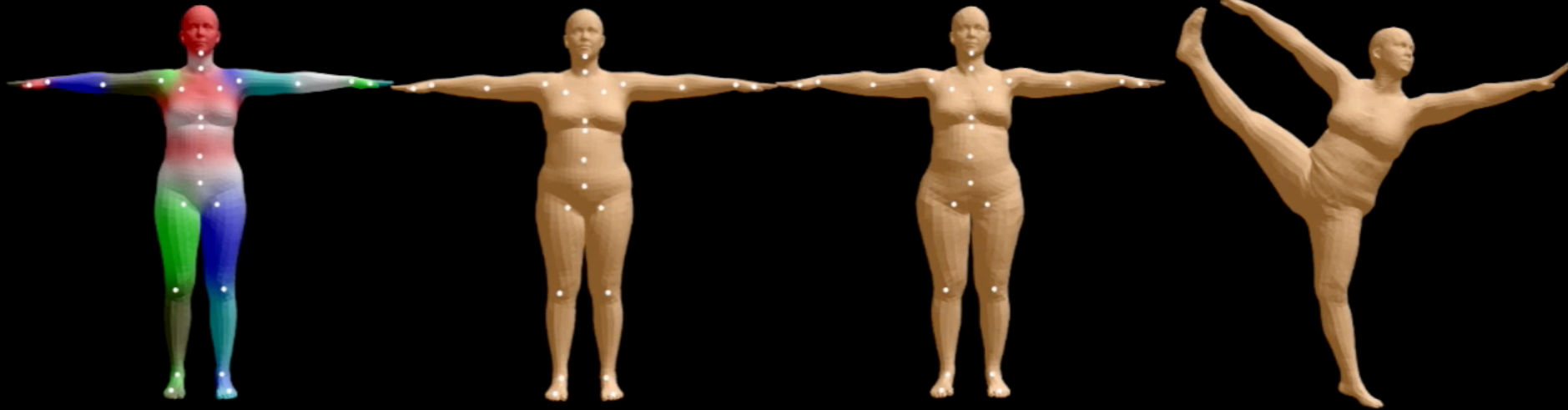
Template Mesh

Shape
Blend Shapes

Pose
Blend Shapes

Given Pose

Pipeline



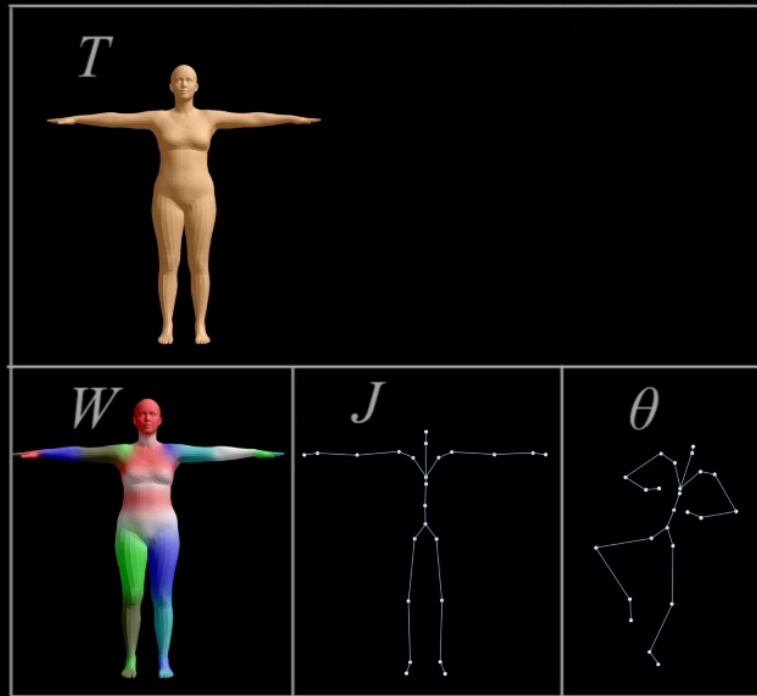
Template Mesh

Shape
Blend Shapes

Pose
Blend Shapes

Final Mesh

Standard Skinning



Parameterized Skinning

Standard skinning $W(\mathbf{T}, \mathbf{J}, \mathcal{W}, \vec{\theta}) \mapsto \text{vertices}$

SMPL model

$M(\vec{\theta}, \vec{\beta}) = W(\mathbf{T}_F(\vec{\beta}, \theta), \mathbf{J}(\vec{\beta}), \mathcal{W}, \vec{\theta}) \mapsto \text{vertices}$

SMPL is skinning parameterized by pose $\vec{\theta}$
and shape $\vec{\beta}$

SMPL: BS are a parametric function of pose

- We parameterize the skinning equation by pose

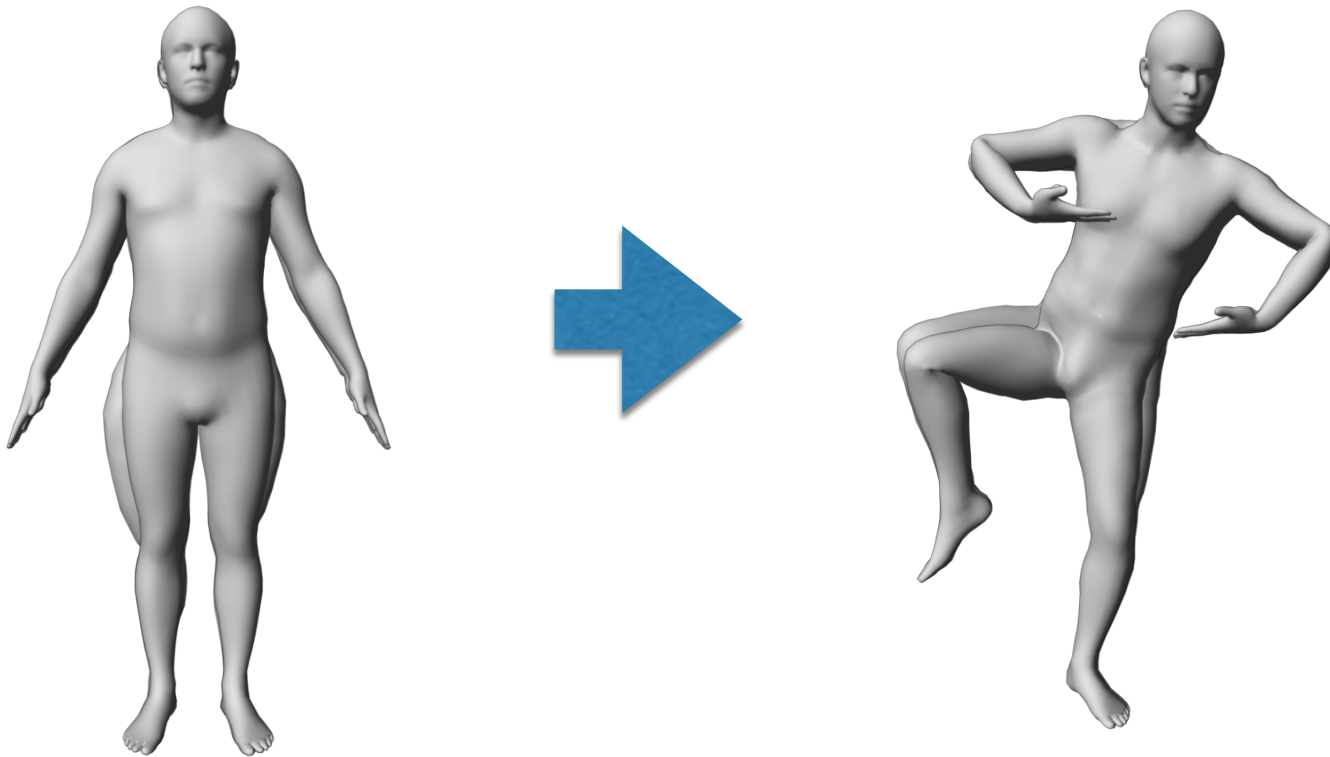
$$W(\mathbf{T}, \mathbf{J}, \mathcal{W}, \vec{\theta})$$



$$W(T(\theta), \mathbf{J}, \mathcal{W}, \vec{\theta})$$

Remember: Pose Blend Shapes

- **With** blend shape correction



Parameterized Skinning

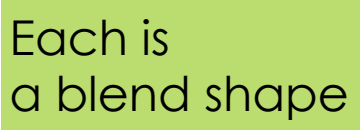
$$W(T(\theta), \mathbf{J}, \mathcal{W}, \vec{\theta}) \mapsto \text{vertices}$$

$$T(\vec{\theta}) = \mathbf{T} + B_P(\vec{\theta})$$

- Our rest vertices are linear in $f(\theta)$

$$B_P(\vec{\theta}) = \sum_i^{|f(\vec{\theta})|} f_i(\vec{\theta}) \mathbf{P}_i$$

Each is
a blend shape



Parameterized Skinning

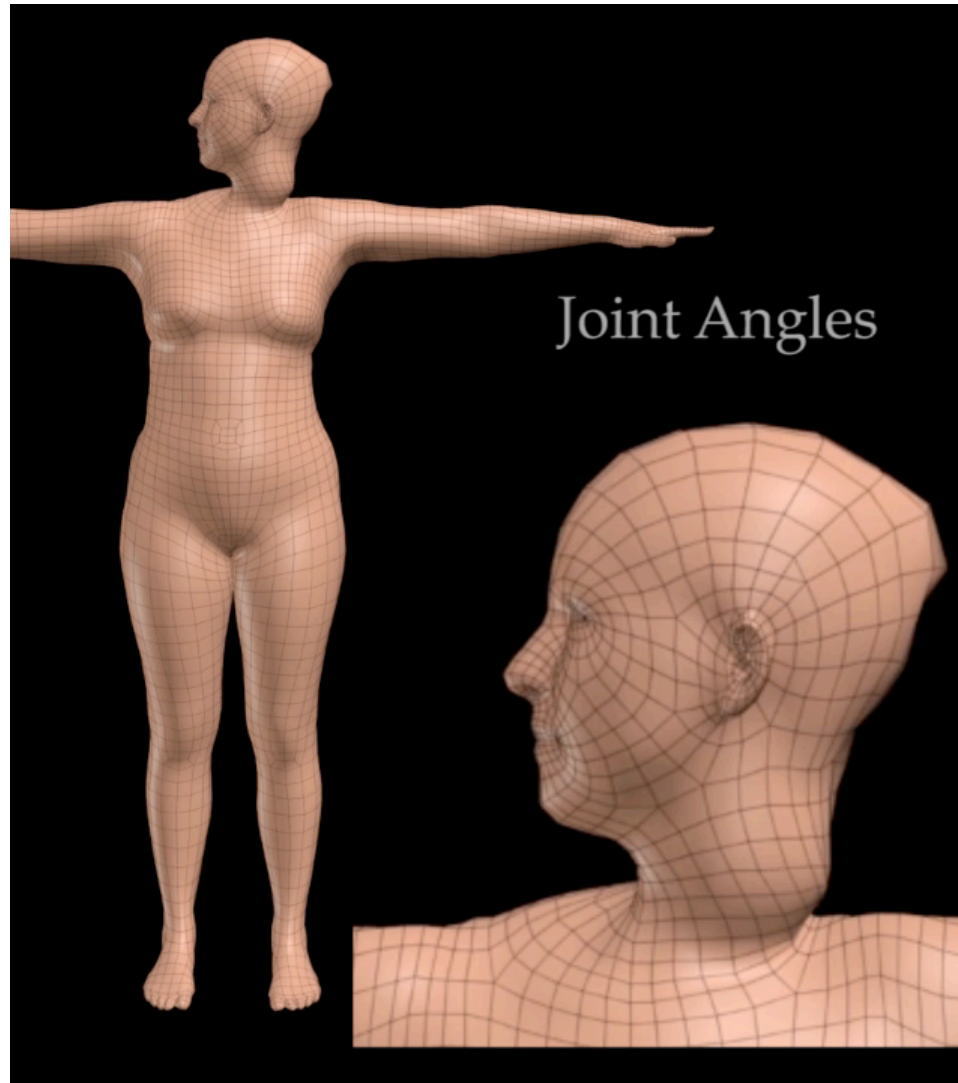
- What function $f(\vec{\theta})$?

$$B_P(\vec{\theta}) = \sum_i \frac{|f(\vec{\theta})|}{|f(\vec{\theta})|} f_i(\vec{\theta}) \mathbf{P}_i$$

- Simplest possible:

$$f(\vec{\theta}) = \vec{\theta}$$

Neck Rotation



Parameterized Skinning

- What function $f(\vec{\theta})$?

$$B_P(\vec{\theta}) = \sum_i \frac{|f(\vec{\theta})|}{|f(\vec{\theta})|} f_i(\vec{\theta}) \mathbf{P}_i$$

- Idea: we consider $f(\vec{\theta})$ as the vectorized joint rotation matrices
- Blend shapes are *linear in rotation matrix elements*

Pose Blend Shapes

$$B_P(\vec{\theta}) = \sum_i^{|f(\vec{\theta})|} f_i(\vec{\theta}) \mathbf{P}_i$$

$$\vec{\theta} = (\vec{\omega}_1, \dots, \vec{\omega}_k)^T$$

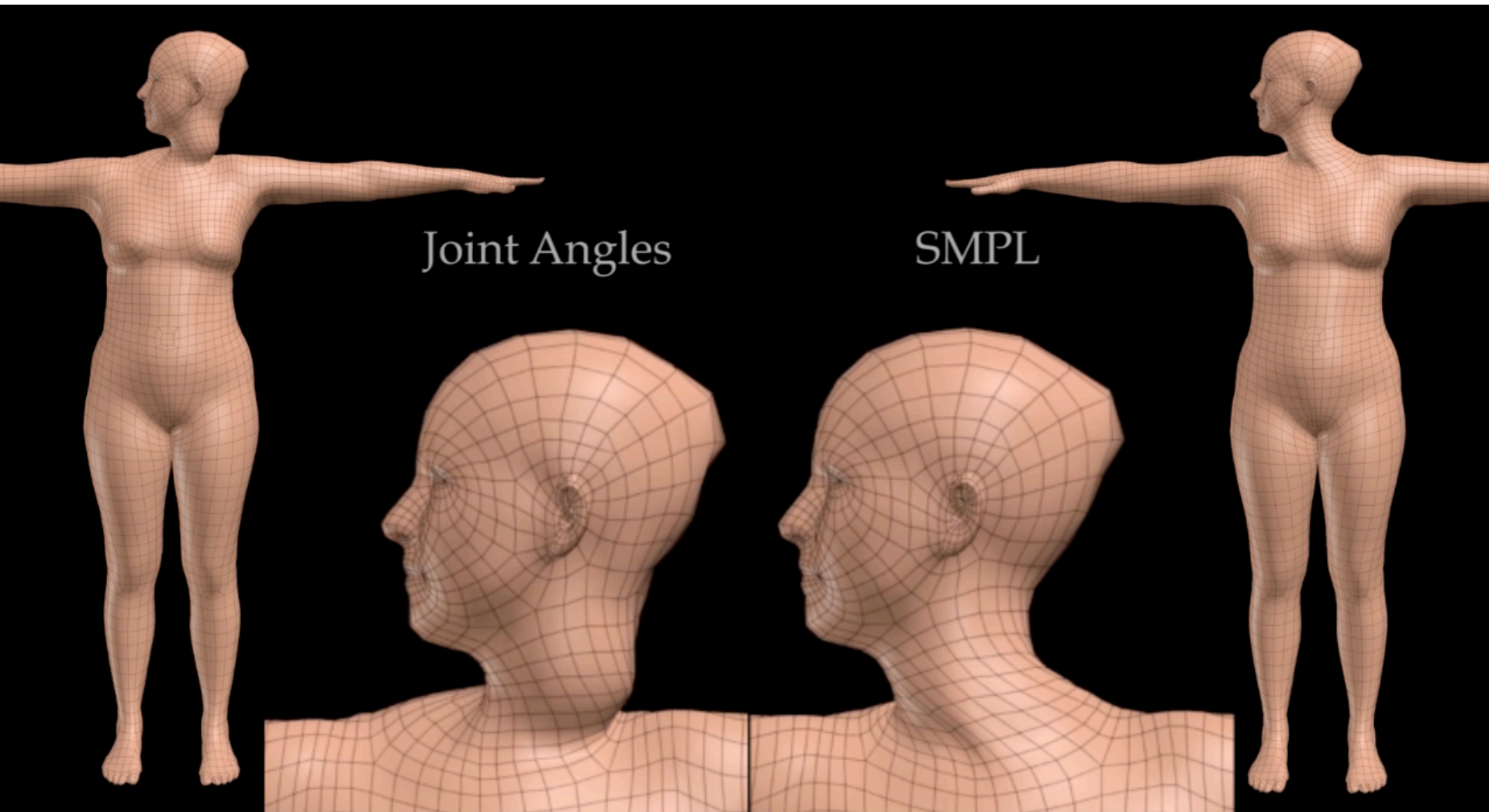
$$e^{\hat{\omega}_1} - \mathcal{I} \quad e^{\hat{\omega}_K} - \mathcal{I}$$

Not a minus

$$f(\vec{\theta}) = \left[\underbrace{\bar{e}_{1,1}^{\hat{\omega}_1} \dots \bar{e}_{3,3}^{\hat{\omega}_1}} \quad \dots \quad \underbrace{\bar{e}_{1,1}^{\hat{\omega}_K} \dots \bar{e}_{3,3}^{\hat{\omega}_K}} \right]$$

9 elements of the rotation matrix -> We learn $9 \times K = 207$ blendshapes

Neck Rotation



Pose Blendshapes demo

- `>> python visualize_pose_blends.py`

Joint Location Estimation

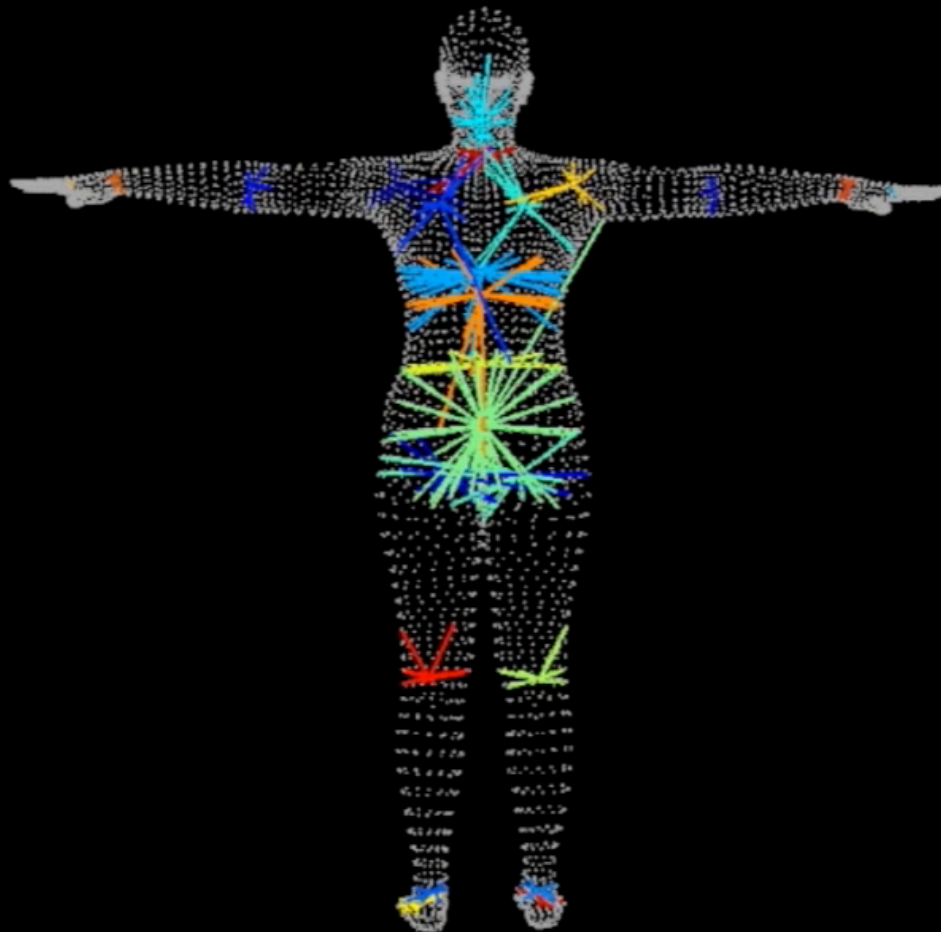
- How to get the joints \mathbf{J} for a new shape?
What is the simplest way?
- Joints are considered linear in rest vertices
(much like in Allen et al. '06)

$$\mathbf{J} = J(\mathbf{T}; \mathcal{J}) = \mathcal{J}\mathbf{T}$$



Joint regressor matrix

Joint Location Estimation



Adding a shape space

Problem: want a shape space with different identities

$$W(T(\vec{\theta}), J(\mathbf{T}), \mathcal{W}, \vec{\theta}) \mapsto \text{vertices}$$

$$T(\vec{\theta}) = \mathbf{T} + B_P(\vec{\theta})$$

$$\text{Pose contribution} \left\{ B_P(\vec{\theta}) = \sum_i^{|f(\vec{\theta})|} f_i(\vec{\theta}) \mathbf{P}_i \right.$$

Adding a shape space

Solution: add blend shapes linear with $\vec{\beta}$

$W(T(\vec{\theta}, \vec{\beta}), J(\vec{\beta}), \mathcal{W}, \vec{\theta}) \mapsto$ vertices

$$T_P(\vec{\theta}, \vec{\beta}) = \mathbf{T} + B_P(\vec{\theta}) + B_S(\vec{\beta})$$

Pose contribution

$$\left\{ B_P(\vec{\theta}) = \sum_i^{|f(\vec{\theta})|} f_i(\vec{\theta}) \mathbf{P}_i \right.$$

Shape contribution

$$\left\{ B_S(\beta) = \sum_j^{|\beta|} \beta_j S_j \right. \rightarrow \mathcal{S} = [\mathbf{S}_1 \quad \mathbf{S}_2 \quad \dots \quad \mathbf{S}_{N_{\text{subj}}}]$$

Shape Blend shape matrix

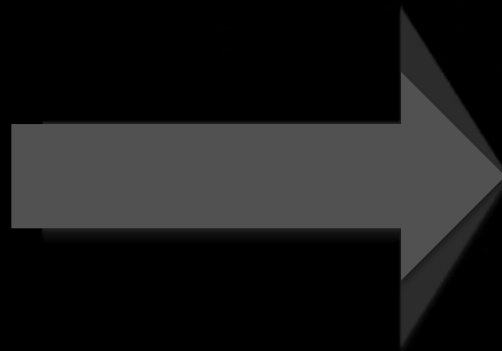
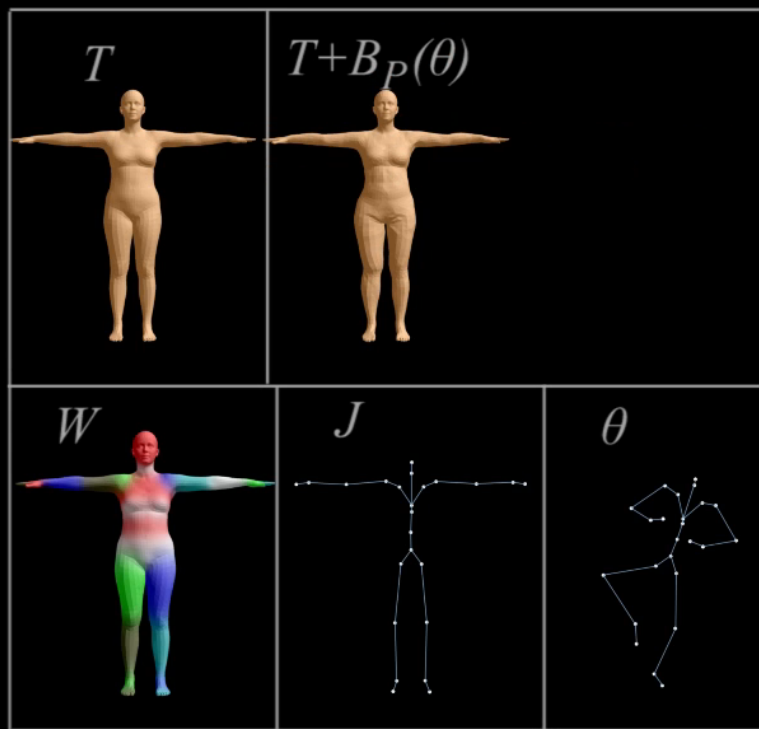
SMPL

Additive Model

$$\bar{\mathbf{t}}'_i = \sum_{k=1}^K w_{k,i} G'_k(\vec{\theta}, J(\vec{\beta})) (\bar{\mathbf{t}}_i + \mathbf{b}_{S,i}(\vec{\beta}) + \mathbf{b}_{P,i}(\vec{\theta}))$$

Blendweights Vertices Shape-bs Pose-bs

SMPL Skinning



Parameterized Skinning

Standard skinning $W(\mathbf{T}, \mathbf{J}, \mathcal{W}, \vec{\theta}) \mapsto$ vertices

SMPL model

$M(\vec{\theta}, \vec{\beta}) = W(\mathbf{T}_F(\vec{\beta}, \theta), \mathbf{J}(\vec{\beta}), \mathcal{W}, \vec{\theta}) \mapsto$ vertices

SMPL is skinning parameterized by pose $\vec{\theta}$
and shape $\vec{\beta}$

SMPL

$$M(\underbrace{\vec{\theta}, \vec{\beta}}_{\text{Input}}; \underbrace{\mathbf{T}, \mathcal{S}, \mathcal{P}, \mathcal{W}, \mathcal{J}}_{\text{Model parameters to be learned from data}})$$

pose shape

- \mathbf{T} Template (average shape)
- \mathcal{S} Shape blend shape matrix
- \mathcal{P} Pose blend shape matrix
- \mathcal{W} Blendweights matrix
- \mathcal{J} Joint regressor matrix

Remember ?

$$M(\underbrace{\vec{\theta}, \vec{\beta}}_{\text{Input parameters}}; \underbrace{\mathbf{T}, \mathcal{S}, \mathcal{P}, \mathcal{W}, \mathcal{J}}_{\text{Hyper-parameters}})$$

$$f(x; \mathbf{w})$$

Input parameters

Hyper-parameters ?

DATA

Model Training

Multipose database: 20 males, 24 females
1800 registrations



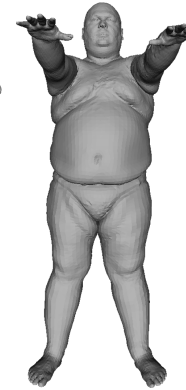
Model Training

Multishape database: PCA on ~2000 single-pose registrations per gender



Model Training

$$\mathbf{w} = \arg \min_{\mathbf{w}} \sum_j \|M(\vec{\theta}, \vec{\beta}; \mathbf{w}) - \text{Target}_j\|^2$$



Training Details

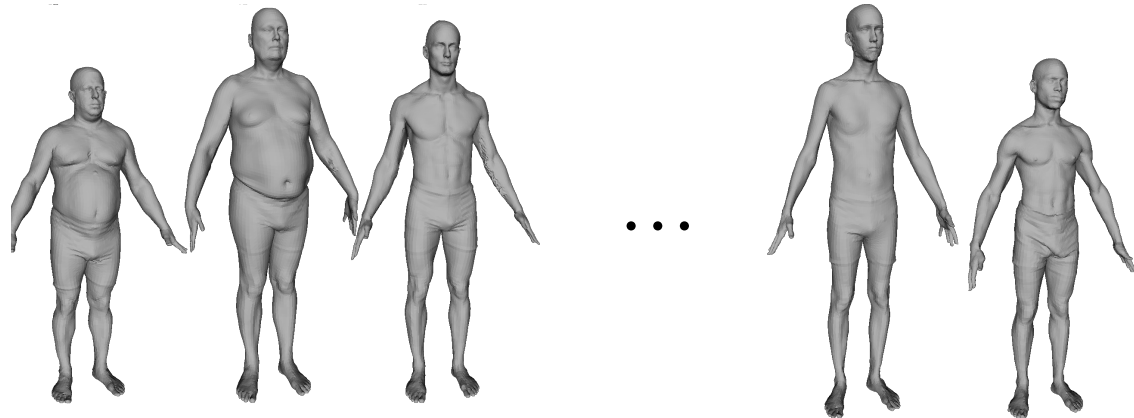
- $\mathcal{P}, \mathcal{W}, \mathcal{J}$ are trained from our **multipose** dataset
- \mathcal{P} regularized towards zero (ridge regression)
- \mathcal{W} regularized towards initialization
- \mathcal{J} regularized towards predicting part boundary centers and is forced to be sparse
- \mathbf{T}, \mathcal{S} are trained from our **multishape** dataset

Number of Parameters Learned

For a model with 6890 vertices

- \mathcal{P} $9 \times 23 \times 6890 = 4,278,690$
- \mathcal{W} $4 \times 3 \times 6890 = 82,680$
- \mathcal{J} $3 \times 6890 \times 23 \times 3 = 1,426,230$
- \mathcal{T}, \mathcal{S} $3 \times 6890 + 3 \times 6890 \times 10 \text{blendshapes} = 227,370$

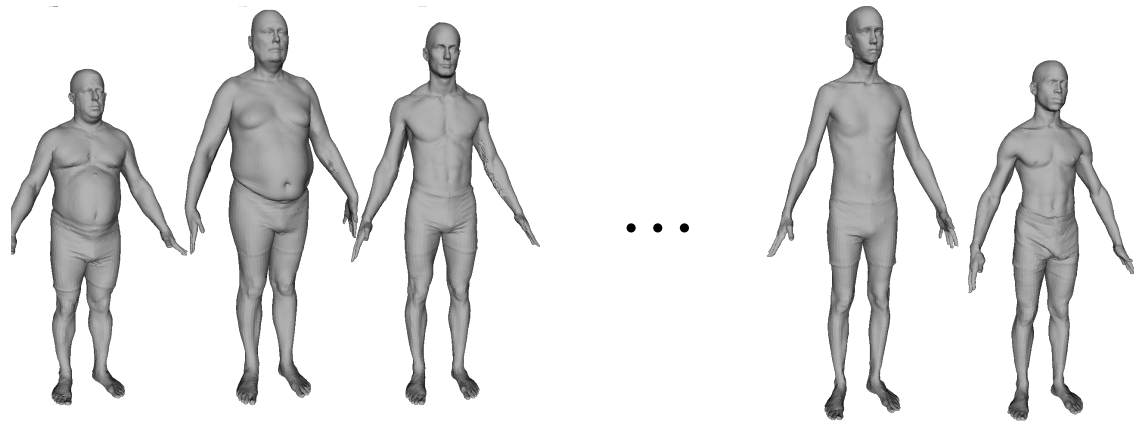
A total of 6.014.970 parameters are learned



$$\begin{bmatrix} \mathbf{V}_1 & \mathbf{V}_2 & \dots & \mathbf{V}_{N_{\text{subj}}} \end{bmatrix} = \mathbf{T} + \begin{bmatrix} \mathbf{S}_1 & \mathbf{S}_2 & \dots & \mathbf{S}_{N_{\text{subj}}} \end{bmatrix} \mathcal{B}$$

Average of shapes

Shape blend shapes are the first eigenvectors



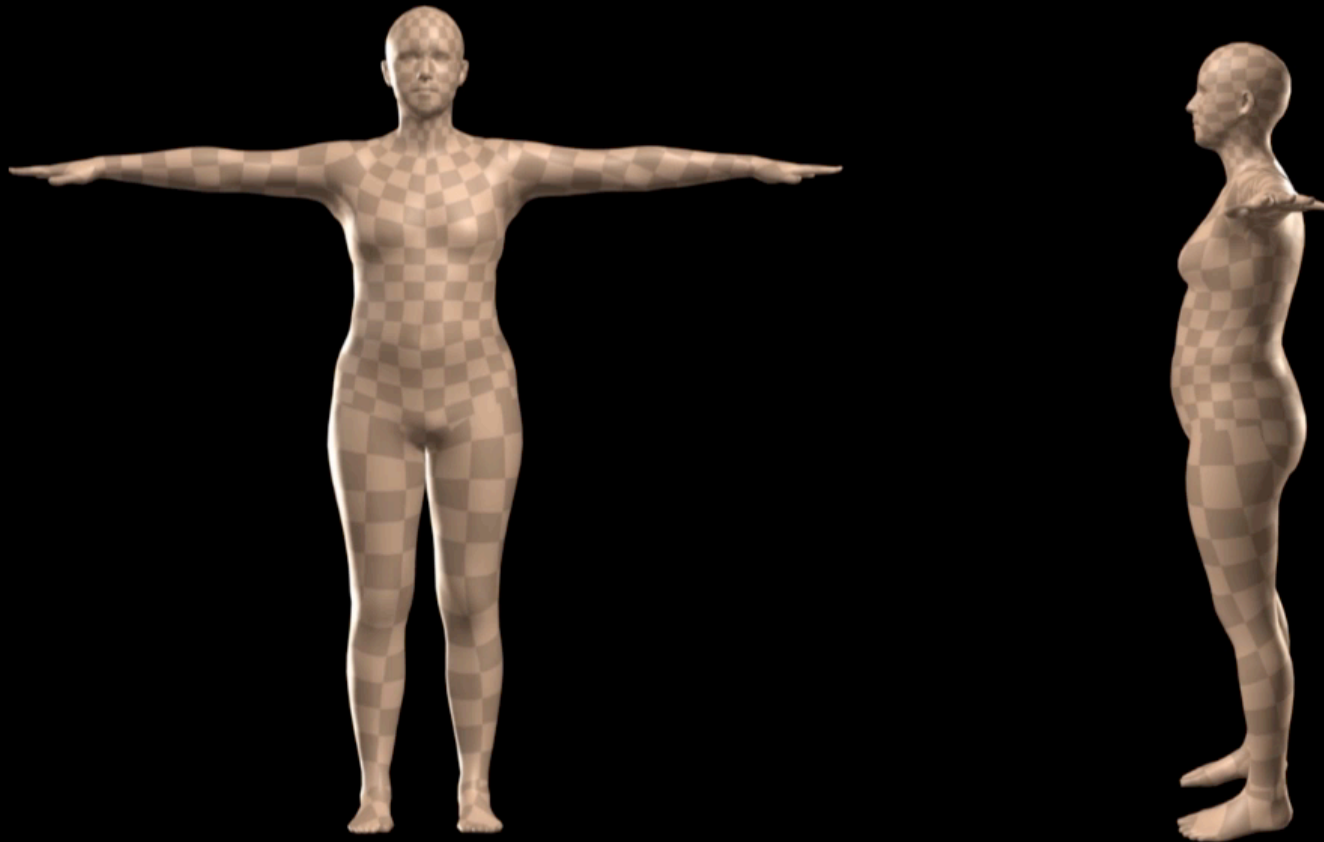
$$\left[\mathbf{V}_1 \quad \mathbf{V}_2 \quad \dots \quad \mathbf{V}_{N_{\text{subj}}} \right] \approx \mathbf{T} + \mathcal{S}\mathcal{B}$$

Average of shapes

Shape blend shapes matrix

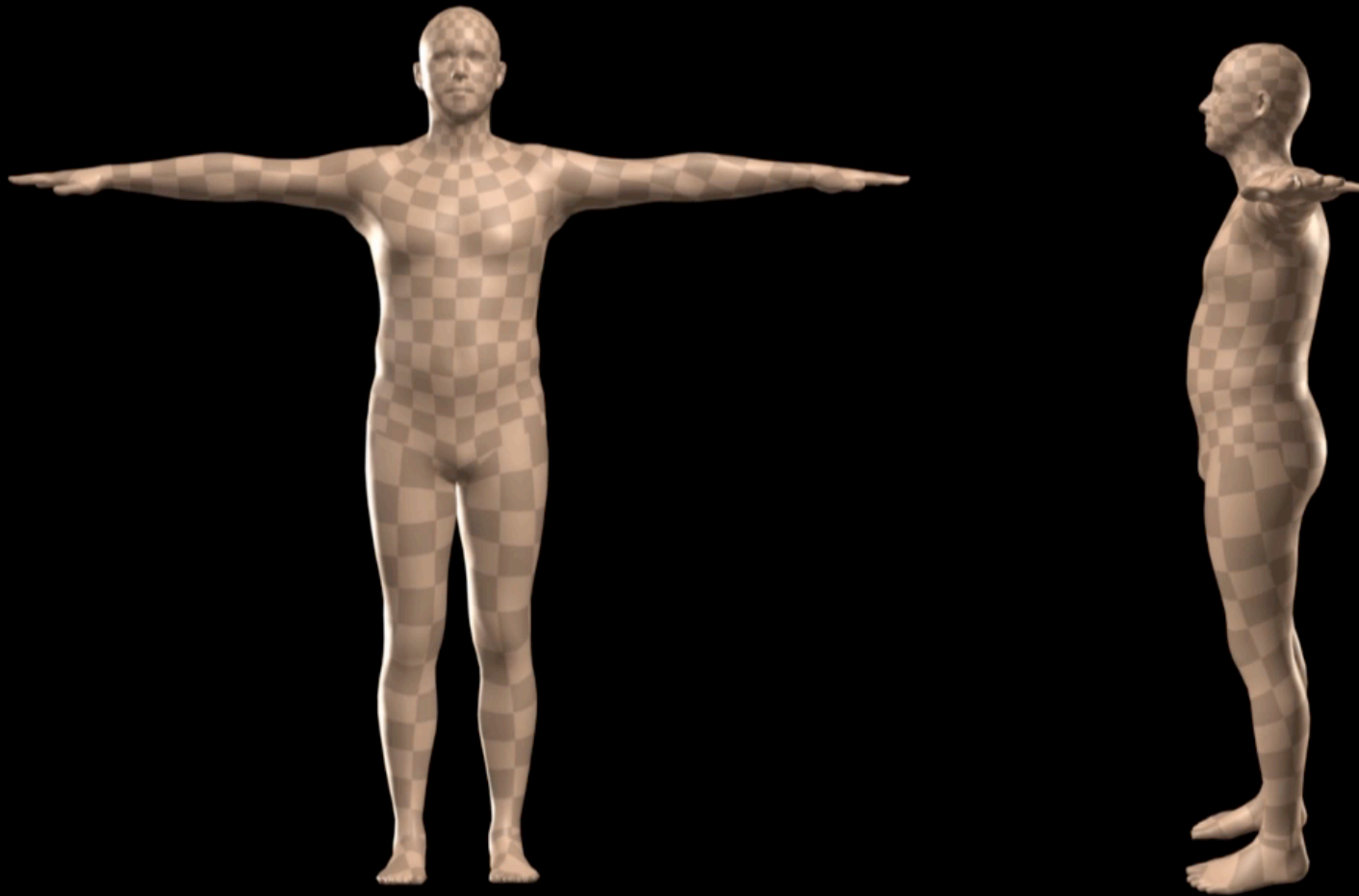
Before doing PCA all shapes have to be in the same pose (pose needs to be optimized)

Shape Blend Shapes- Female



PC 1 varied between ± 3 std dev

Shape Blend Shapes- Male



PC 1 varied between ± 3 std dev

Pose Blendshapes

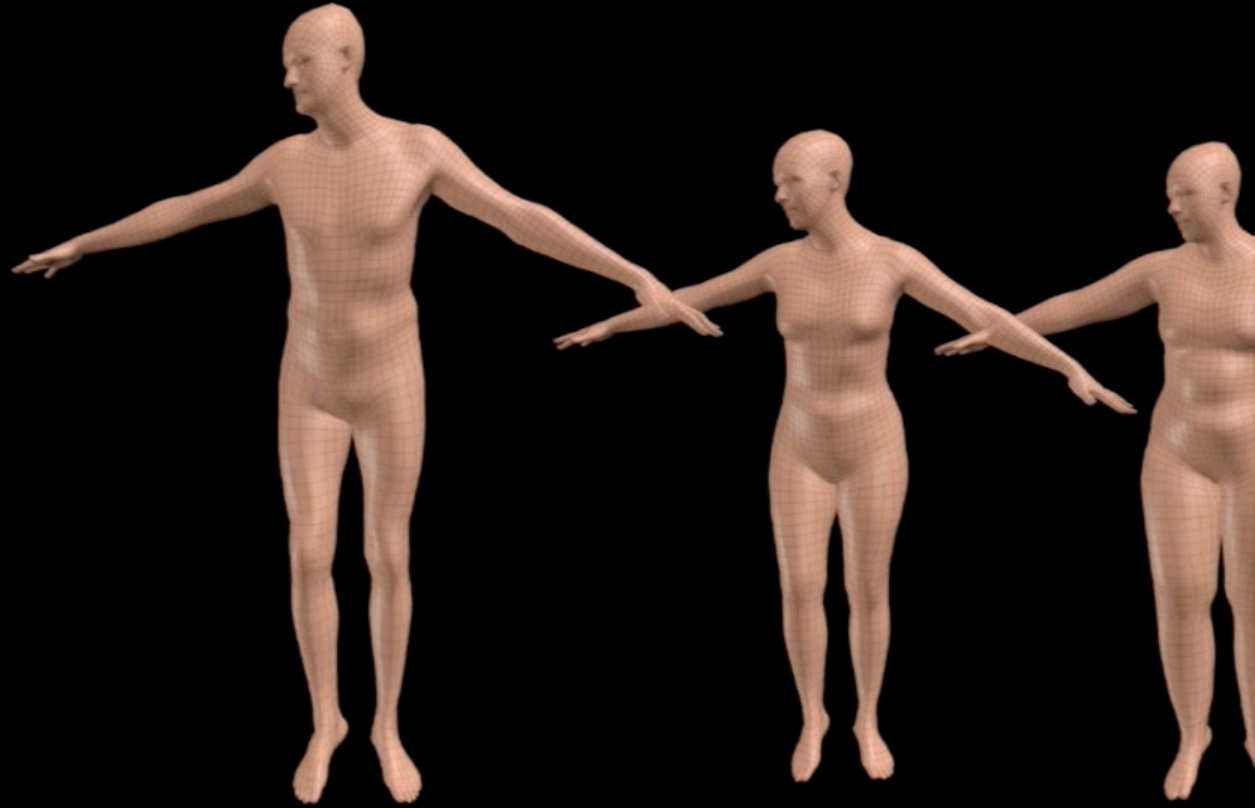


Conclusion

- **Speed:** fast run-time
- **Fidelity:** superior accuracy to Blend-SCAPE, trained on the same data
- **Compatibility:** works in Maya, other platforms soon
- Is publicly available for research purposes

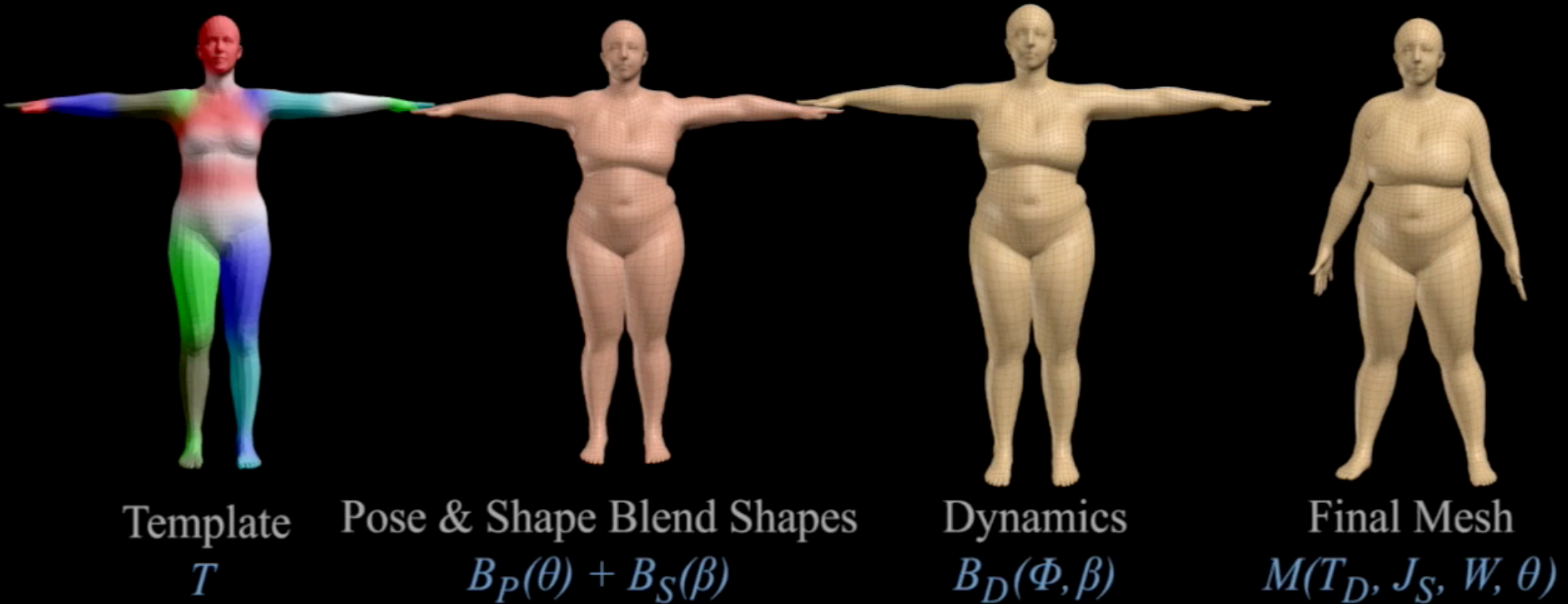
Download: <http://smpl.is.tue.mpg.de>

SMPL results

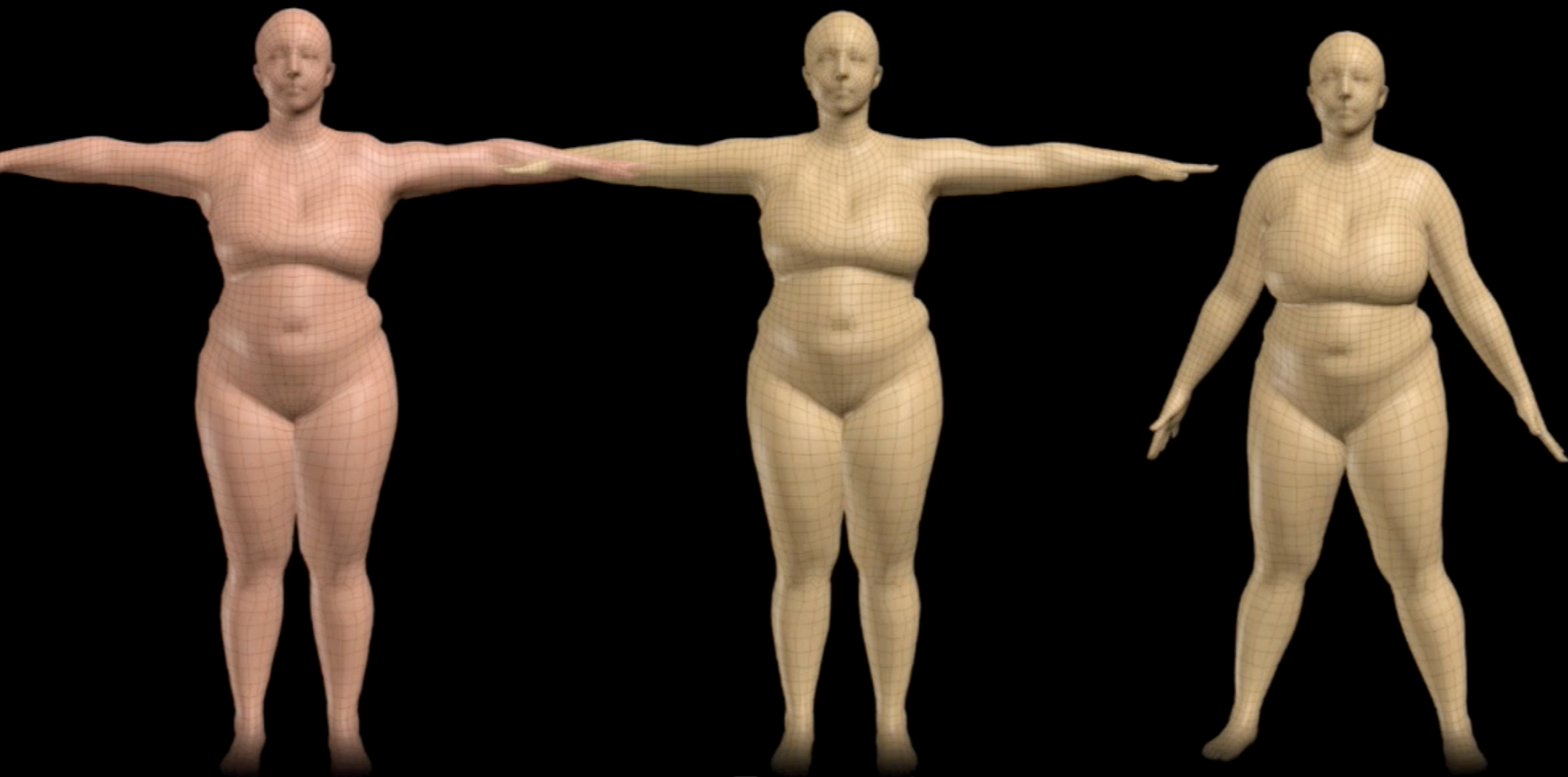


SMPL Model

Model Decomposition



Dynamics of Soft Tissue

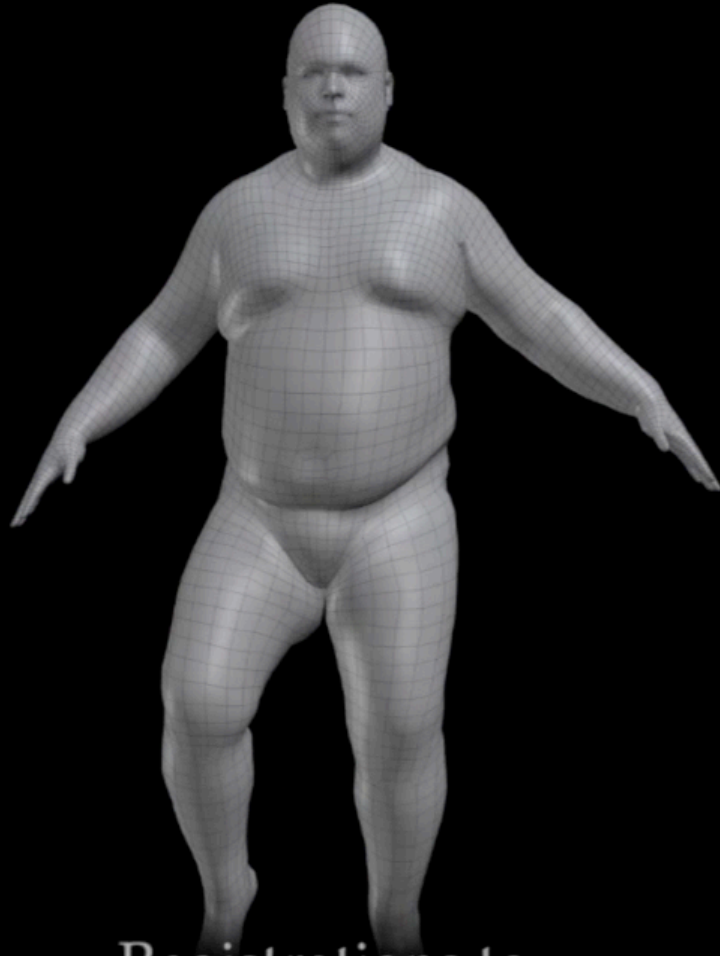


Pose
Blend Shapes

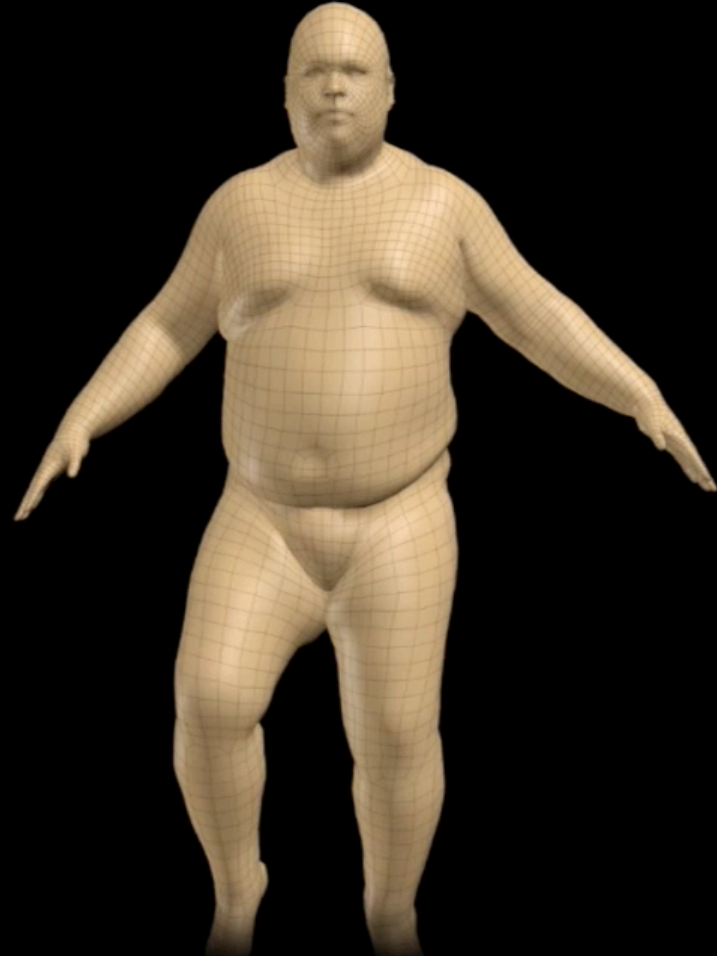
Dynamic
Blend Shapes

DMPL

DMPL exaggeration



Registrations to
4D Scans



DMPL

Applications 1

- Given a new registration, find the pose and shape. Correspondences are known.
- >> align_3Dpoints.py

Fitting SMPL to a scan/mesh

- Problem: Given a registration, find the model pose and shape.

$$\vec{\theta}, \vec{\beta} = \arg \min_{\vec{\theta}, \vec{\beta}} \|M(\vec{\theta}, \vec{\beta}) - \mathbf{V}\|^2$$

Model

Registration

Chumpy does it for you but you
have to know what you are doing!!

- Chumpy minimizes the **sum of squares** of a **vector valued error** function

Optimization variables (vector)

$$e(\mathbf{x}) = \sum_i \mathbf{e}_i(\mathbf{x})^2 = \mathbf{e}(\mathbf{x})^T \mathbf{e}(\mathbf{x})$$

Sum of squares
(scalar)

Residuals
(vector valued error function)

Jacobian of the vector valued error function:

$$J_{\mathbf{e}}(\mathbf{x}) = \frac{d\mathbf{e}(\mathbf{x})}{d\mathbf{x}} = \underbrace{\begin{bmatrix} \frac{\partial \mathbf{e}_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{e}_1}{\partial \mathbf{x}_P} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{e}_N}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{e}_N}{\partial \mathbf{x}_P} \end{bmatrix}}_{\text{P parameters}} \left. \vphantom{\begin{bmatrix} \frac{\partial \mathbf{e}_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{e}_1}{\partial \mathbf{x}_P} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{e}_N}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{e}_N}{\partial \mathbf{x}_P} \end{bmatrix}} \right\} \text{N residuals}$$

Gradient

$$\mathbf{g}(\mathbf{x}) = \frac{de}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial e}{\partial \mathbf{x}_1} \\ \vdots \\ \vdots \\ \frac{\partial e}{\partial \mathbf{x}_P} \end{bmatrix} = \mathbf{J}_e^T(\mathbf{x})\mathbf{e}(\mathbf{x})$$

Gradient of sum
of squares

Jacobian of
vector valued
error function

Who cares about the Jacobian ?

- Gradient is just a direction not a step.
- To compute the step most optimizers need to approximate the Hessian which requires the Jacobian.
- Many optimizers exploit the structure of the Jacobian.
- Direct application of chain rule makes you compute Jacobians

If optimization takes too long, or breaks etc..

Ask yourself the following:

- Is my Jacobian too big ?
- Is it too dense ? (sparsity is exploited for speed).
- Is my Jacobian full rank ? If Jacobian loses rank optimization can break. This is a typical case is when the error function does not depend on a particular variable x_i .

How do we use the model to solve computer vision problems ?

- Model the 3D world first, then explain image observations
- In the next lecture we will cover modeling appearance and fitting models to images