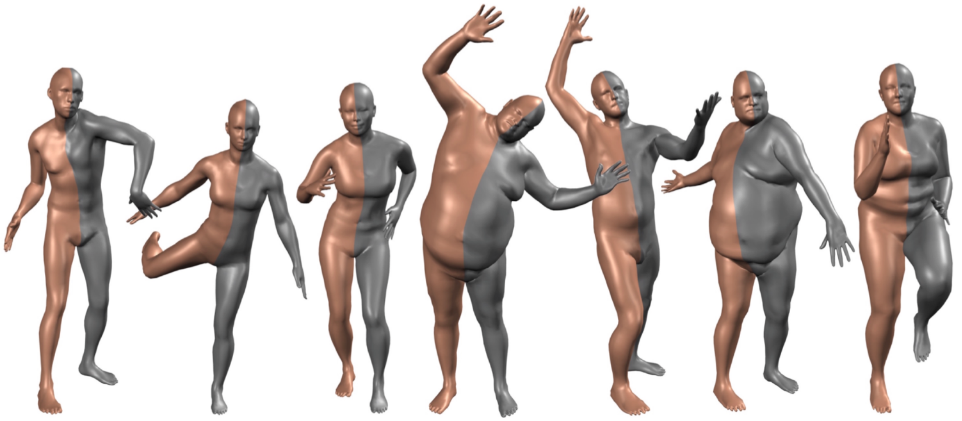
Body Models II



Gerard Pons-Moll and Paul Swoboda

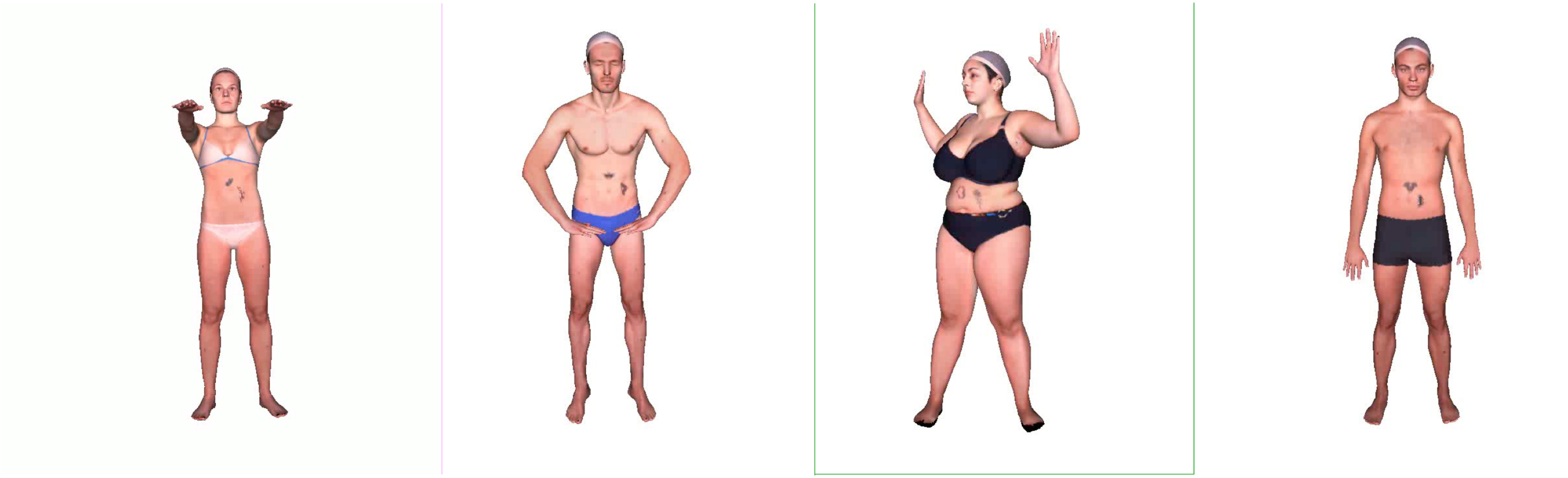
Max Planck Institut für Informatik

December 19, 2018

Schedule

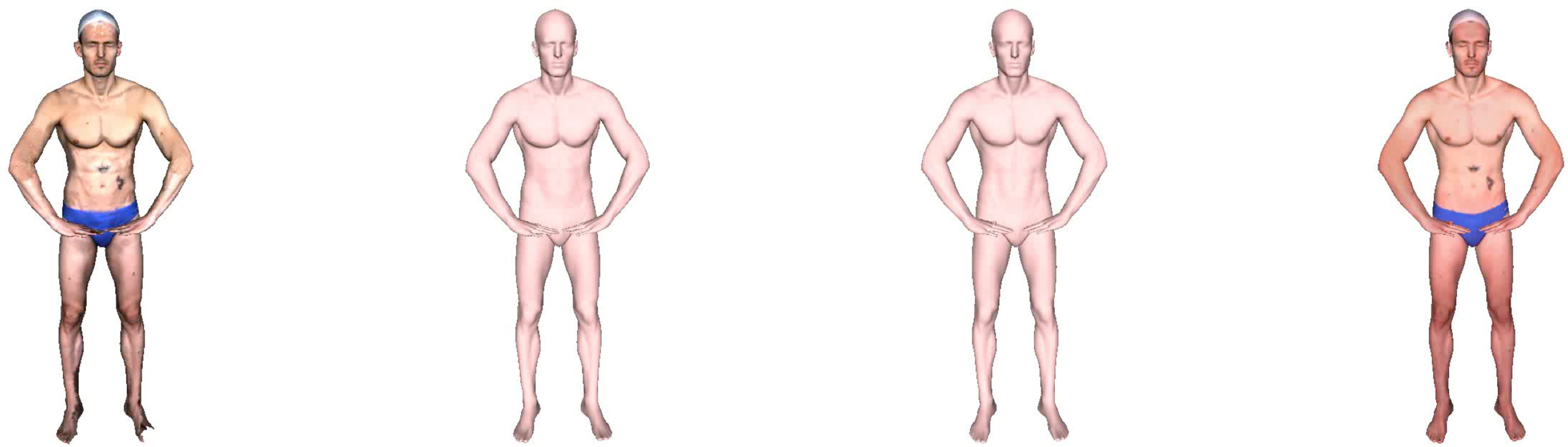
17.10.2018	An Optimization Perspective
24.10.2018	Introduction to probabilities and directed/undirected graphs
31.10.2018	An Optimization Perspective
07.11.2018	An Optimization Perspective
14.11.2018	An Optimization Perspective
21.11.2018	An Optimization Perspective
12.12.2018	Body Models 1
19.12.2018	**Body Models 2**
09.01.2019	Body Models 3
~~16.01.2019~~ 11.01.2019	Sampling and Tracking
23.01.2019	Graphical Models in Computer Vision
06.02.2019	Wrap-up

Our research goal: Virtual humans



What is a virtual human model?

3D scan with texture | **Ground truth shape** | **Model** | **Model** with texture



$$M(\vec{\theta}, \vec{\beta}, \vec{u}; \Phi) \longrightarrow \text{3D mesh}$$

pose

shape

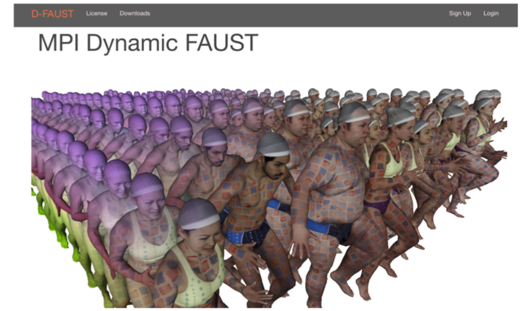
texture

Hyper-parameters to learn

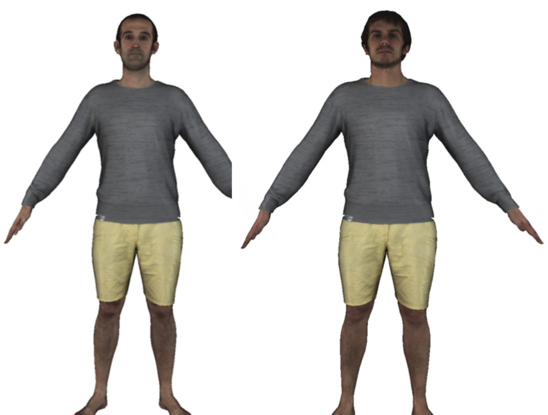
Applications

Tracking from depth

Virtual Reality

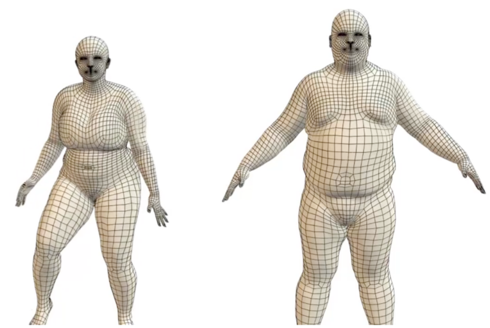


Registration



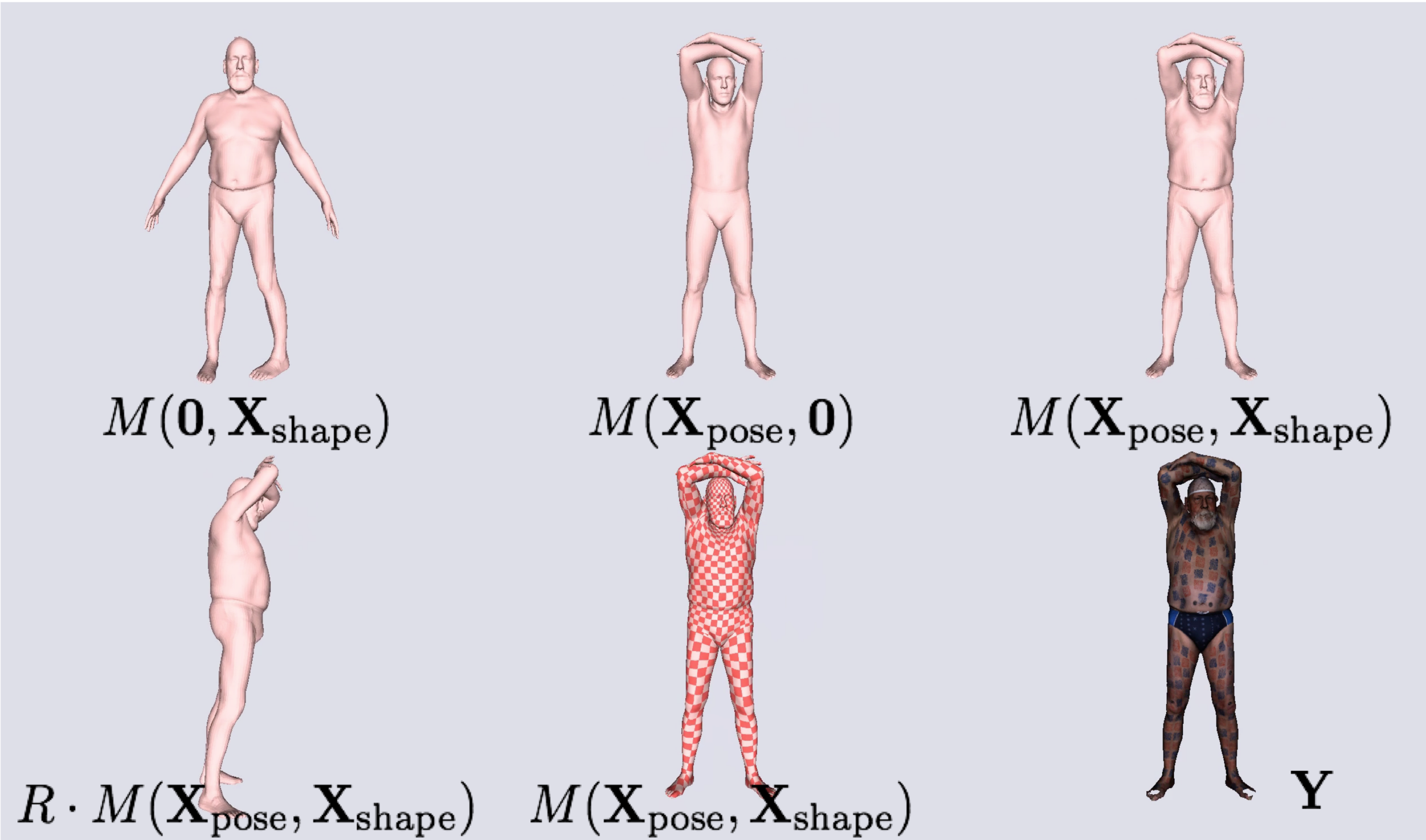
Cloth modeling and try-on

Tracking from images



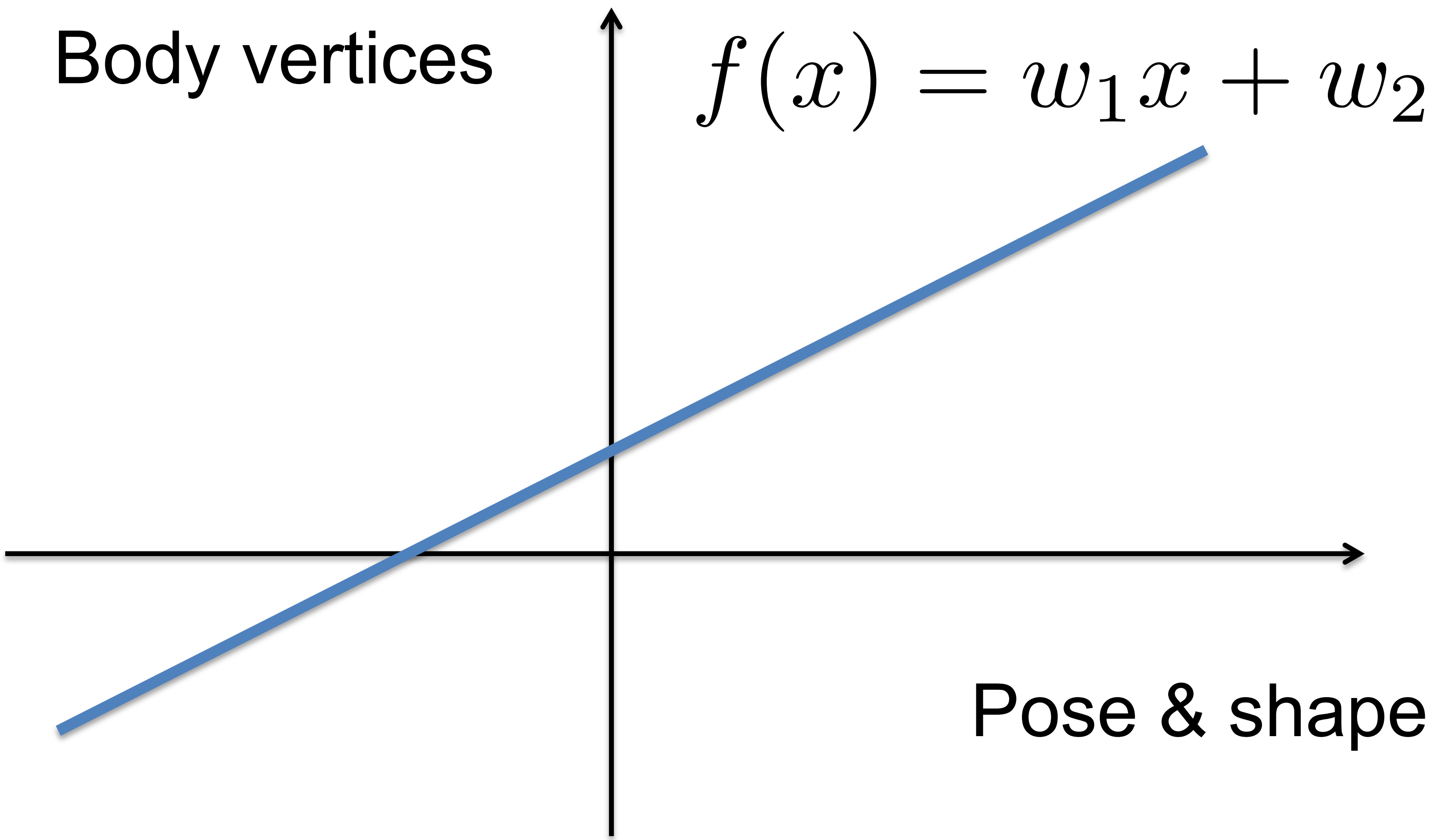
Animation

A Body Model is a function



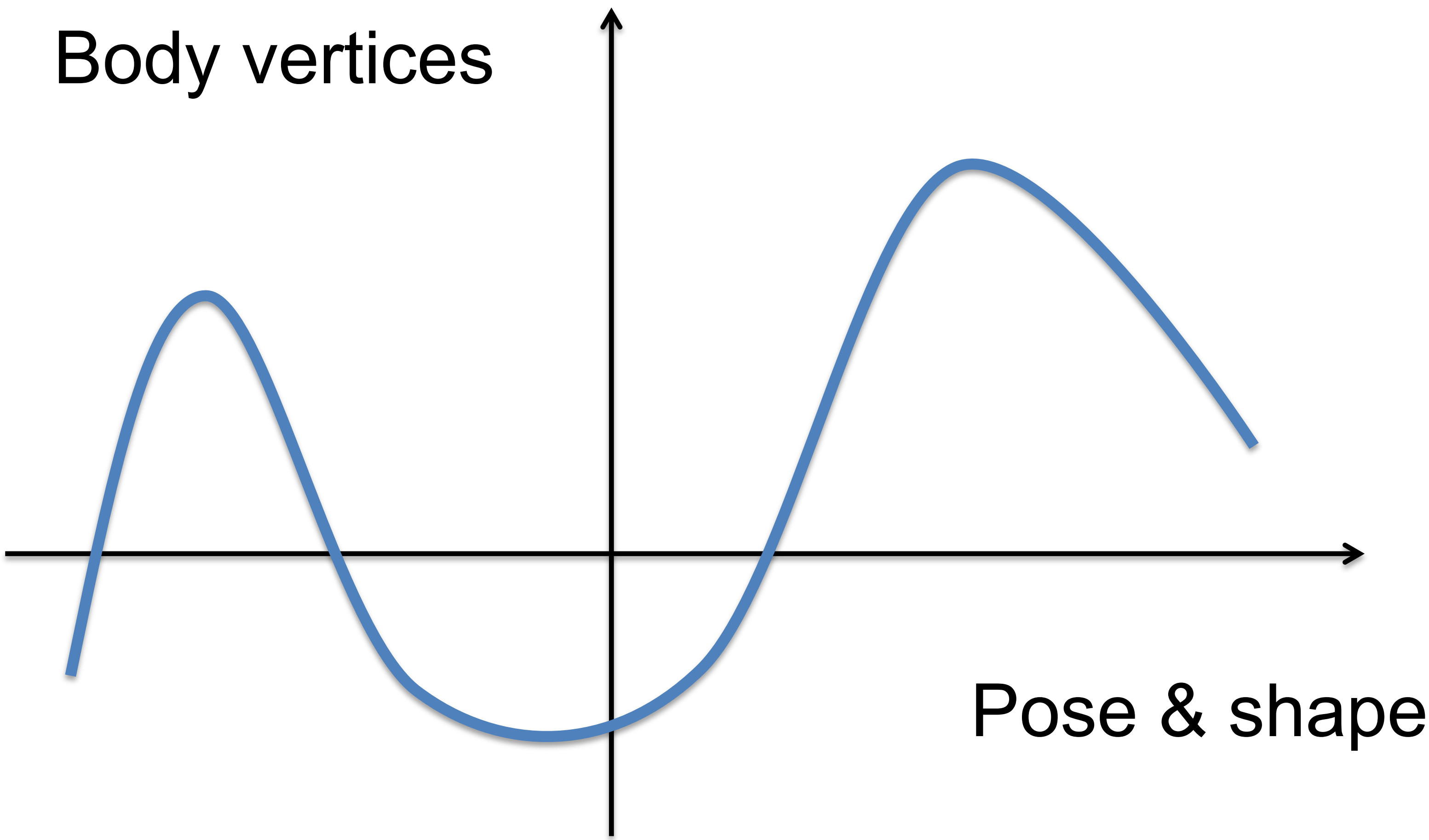
$$\mathbf{X} = \{\mathbf{X}_{\text{pose}}, \mathbf{X}_{\text{shape}}\}$$

What kind of function ?



Linear ?

What kind of function ?



Polynomial ?

Given the function, what **w** ?

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

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

Input parameters

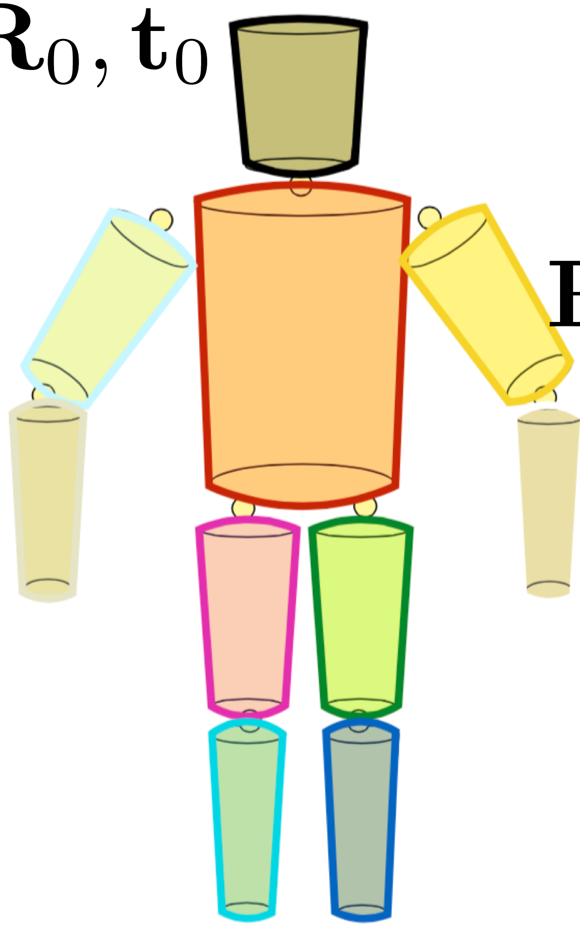
Hyper-parameters

And also why our input **X** is shape and pose ?

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

How do we parameterize pose ?

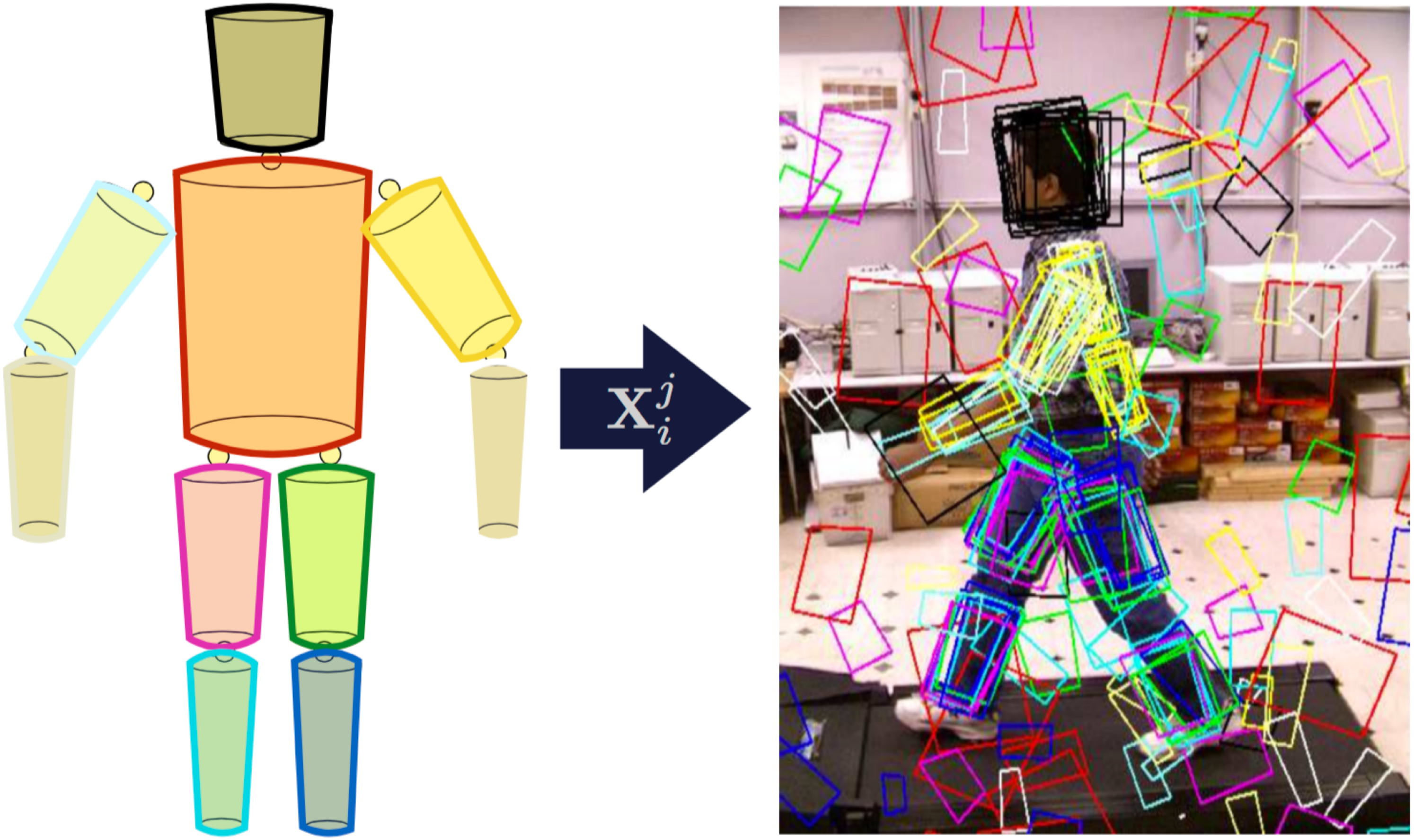
- Parameterize every body part separately ?



$$\mathbf{X}_{\text{pose}} = \{\mathbf{R}_0, \mathbf{t}_0, \ldots \mathbf{R}_N, \mathbf{t}_N\}$$

Problems ?

How do we parameterize pose?



Articulated constraints not satisfied!

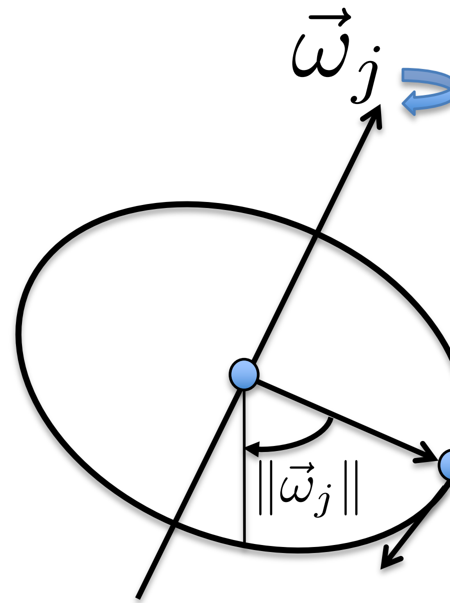
Rotation parameterization

- 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

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

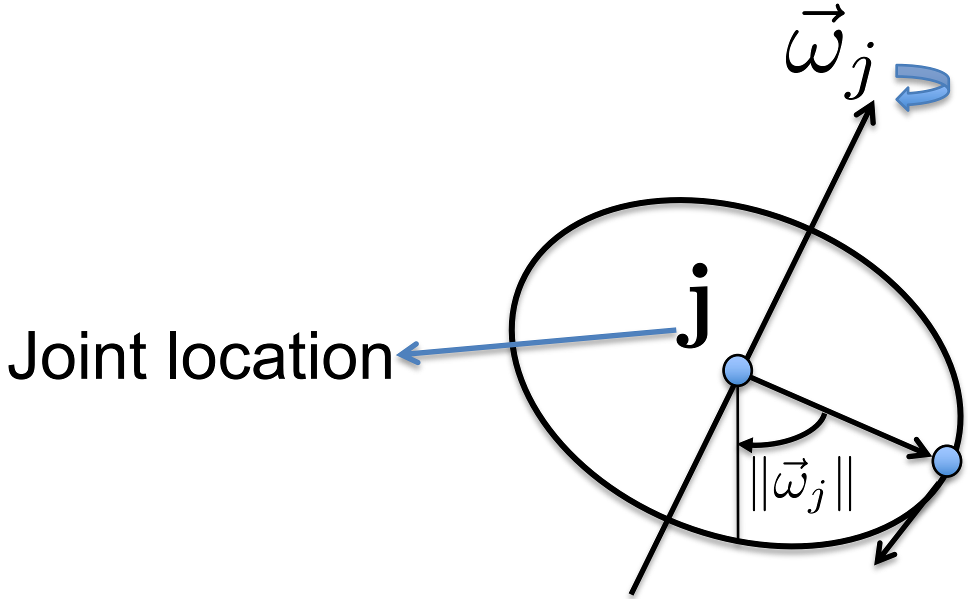


Rotation obtained with Rodrigues formula:

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

Joint Rigid Body Motion

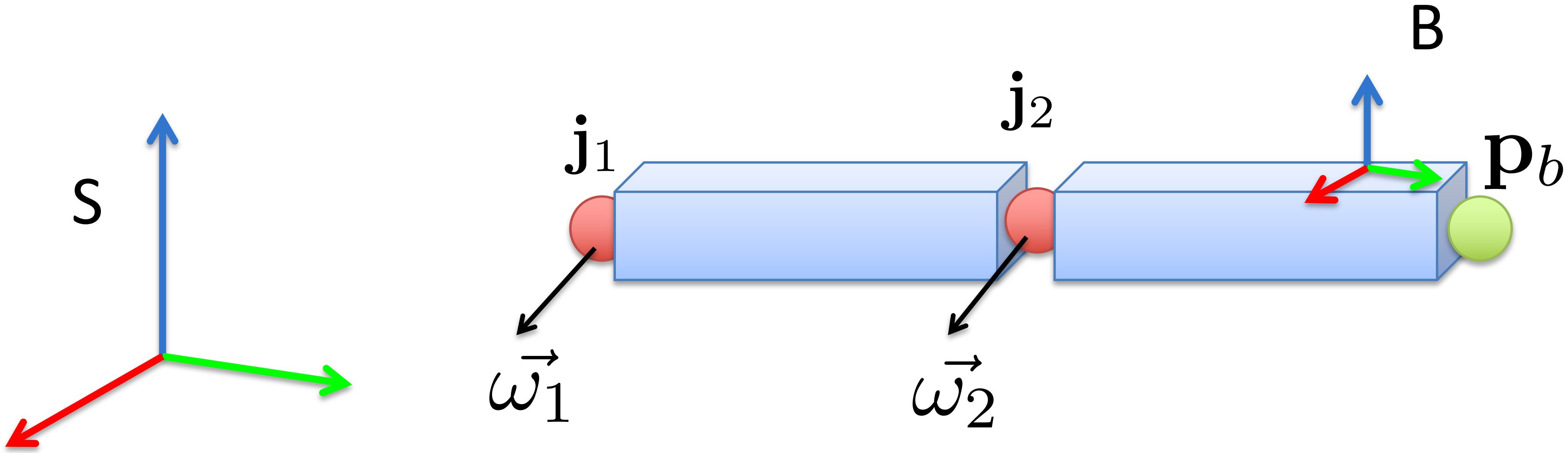
The transformation associated with a rotational joint is



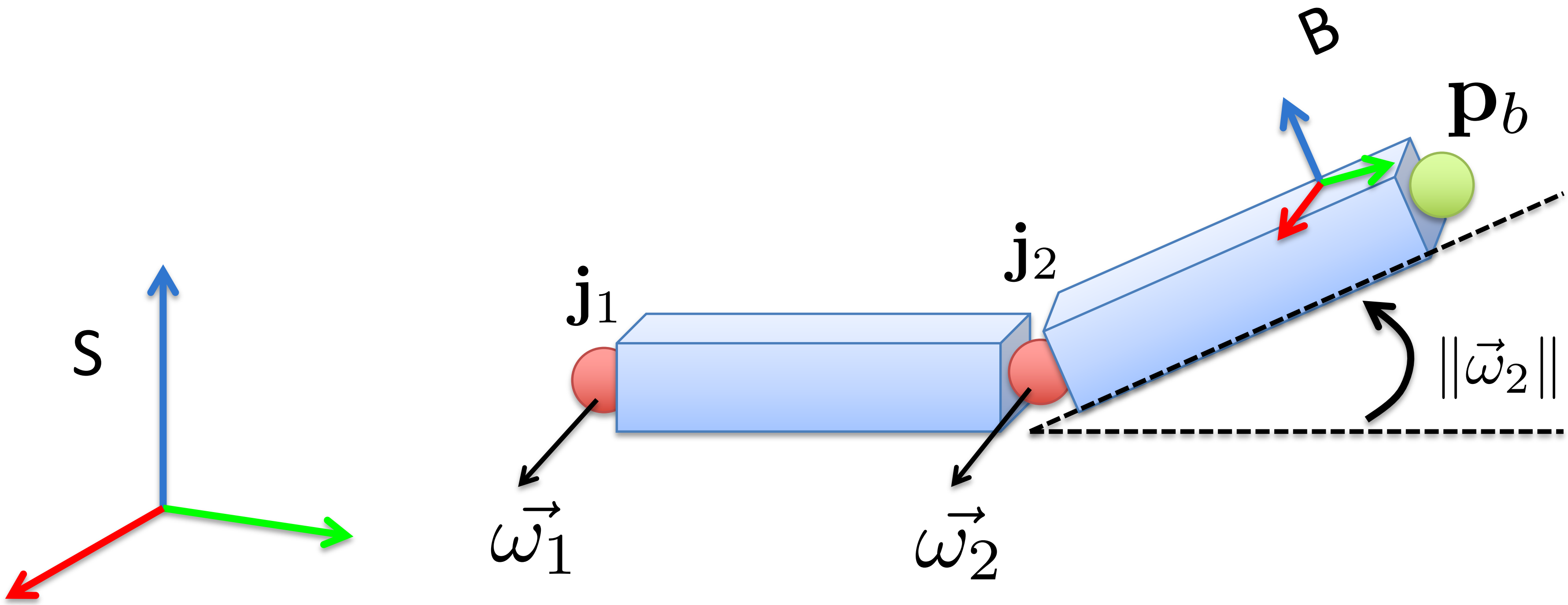
$$G(\vec{\omega}, \mathbf{j}) = \begin{bmatrix} [e^{\vec{\omega}}]_{3\times 3} & \mathbf{j}_{3\times 1} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}$$

Rigid Body Motion

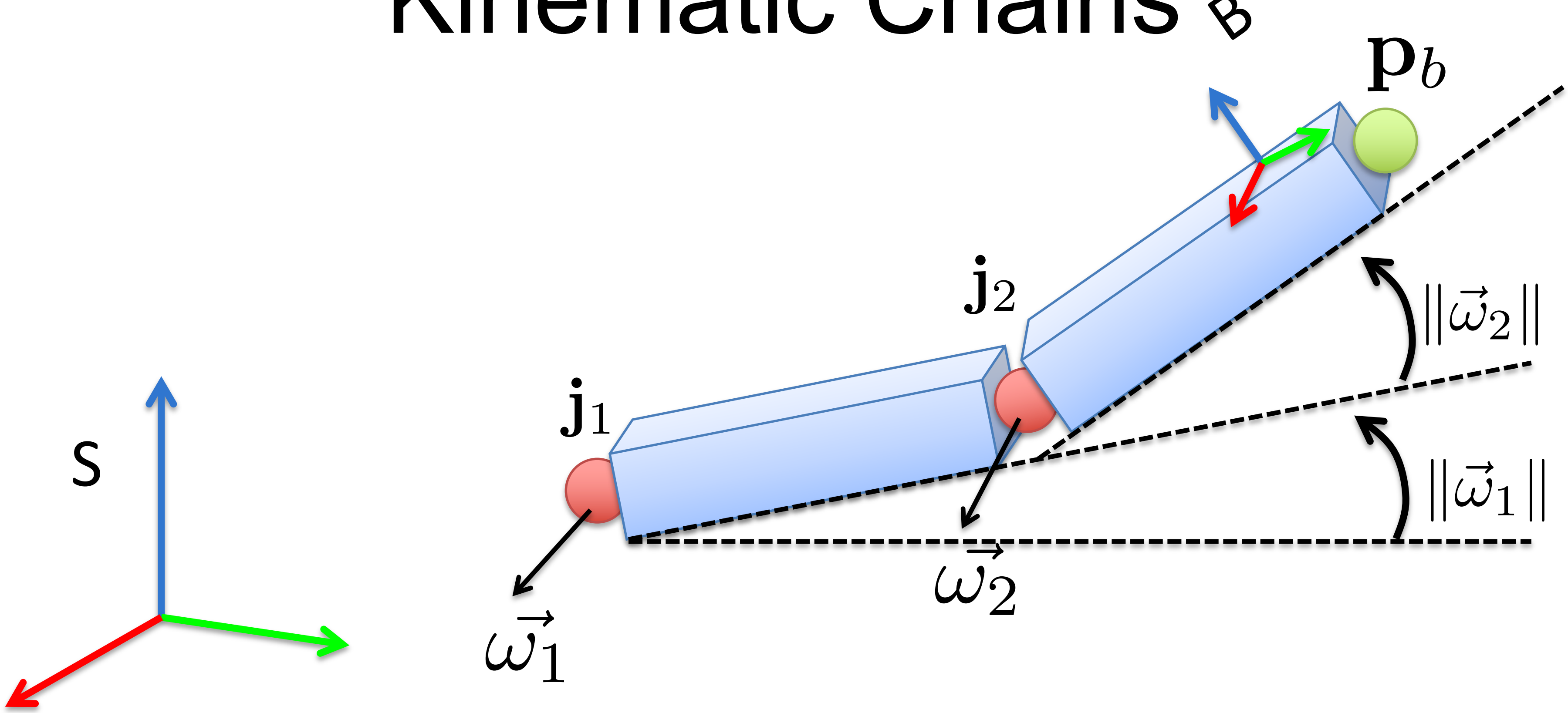
Kinematic Chains



Kinematic Chains



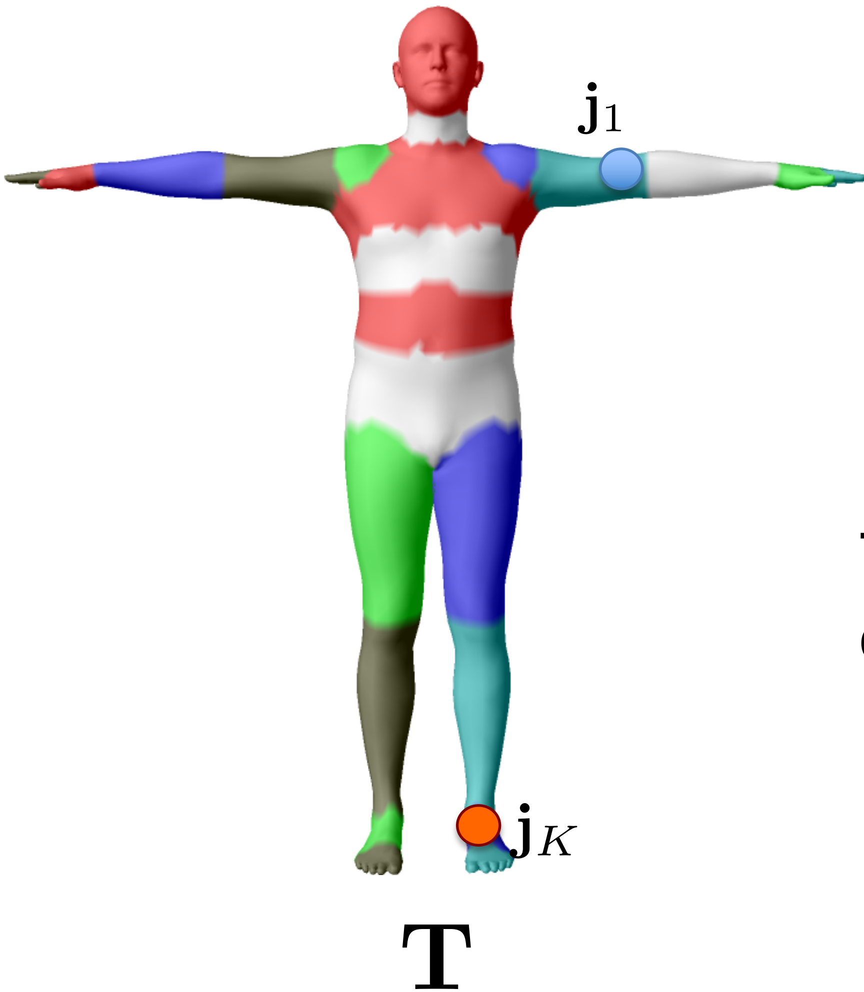
Kinematic Chains



The coordinates of the point in the spatial frame are:

$$\bar{\mathbf{p}}_s = G(\vec{\omega_1}, \vec{\omega_2}, \mathbf{j}_1, \mathbf{j}_2) = G(\vec{\omega_1}, \mathbf{j}_1)\boxed{G(\vec{\omega_2}, \mathbf{j}_2)}\bar{\mathbf{p}}_b$$

Pose Parameters



Given a set of joint locations

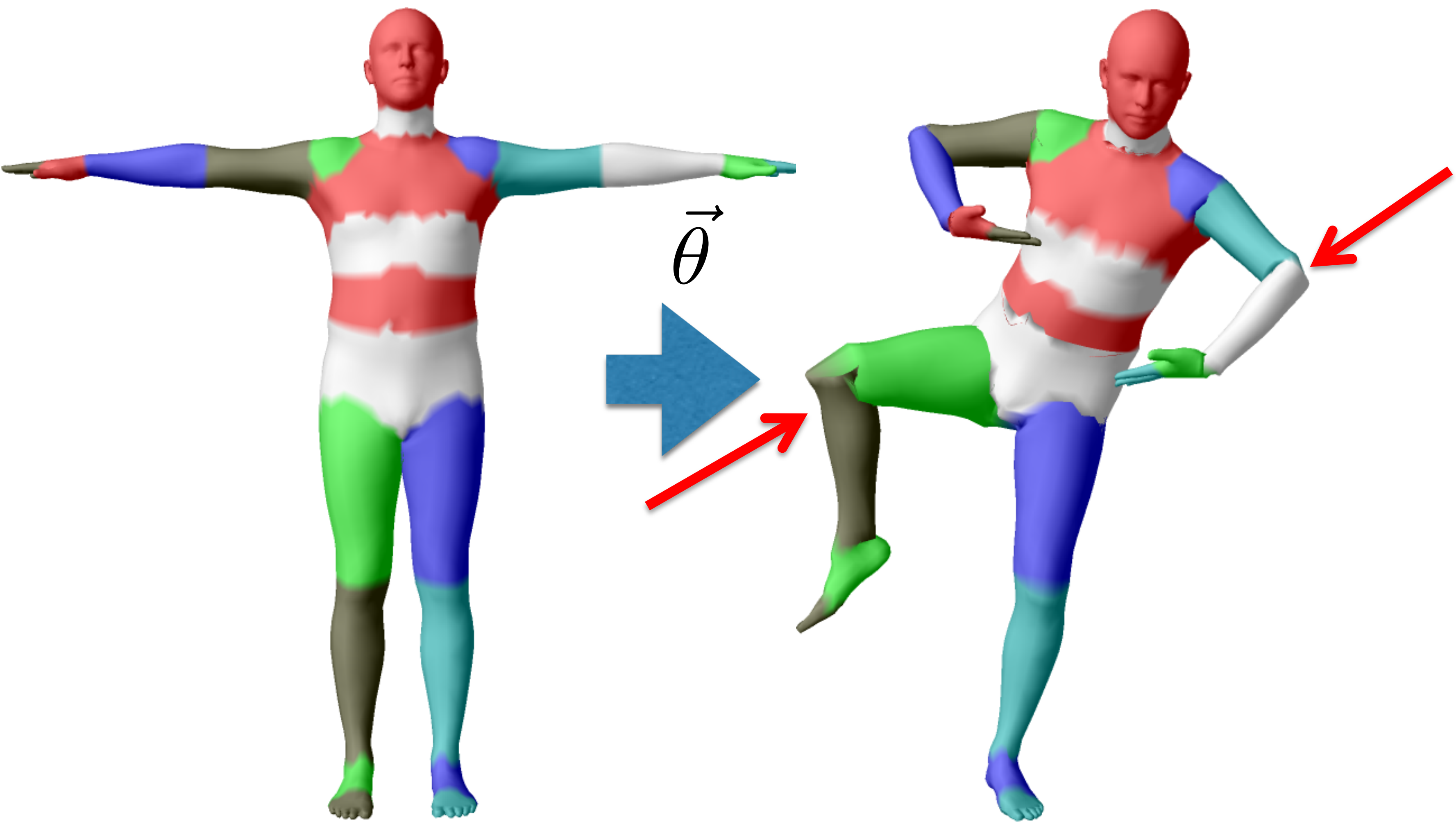
$$\mathbf{J} = (\underline{\mathbf{j}_1}, \ldots, \underline{\mathbf{j}_K})^T$$

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

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

Pons-Moll & Rosenhahn 2011
Model-based Pose Estimation. Looking at People.

Kinematic Chain Problems

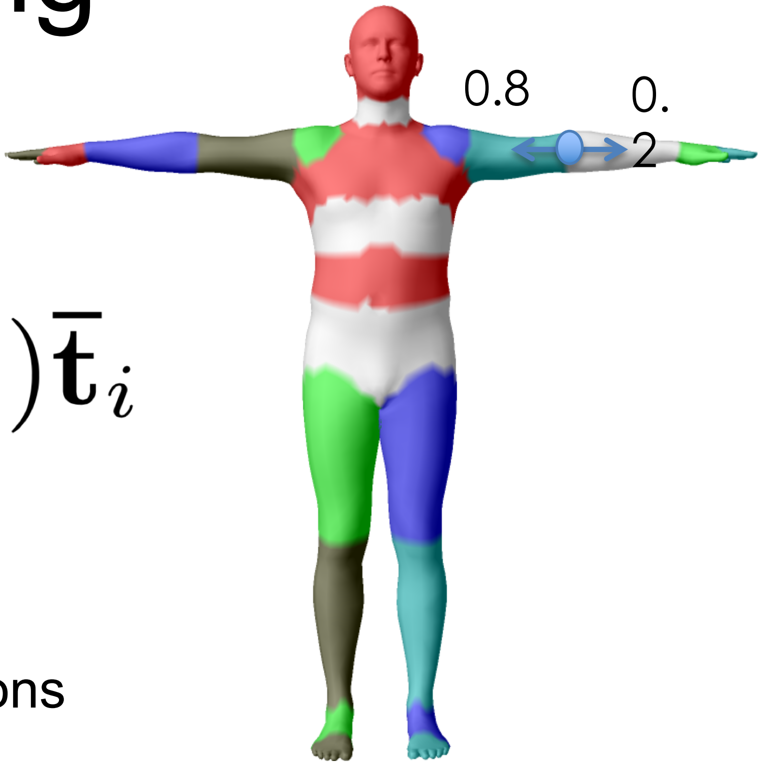


Different poses

- Different poses using no blendweights

  >>python visualize_ablated_smpl.py

Linear Blend Skinning



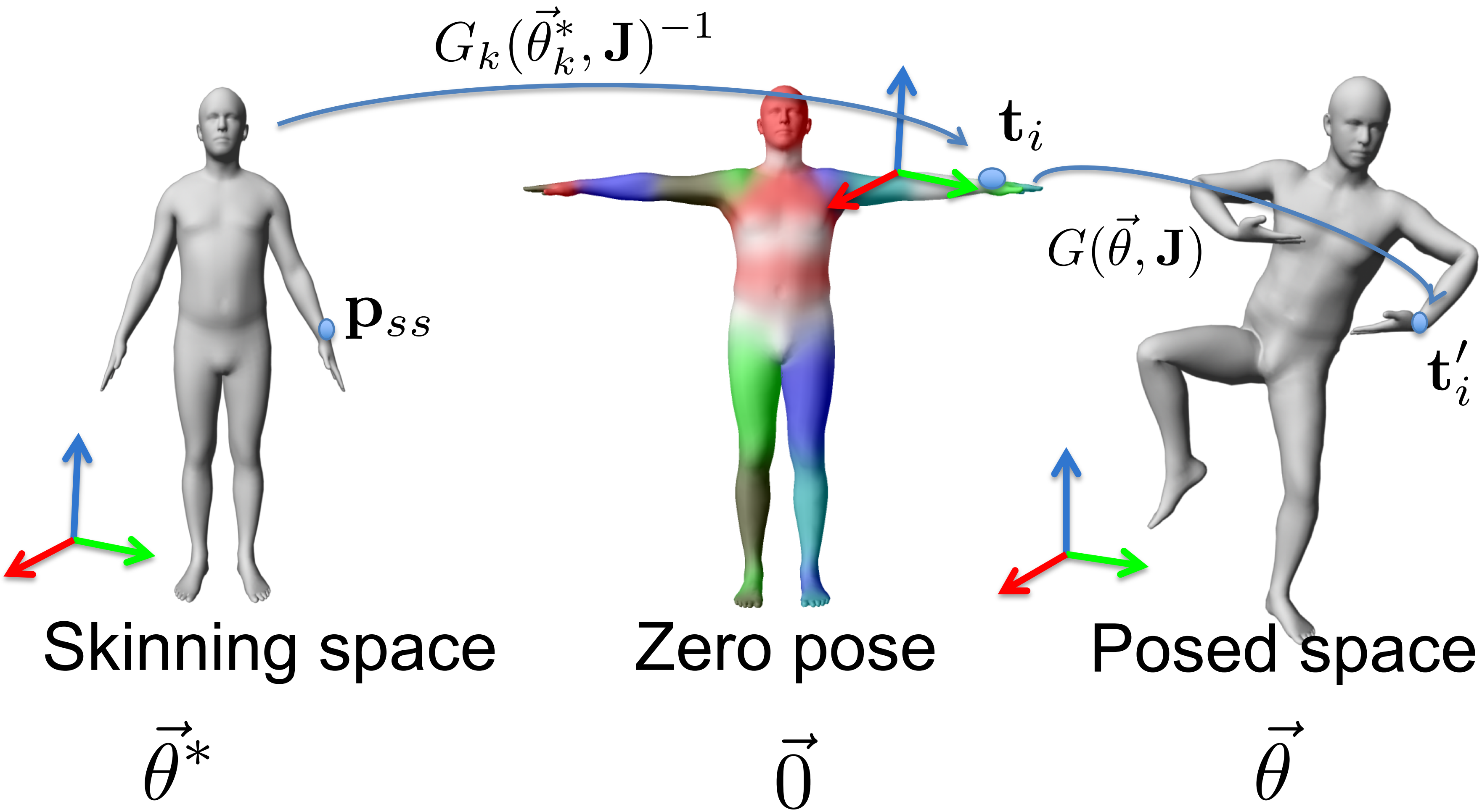
$$\bar{\mathbf{t}}'_i = \sum_{k=1}^{K} w_{k,i} G'_k(\vec{\theta}, \mathbf{J}) \bar{\mathbf{t}}_i$$

Blend weights

Part transformations

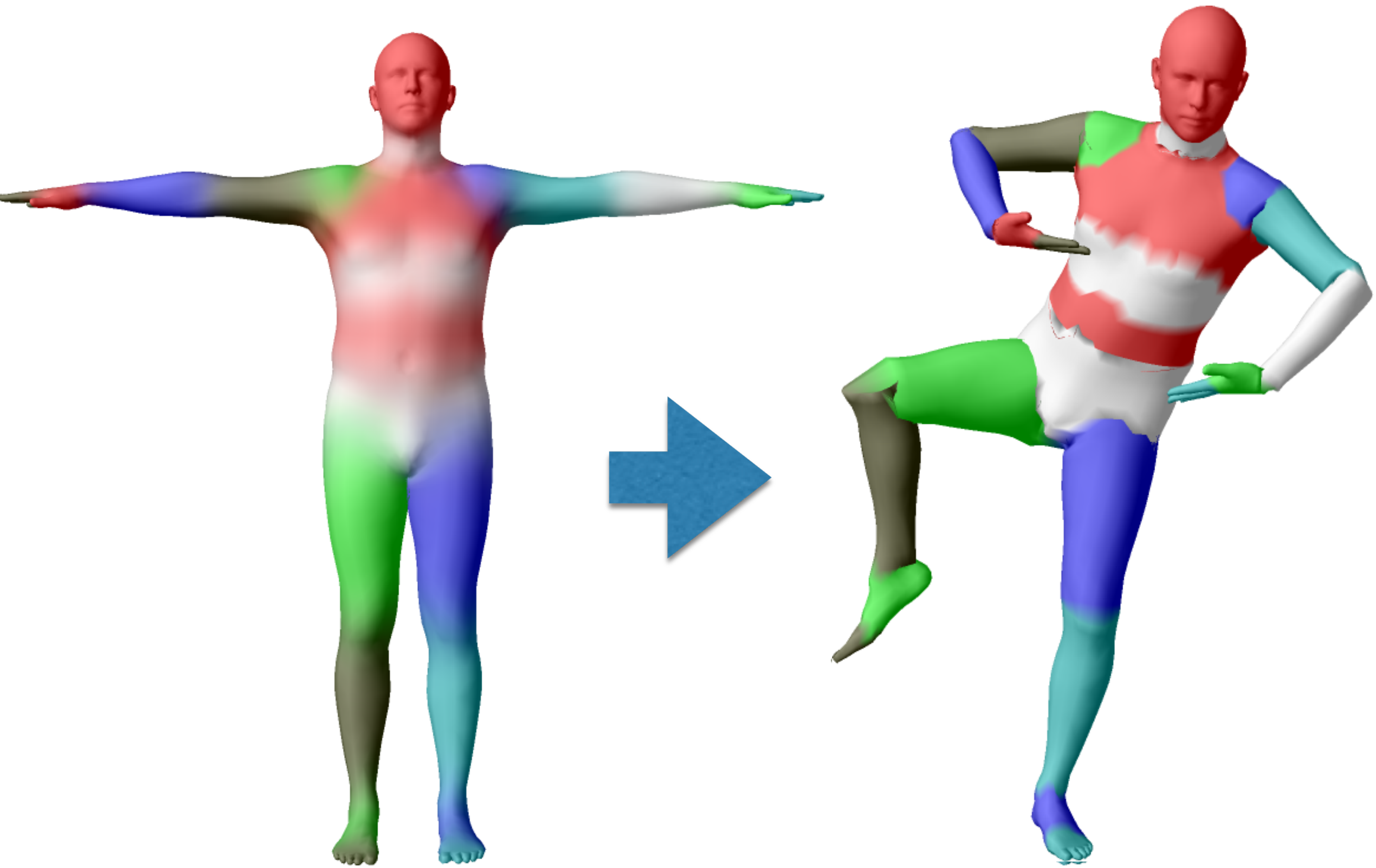
Points transformed as blended linear combination
of joint transformation matrices

Binding Matrices



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

Linear Blend Skinning



Different poses using BW

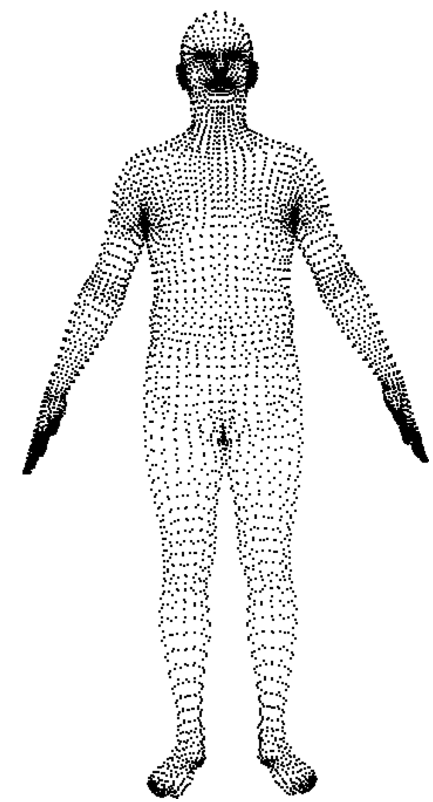
- Different poses using no blendweights

  >>python visualize_ablated_smpl.py

Standard Skinning

Standard skinning produces vertices from…

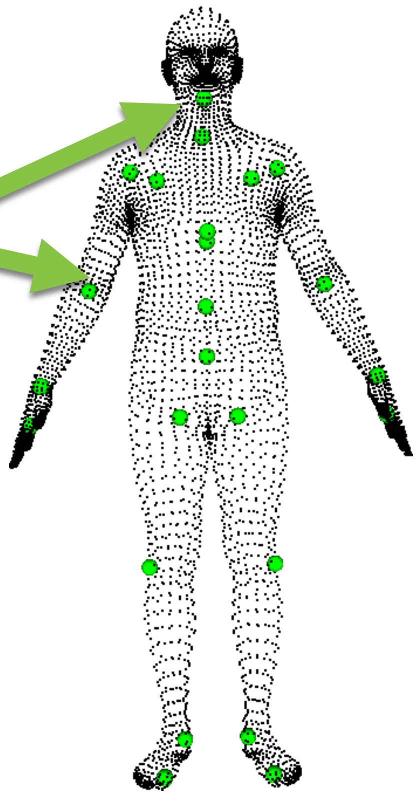
- Rest pose vertices: $\mathbf{T} \in \mathbb{R}^{3N}$
- Joint locations: $\mathbf{J} \in \mathbb{R}^{3K}$
- Weights: $\mathcal{W} \in \mathbb{R}^{N \times K}$
- Pose parameters: $\vec{\theta} \in \mathbb{R}^{3K}$



Standard Skinning

Standard skinning produces vertices from…

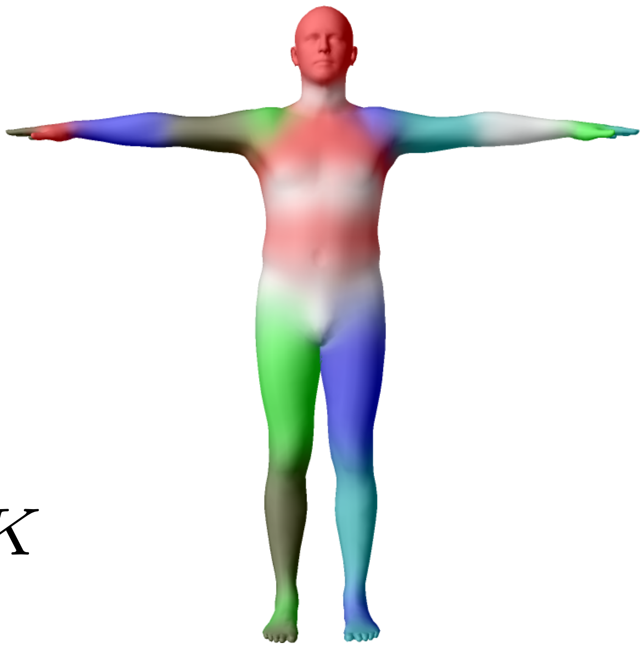
- Rest pose vertices: $\mathbf{T} \in \mathbb{R}^{3N}$
- $\boxed{\text{Joint locations: } \mathbf{J} \in \mathbb{R}^{3K}}$
- Weights: $\mathcal{W} \in \mathbb{R}^{N \times K}$
- Pose parameters: $\vec{\theta} \in \mathbb{R}^{3K}$



Standard Skinning

Standard skinning produces vertices from…

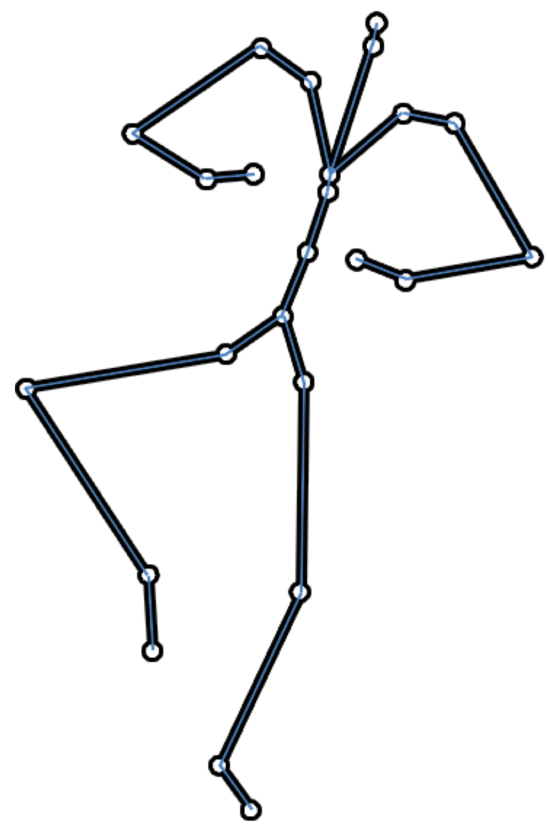
- – Rest pose vertices: $\mathbf{T} \in \mathbb{R}^{3N}$
- – Joint locations: $\mathbf{J} \in \mathbb{R}^{3K}$
- – Weights: $\mathcal{W} \in \mathbb{R}^{N \times K}$
- – Pose parameters: $\vec{\theta} \in \mathbb{R}^{3K}$



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

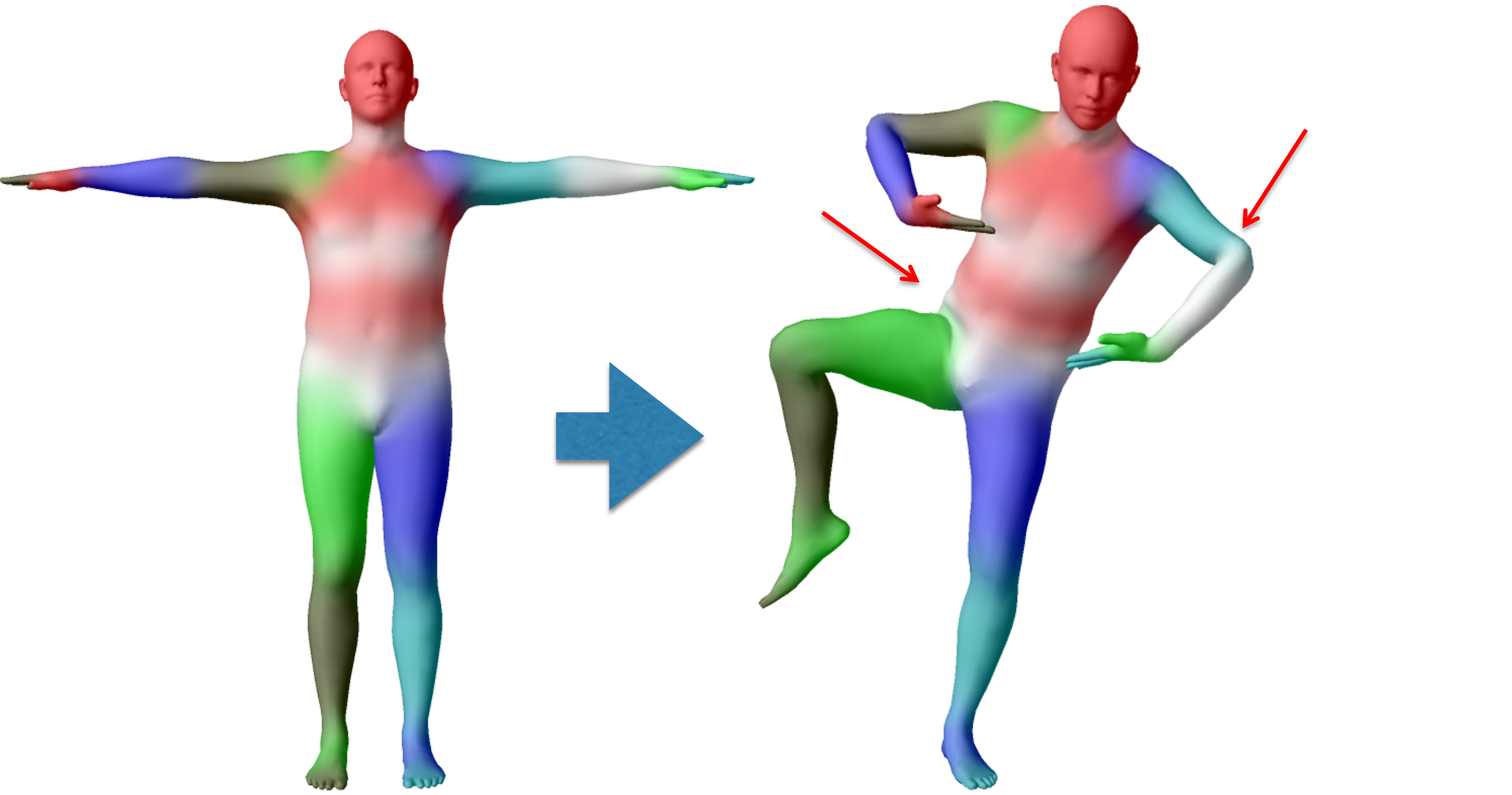


Skinning function

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

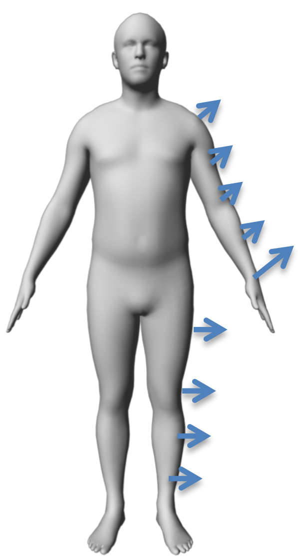
$$\boxed{W(\mathbf{T}, \mathbf{J}, \mathcal{W}, \vec{\theta}) \mapsto \text{vertices}}$$

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

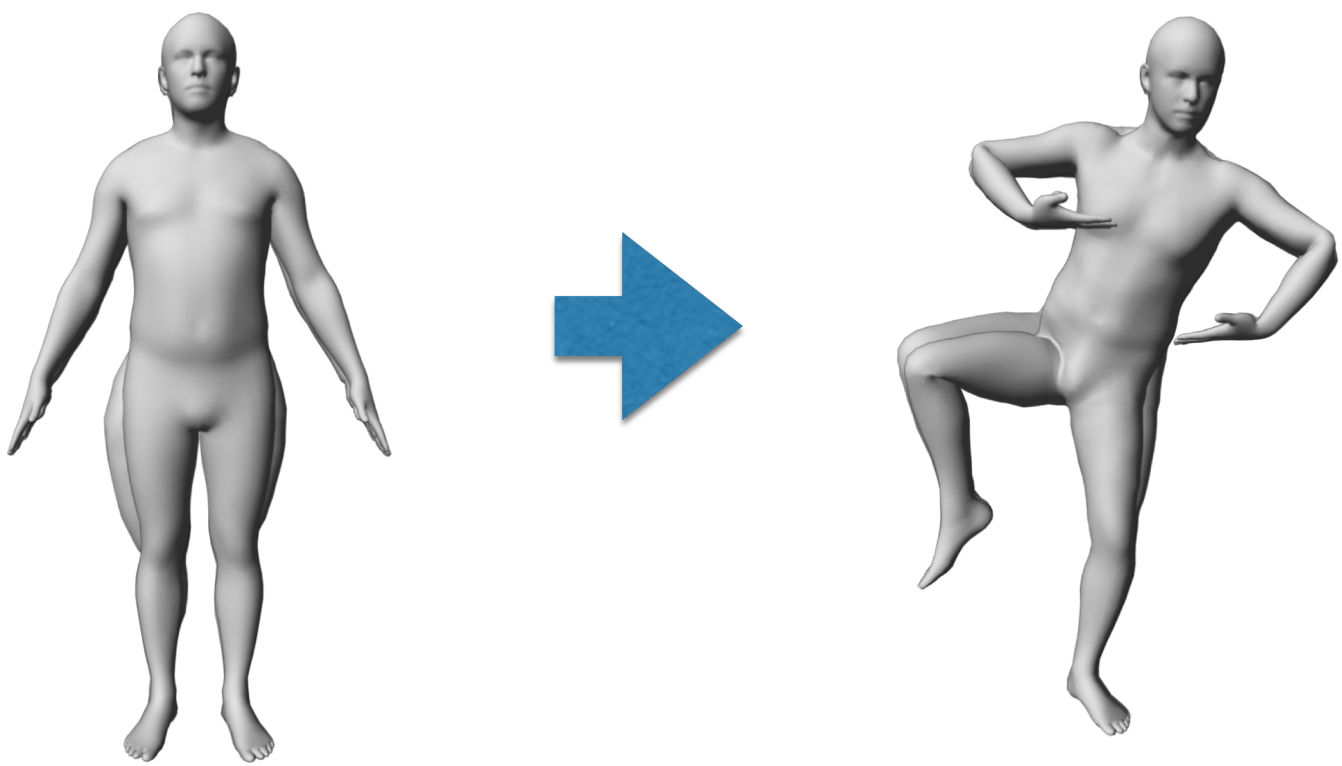


$$\mathbf{P} = \mathrm{vec}\left(\begin{bmatrix} \Delta x_1 & \Delta y_1 & \Delta z_1 \\ & \vdots & \\ & \vdots & \\ \Delta x_N & \Delta y_N & \Delta z_N \end{bmatrix}\right) \in \mathbb{R}^{3N}$$

Offset 1

Pose Blend Shapes

- **With** blend shape correction

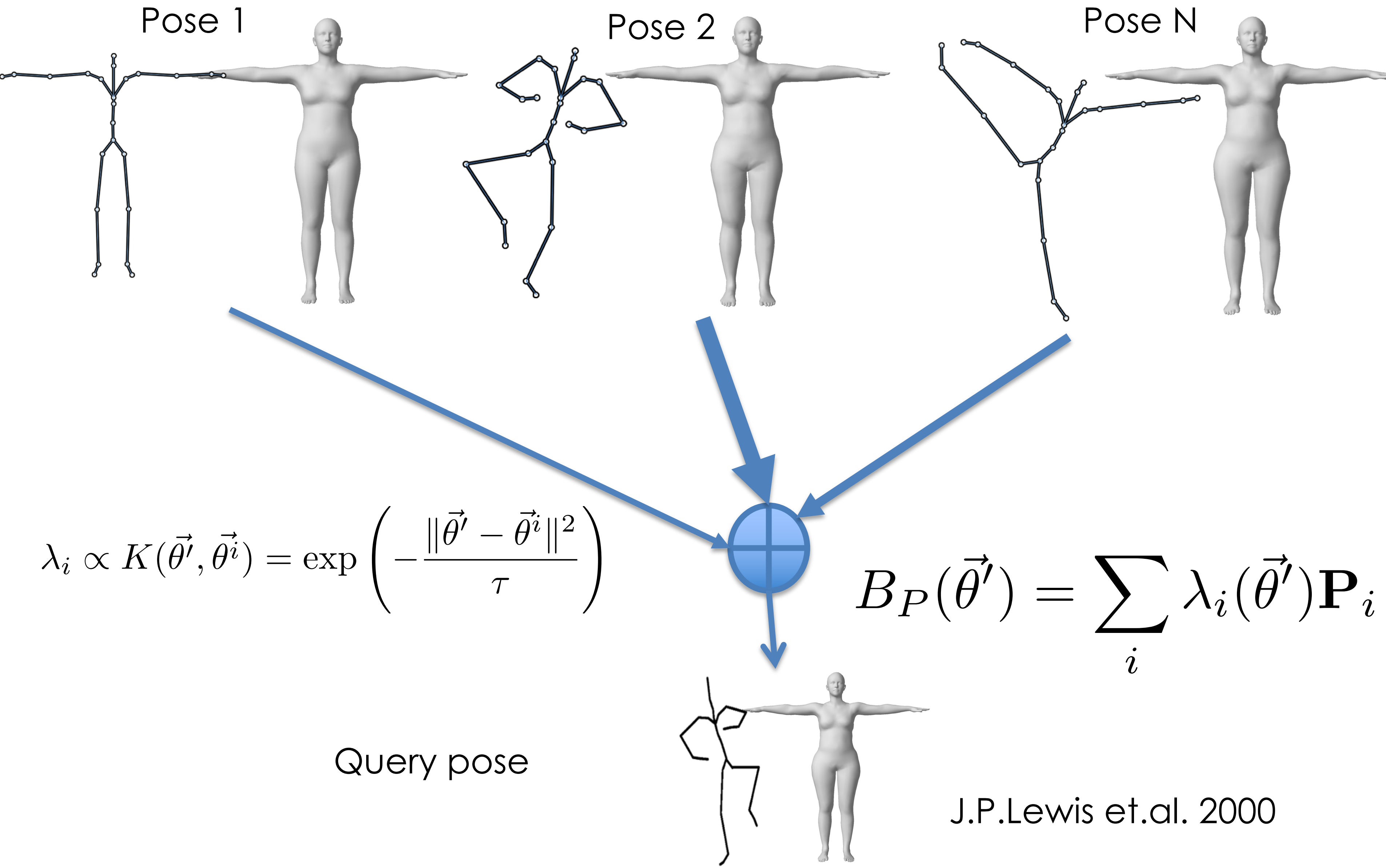


How to predict Blend Shapes ?

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

Can we leverage training data ?

Scattered Data Interpolation



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

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

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 ?

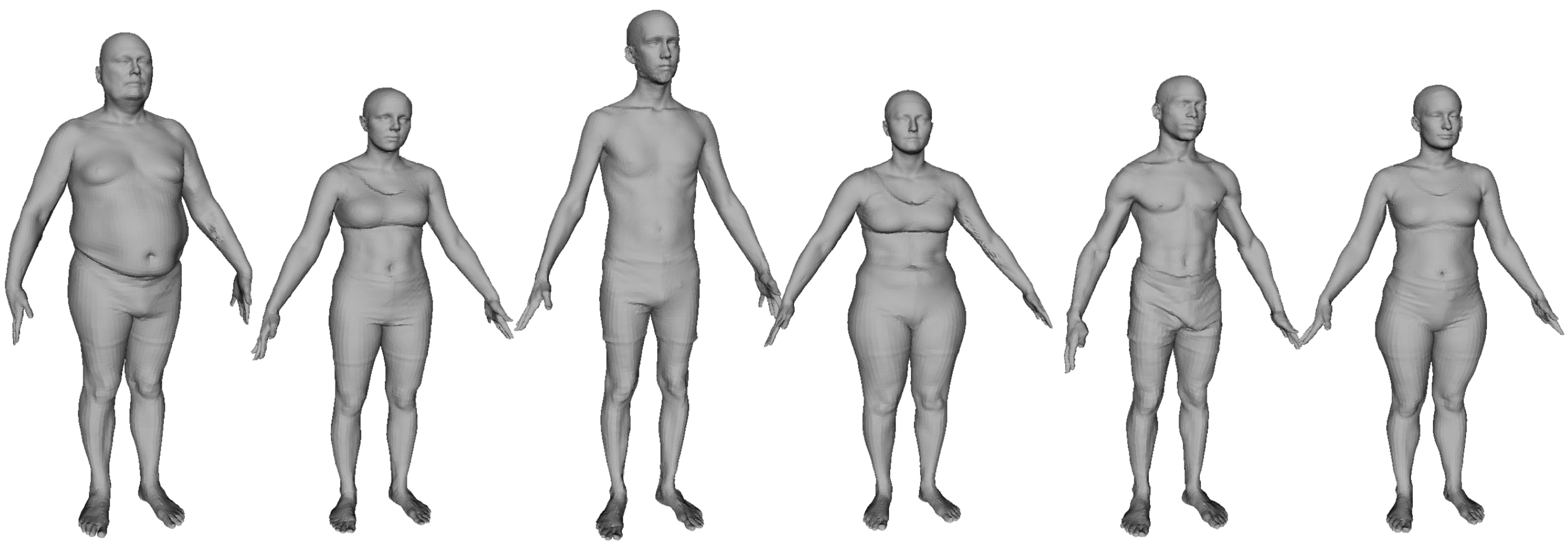
$B_P(\vec{\theta}')$

- How to set the skinning parameters ?

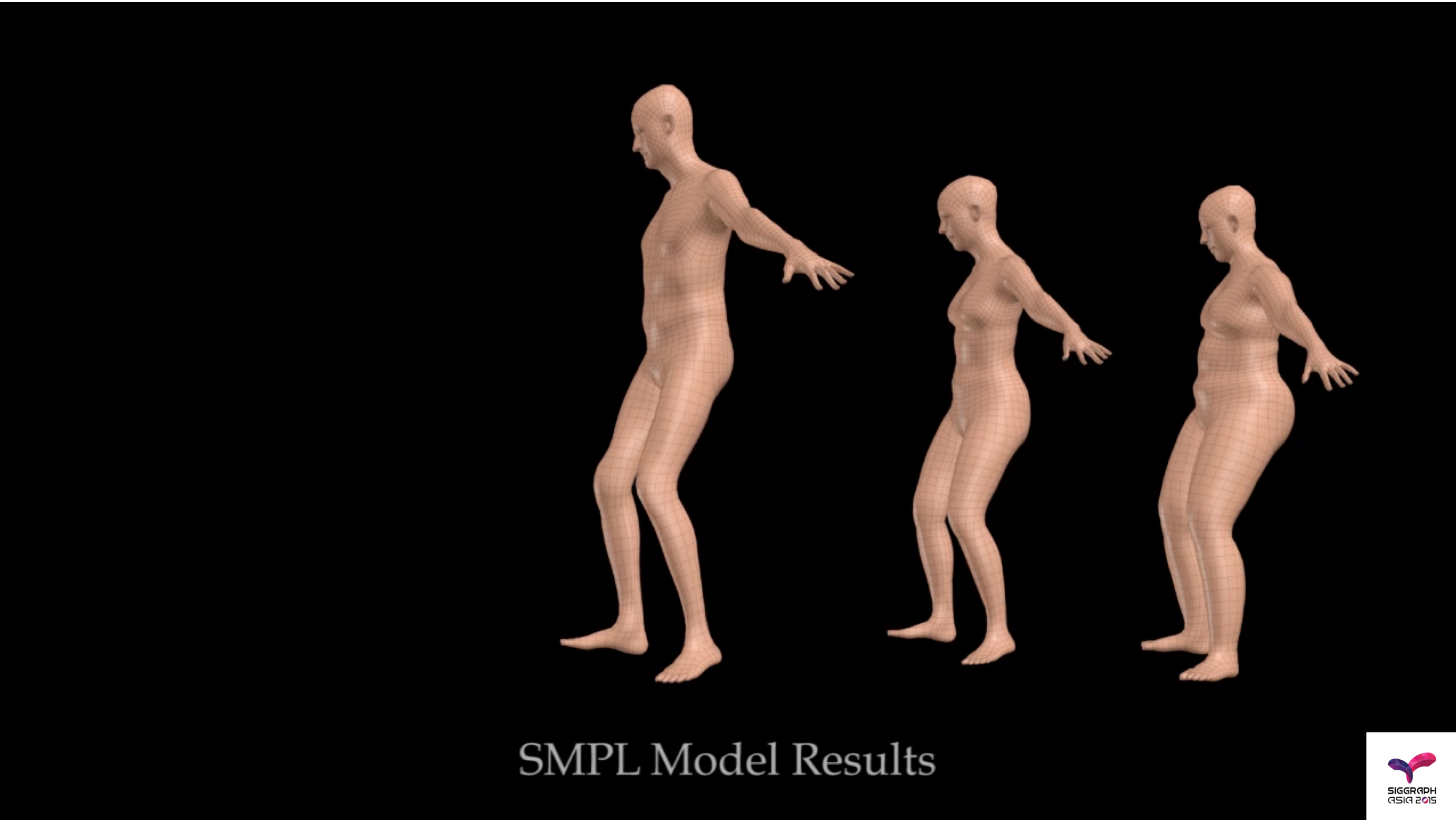
$$\mathbf{T} \in \mathbb{R}^{3N} \quad \mathbf{J} \in \mathbb{R}^{3K} \quad \mathcal{W} \in \mathbb{R}^{N \times K}$$

More Problems

How do we model shape identity variations ?



SMPL



SMPL Philosophy

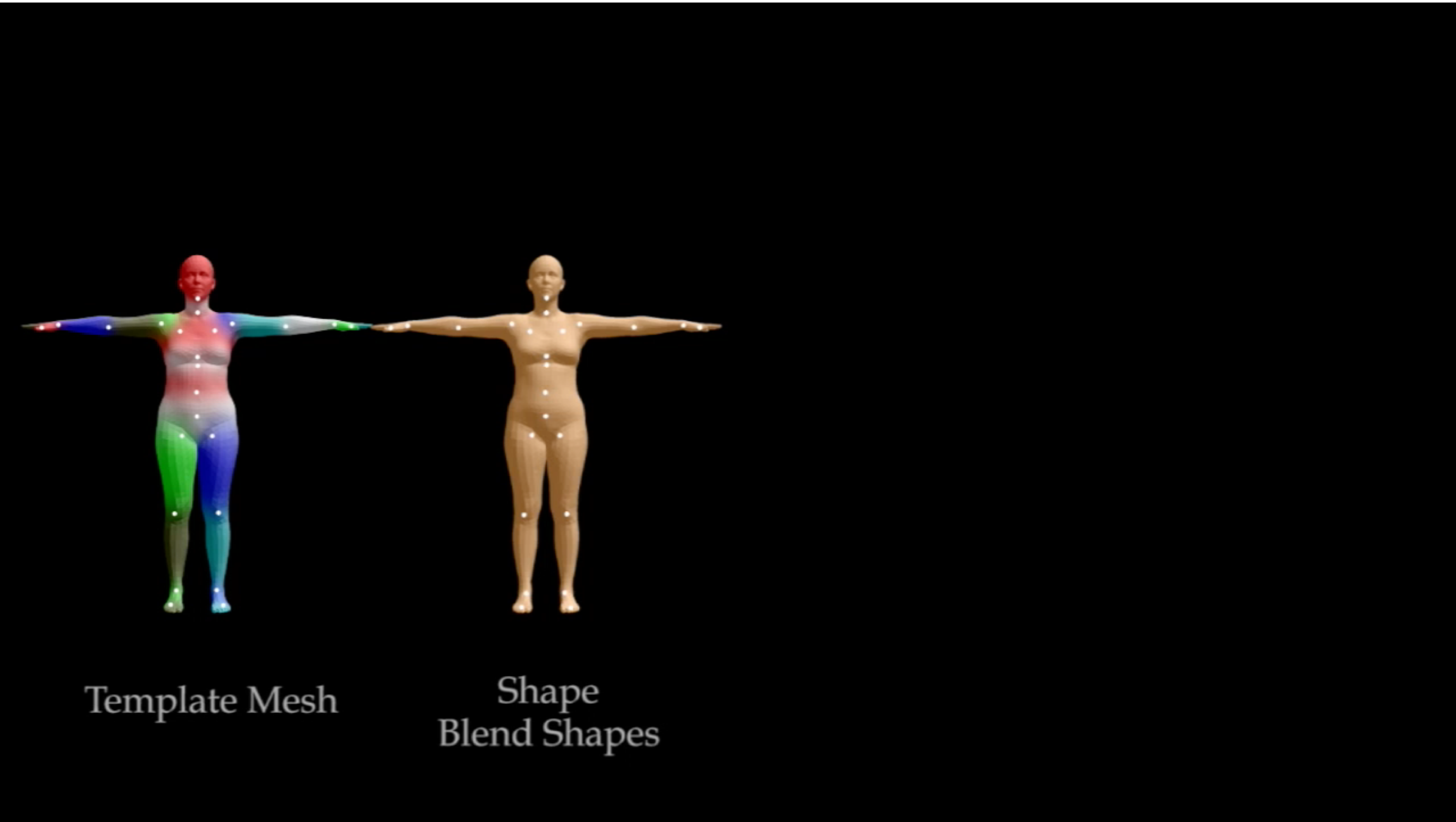
We aim for the simplest possible model while having state-of-the-art performance

- Makes training easier
- Enables compatibility

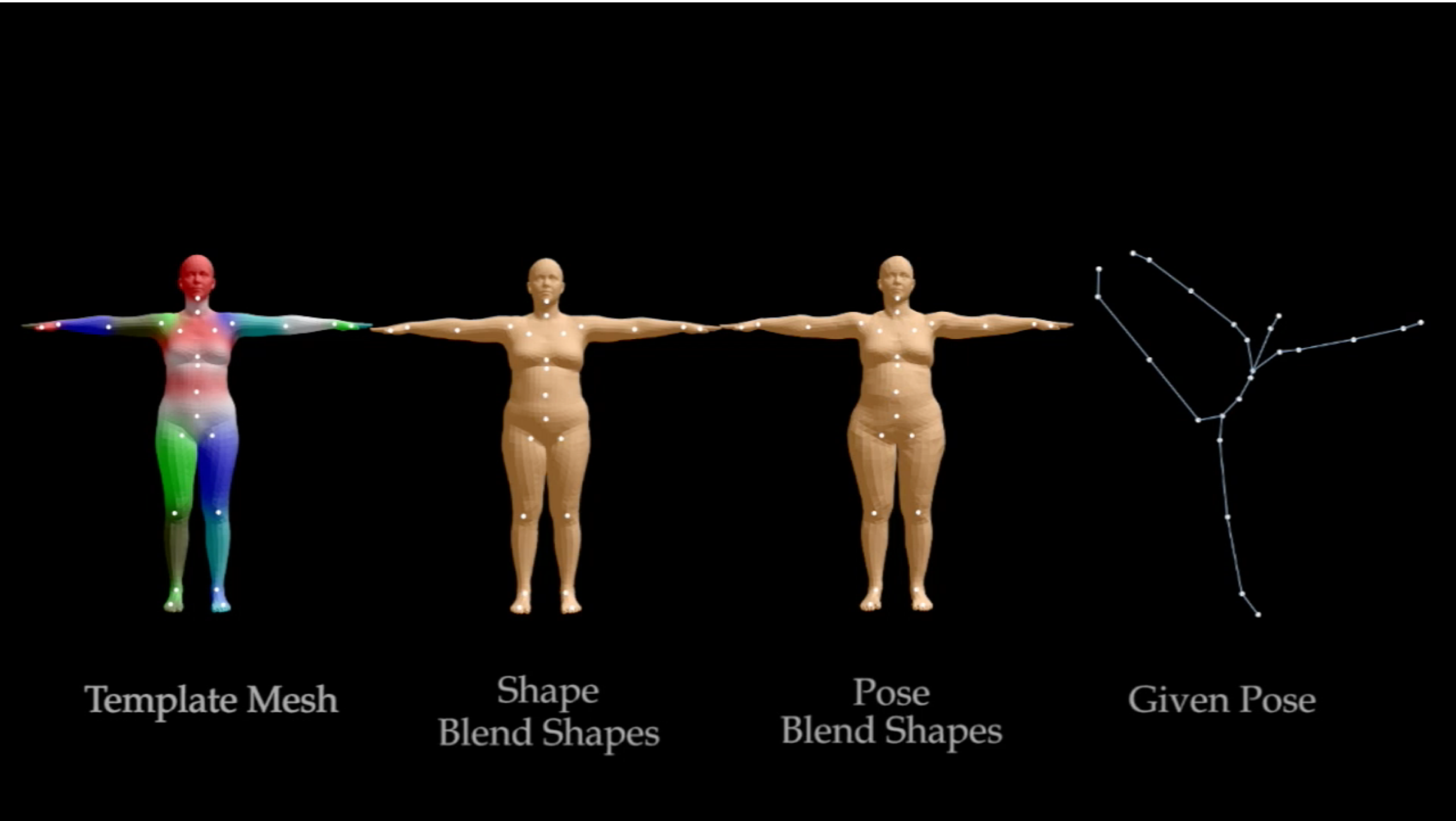
2015 Loper et.al. SIGGRAPH Asia

Pipeline

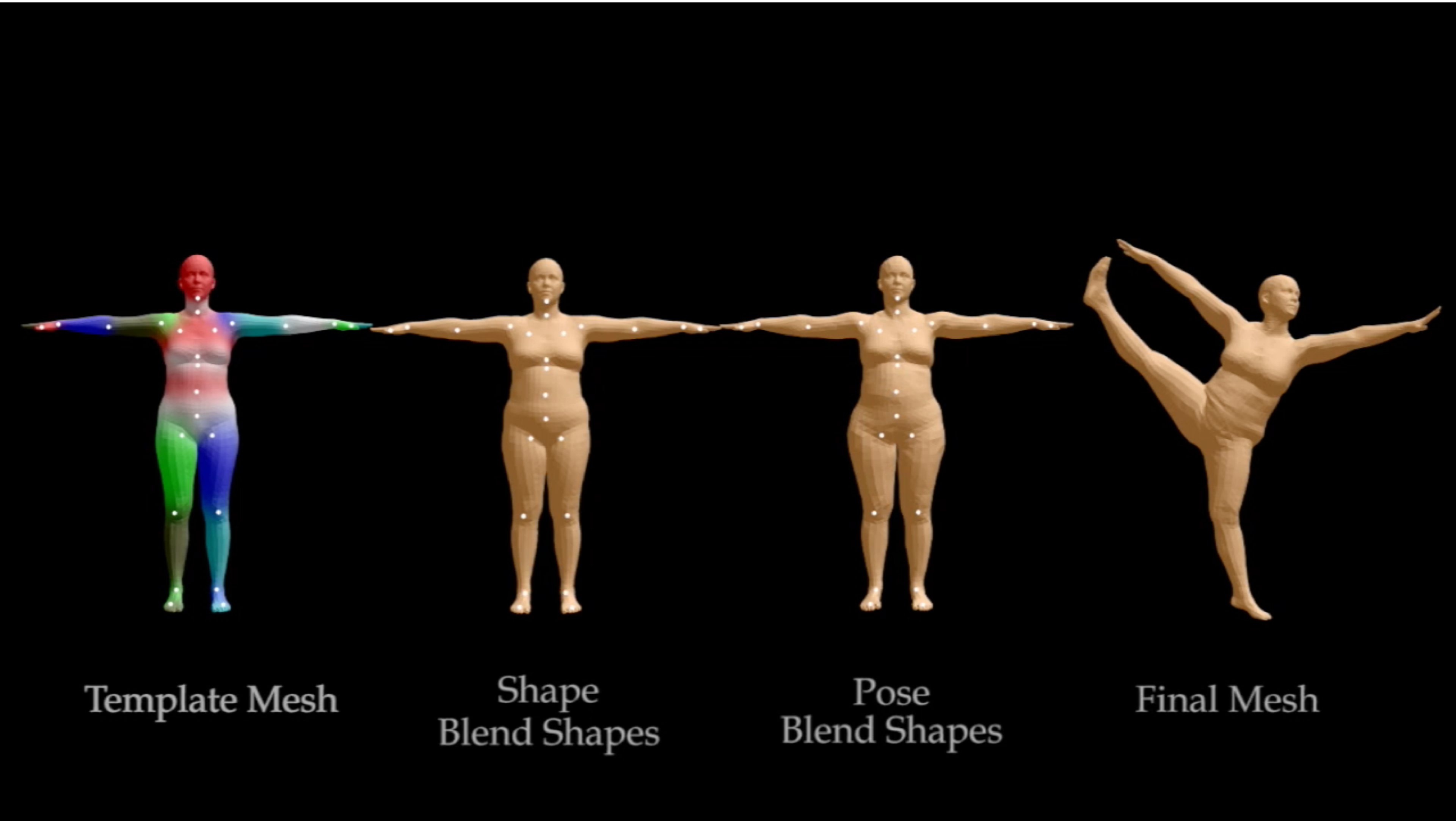
Pipeline



Pipeline

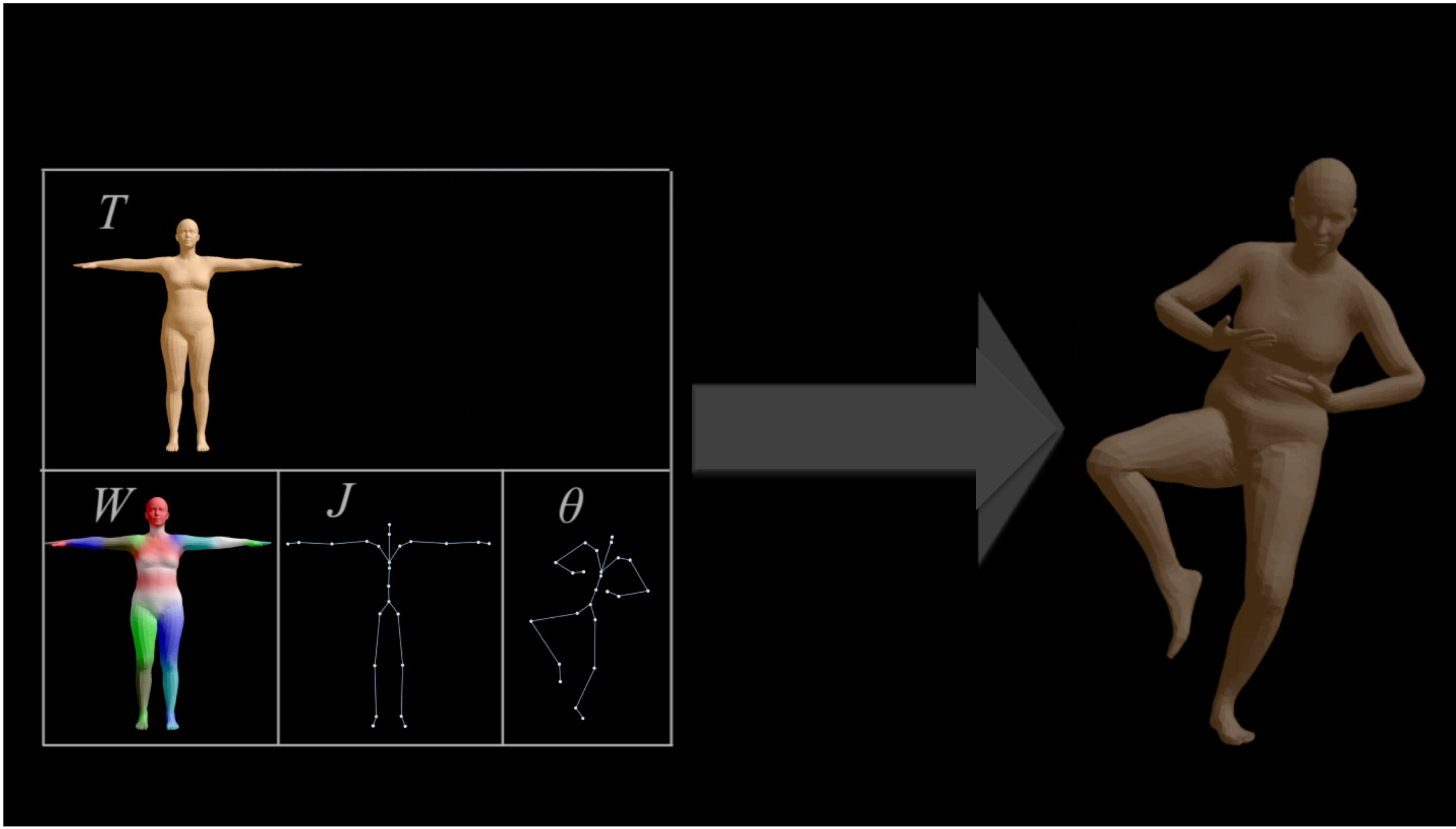

Template Mesh
Shape
Blend Shapes
Pose
Blend Shapes
Given Pose

Pipeline



Template Mesh
Shape
Blend Shapes
Pose
Blend Shapes
Final Mesh

Standard Skinning



Parameterized Skinning

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

SMPL model

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

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

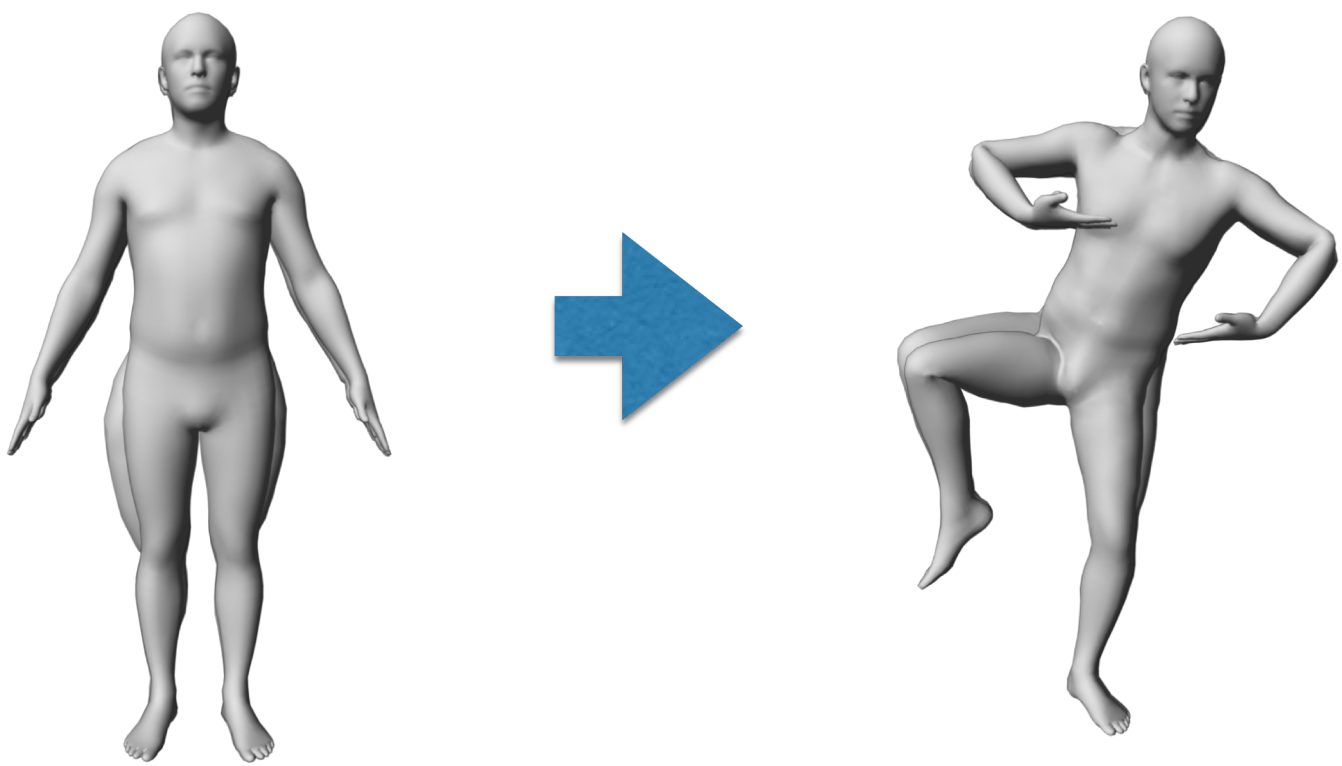
$$W(\mathbf{T}, \mathbf{J}, \mathcal{W}, \vec{\theta})$$

$$\Downarrow$$

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

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

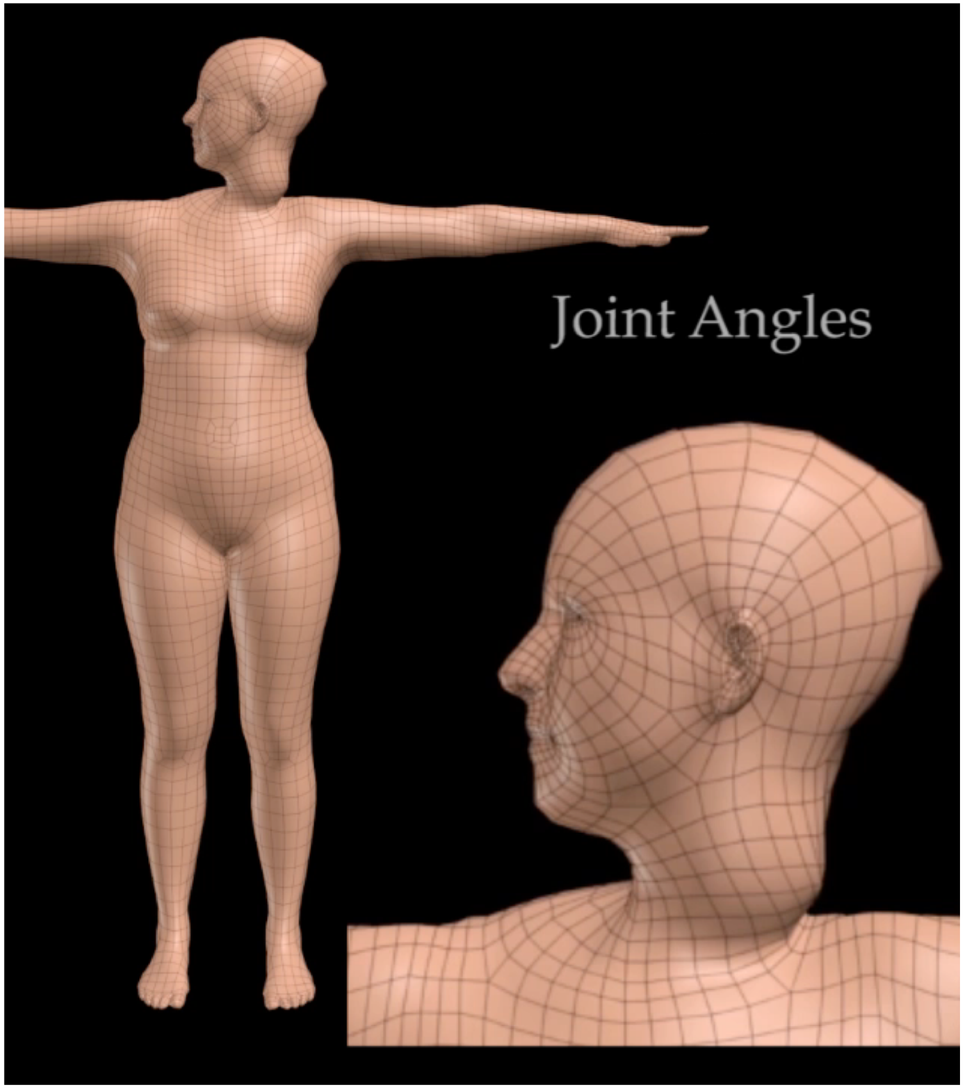
- What function $f(\vec{\theta})$?

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

- Simplest possible:

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

Neck Rotation



Parameterized Skinning

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

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

- Idea: we consider $f(\vec{\theta})$ as the vectorized joint rotation matrices

- Blend shapes are *linear in rotation matrix elements*

Pose Blend Shapes

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

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

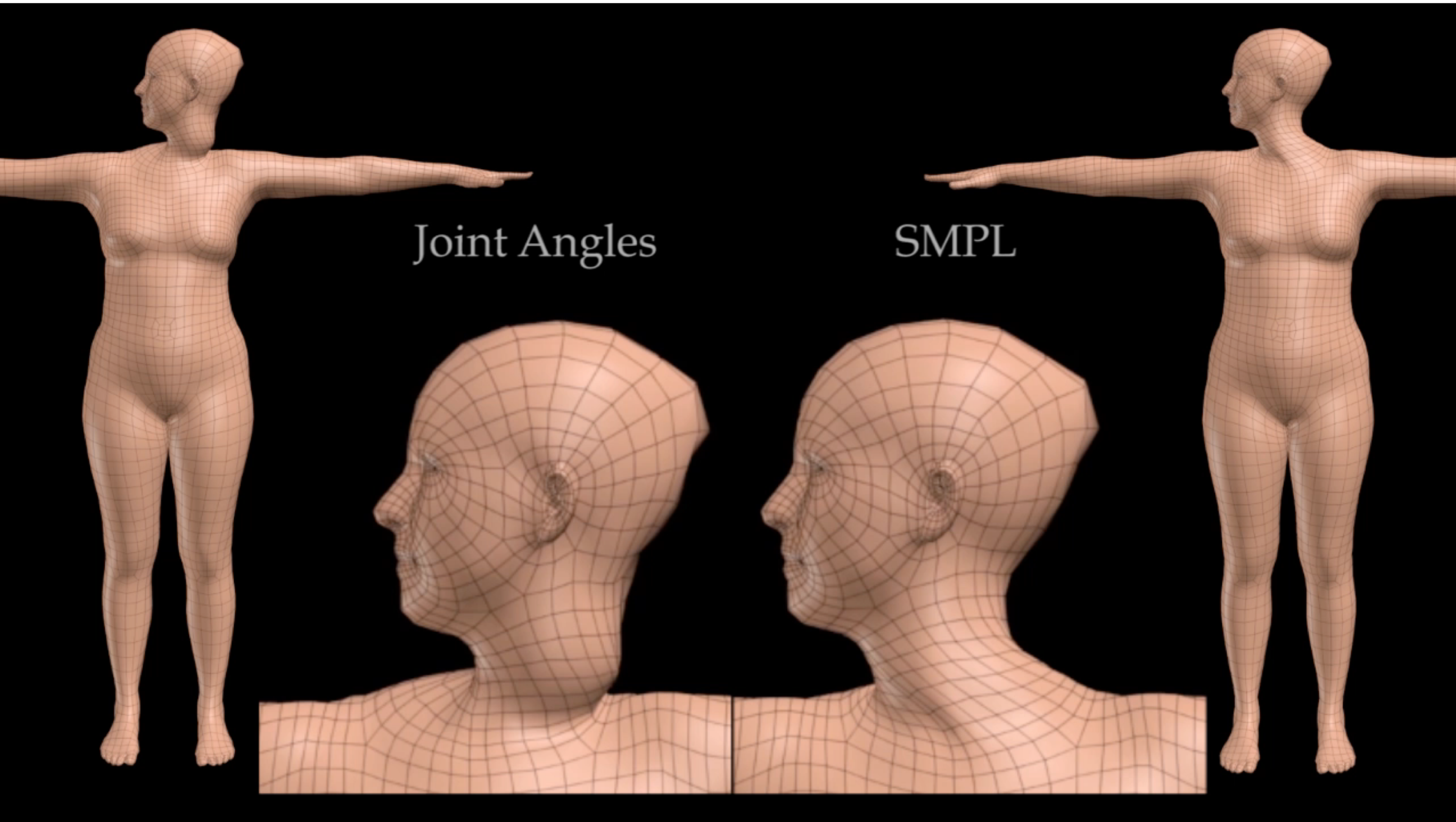
$$e^{\hat{\omega}_1} - \mathcal{I} \qquad e^{\hat{\omega}_K} - \mathcal{I}$$

Not a minus

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

9 elements of the rotation matrix-> We learn 9xK=207 blendshapes

Neck Rotation



Pose Blendshapes demo

- >> python visualize_pose_blends.py

Joint Location Estimation

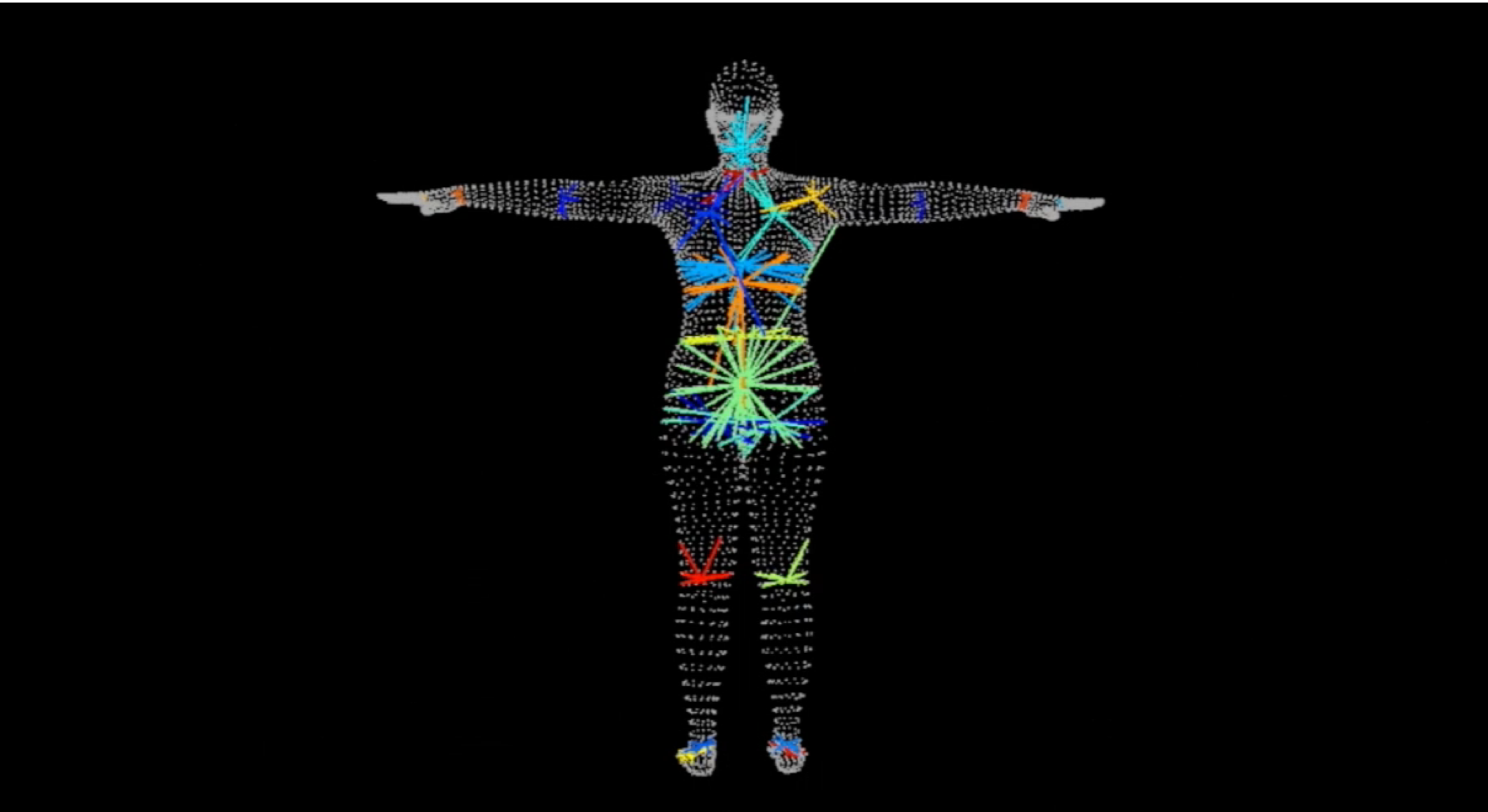
- How to get the joints $\mathbf{J}$ for a new shape? What is the simplest way?

- 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 $\vec{\beta}$

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

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

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

Shape contribution $\left\{ B_S(\beta) = \sum_{j}^{|\beta|} \beta_j S_j \right.$

Shape Blend shape matrix

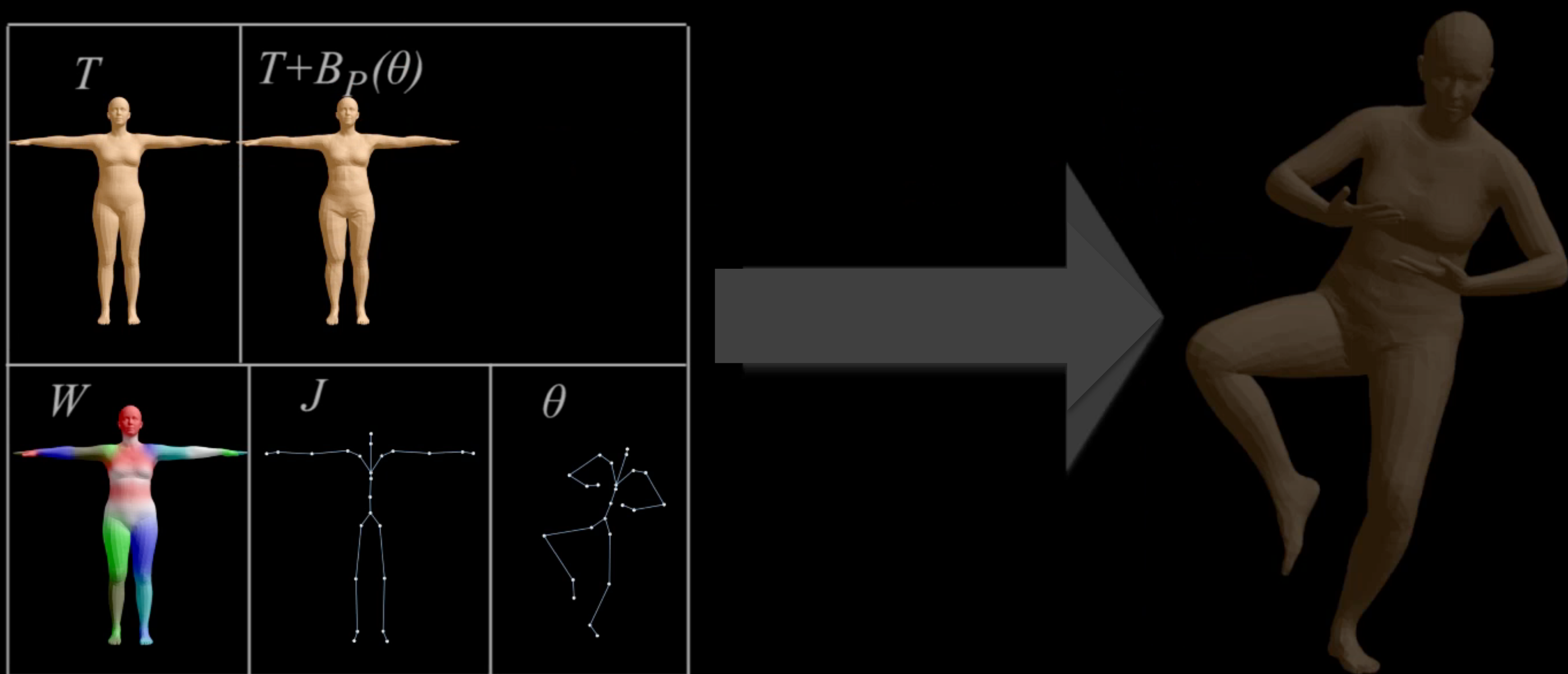
$$\mathcal{S} = [\mathbf{S}_1 \quad \mathbf{S}_2 \quad \ldots \quad \mathbf{S}_{N_{\text{subj}}}]$$

SMPL

Additive Model

$$\bar{\mathbf{t}}'_i = \underbrace{\boxed{\sum_{k=1}^{K} w_{k,i} G'_k(\vec{\theta}, J(\vec{\beta}))}}_{\text{Blendweights}} (\underbrace{\boxed{\bar{\mathbf{t}}_i}}_{\text{Vertices}} + \underbrace{\boxed{\mathbf{b}_{S,i}(\vec{\beta})}}_{\text{Shape-bs}} + \underbrace{\boxed{\mathbf{b}_{P,i}(\vec{\theta})}}_{\text{Pose-bs}})$$

SMPL Skinning



Parameterized Skinning

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

SMPL model

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

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

SMPL

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

pose: $\vec{\theta}$ shape: $\vec{\beta}$

Input

Model parameters to be learned from data

$\mathbf{T}$ 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 ?

$$M(\vec{\theta}, \vec{\beta}; \mathbf{T}, \mathcal{S}, \mathcal{P}, \mathcal{W}, \mathcal{J})$$

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

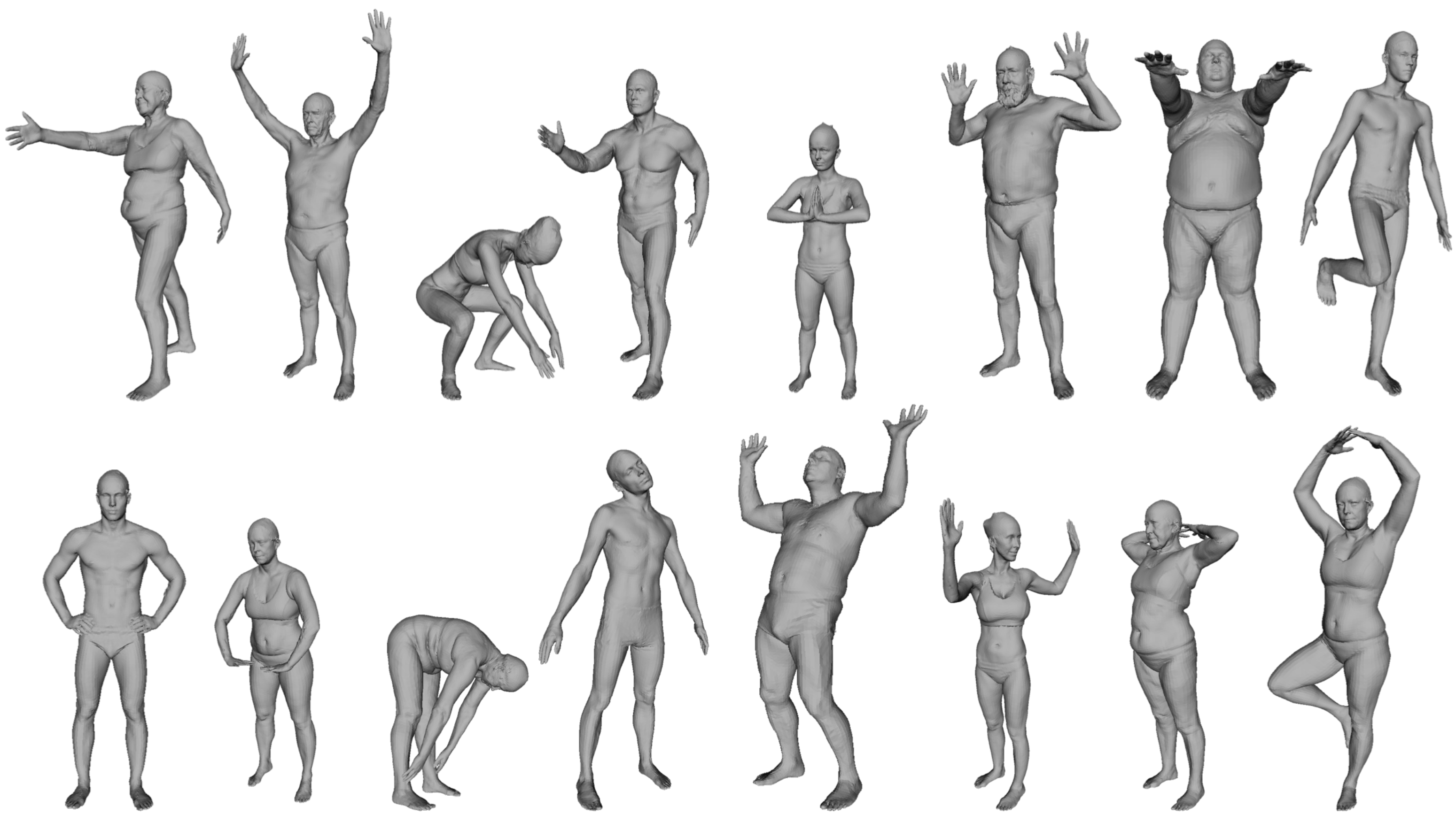
Input parameters

Hyper-parameters ?

DATA

Model Training

Multipose database: 20 males, 24 females
1800 registrations

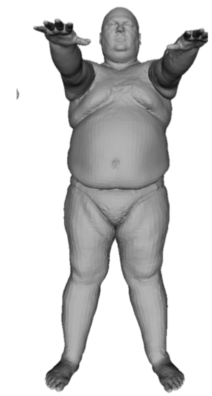


Model Training

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

Model Training

$$\mathbf{w} = \arg\min_{\mathbf{w}} \sum_{j} \| M(\vec{\theta}, \vec{\beta}; \mathbf{w}) - $$



$$\|^2$$

Training

$$\arg \min_{\mathbf{T},\mathcal{S},\mathcal{P},\mathcal{W},\mathcal{J}} \sum_j \min_{\vec{\theta}_j,\vec{\beta}_j} \| M(\vec{\theta}_j, \vec{\beta}_j; \mathbf{T}, \mathcal{S}, \mathcal{P}, \mathcal{W}, \mathcal{J}) - \mathbf{V}_j \|^2$$

Model

Registrations

Ideally one wants to find the model parameters that minimize a single objective measuring the distance between **model** and **registrations**

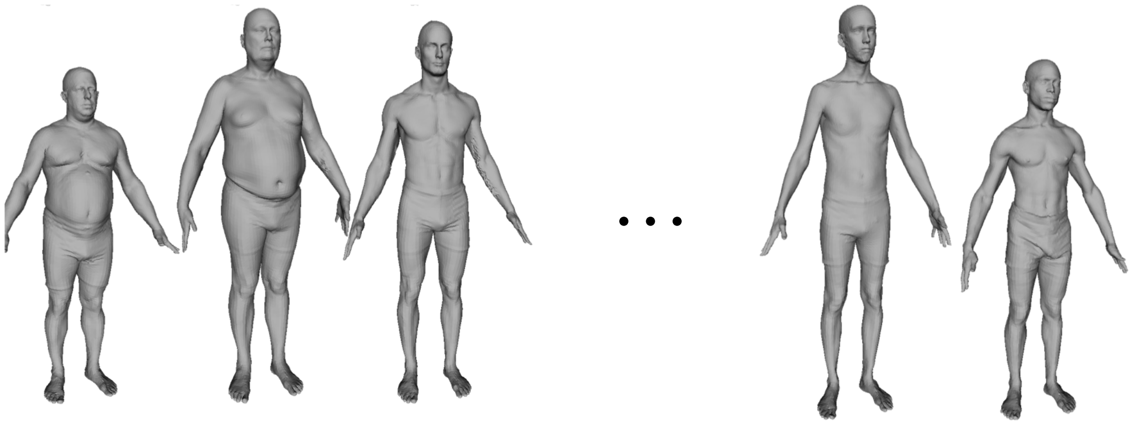
Gradient based optimization!

Number of Parameters Learned

For a model with 6890 vertices

- $\mathcal{P}$ 9x23x6890 = 4,278,690
- $\mathcal{W}$ 4x3x6890 = 82,680
- $\mathcal{J}$ 3x6890x23x3 = 1,426,230
- $\mathbf{T}, \mathcal{S}$ 3x6890 + 3x6890x10blendshapes = 227,370

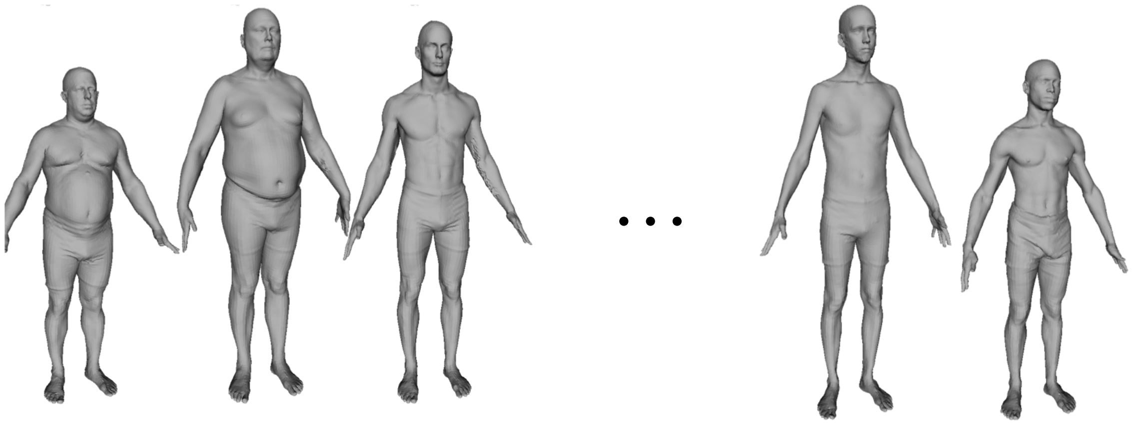
A total of 6.014.970 parameters are learned



$$\left[\mathbf{V}_1 \quad \mathbf{V}_2 \quad \ldots \quad \mathbf{V}_{N_{\mathrm{subj}}}\right] = \mathbf{T} + \left[\mathbf{S}_1 \quad \mathbf{S}_2 \quad \ldots \quad \mathbf{S}_{N_{\mathrm{subj}}}\right] \mathcal{B}$$

Average of shapes

Shape blend shapes are the first eigenvectors



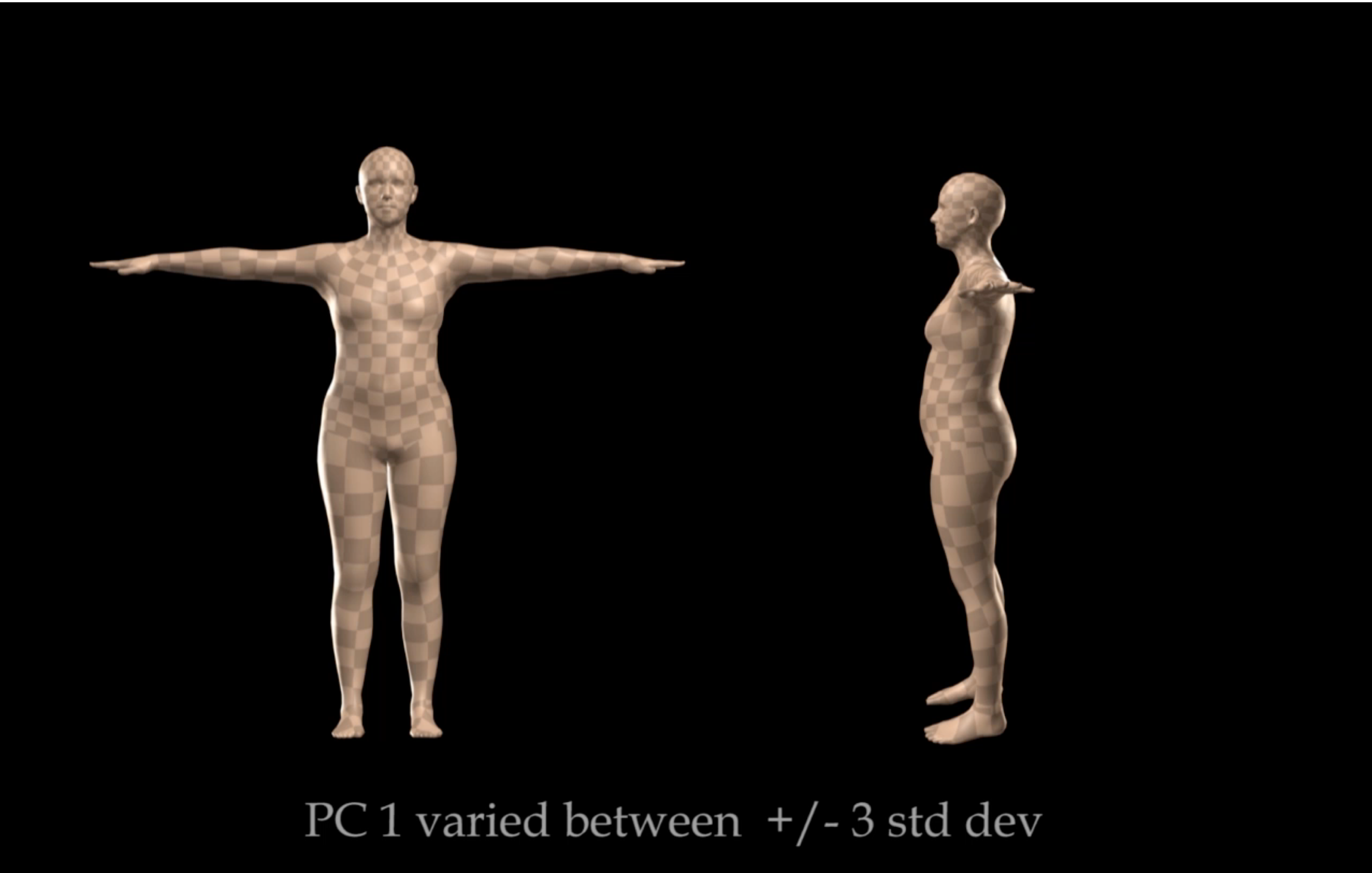
$$\begin{bmatrix}\mathbf{V}_1 & \mathbf{V}_2 & \dots & \mathbf{V}_{N_{\mathrm{subj}}}\end{bmatrix} \approx \mathbf{T} + \mathcal{S}\mathcal{B}$$

Average of shapes

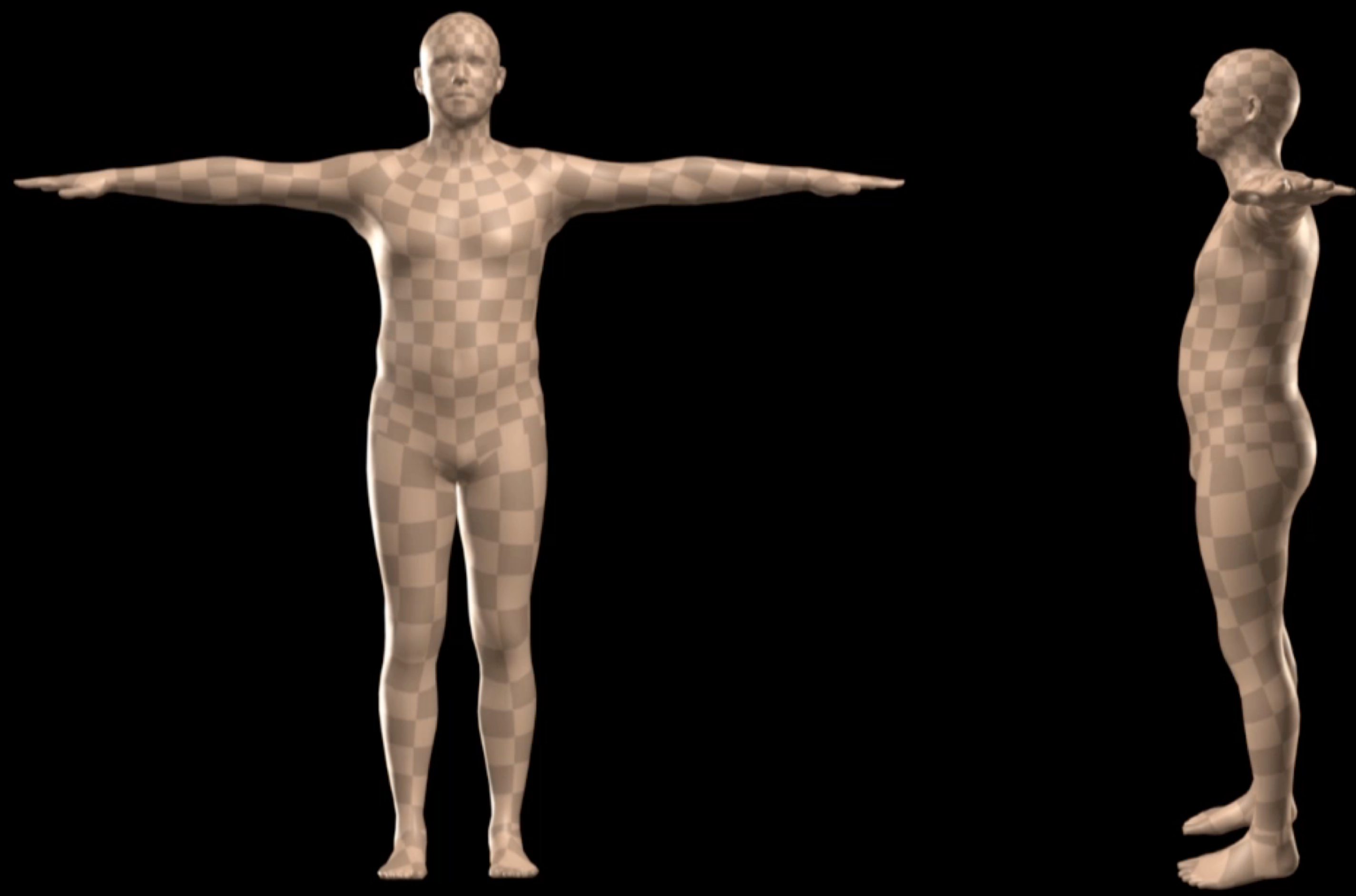
Shape blend shapes matrix

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

Shape Blend Shapes- Female

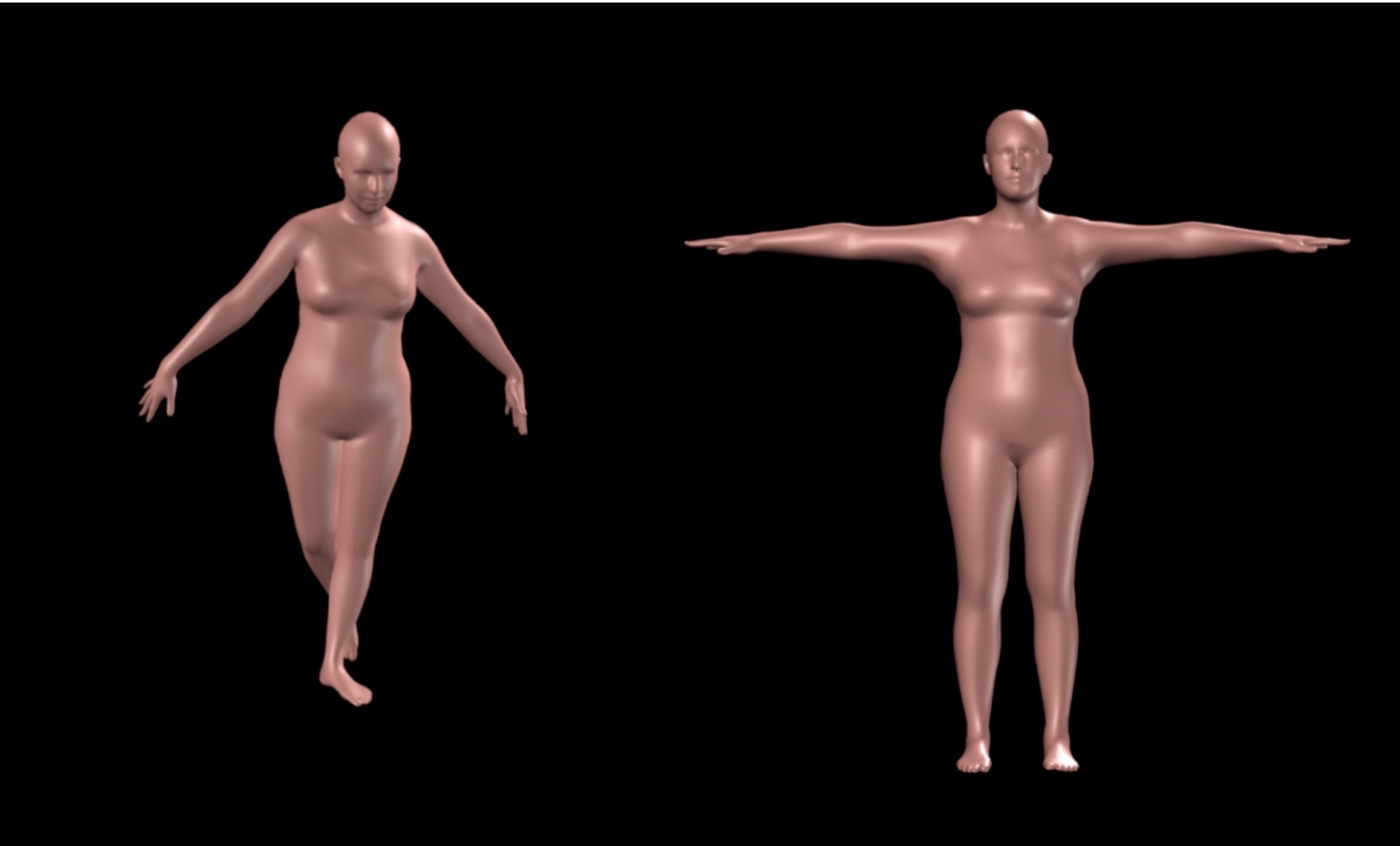


Shape Blend Shapes- Male



PC 1 varied between +/- 3 std dev

Pose Blendshapes

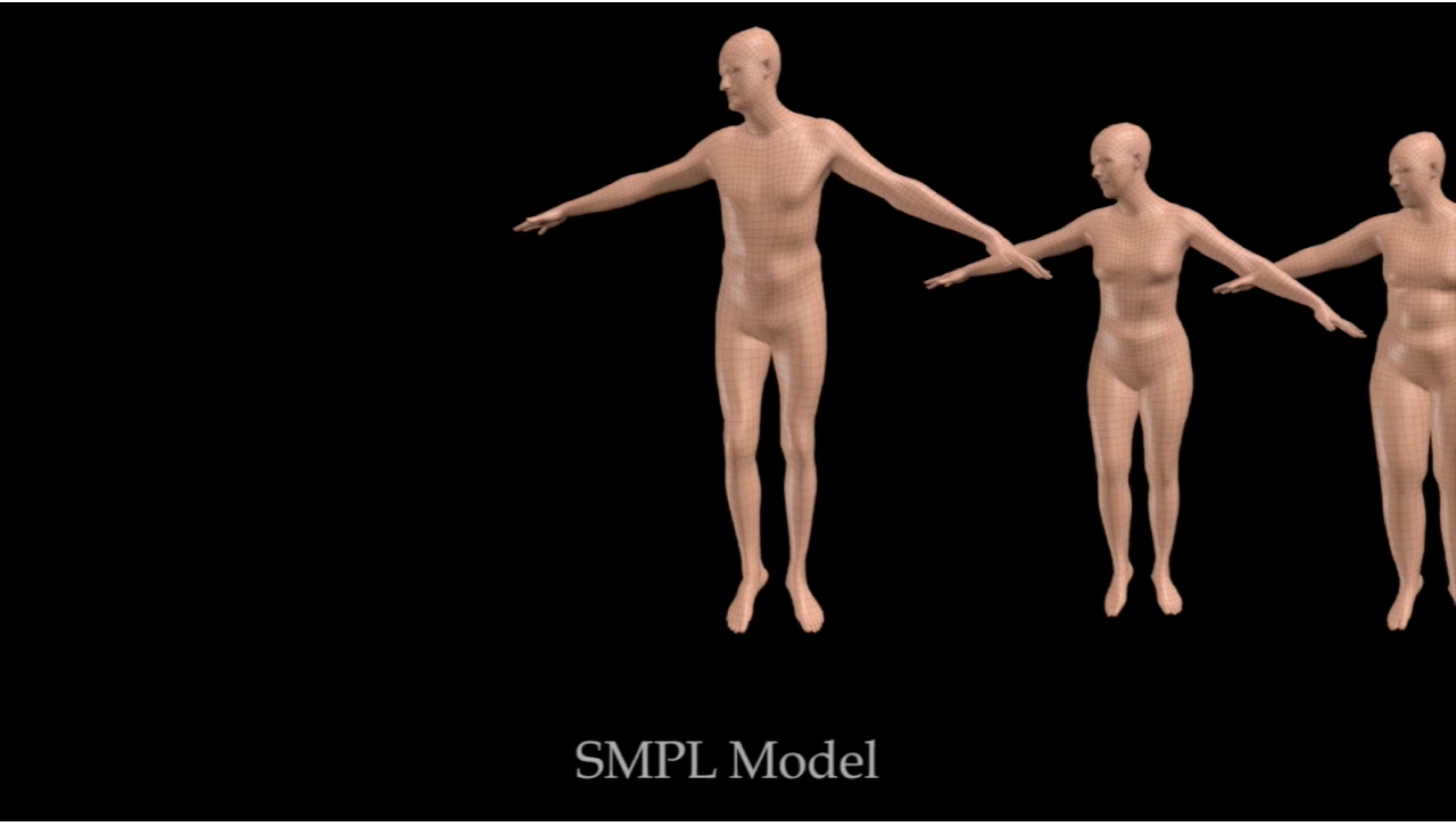


Conclusion

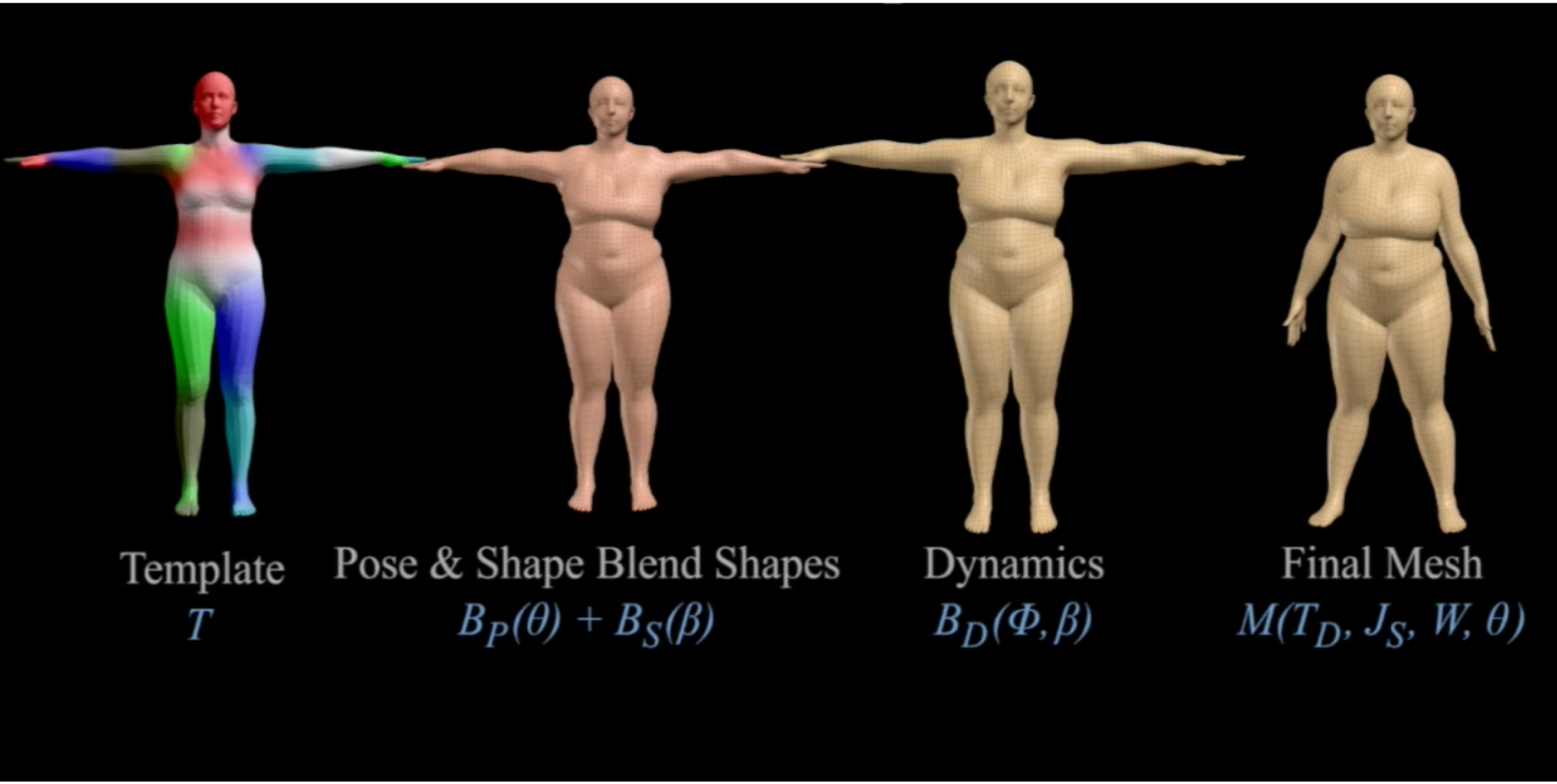
- **Speed**: fast run-time
- **Fidelity**: superior accuracy to Blend-SCAPE, trained on the same data
- **Compatibility**: works in Maya, other platforms soon
- Is publicly available for research purposes

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

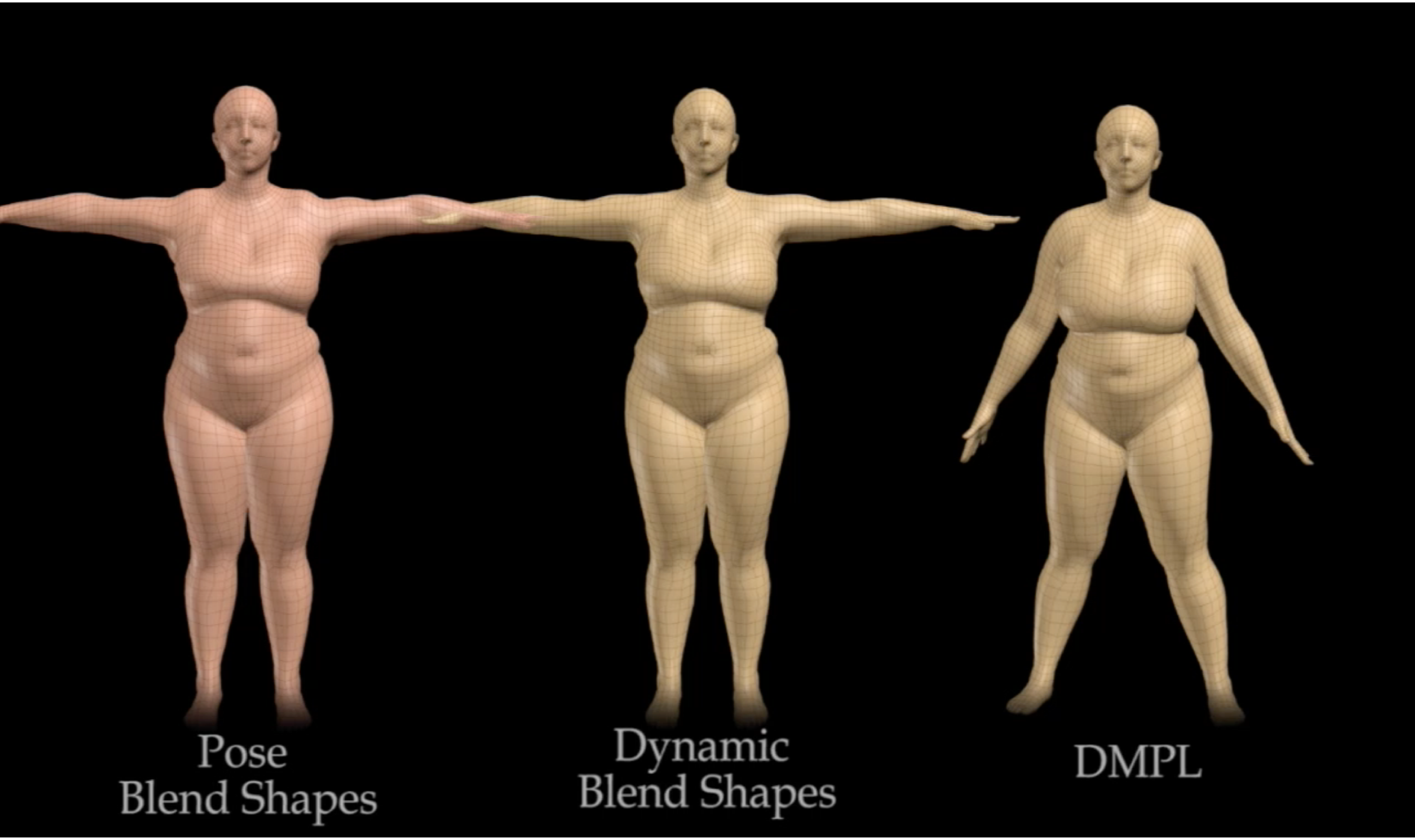
SMPL results



Model Decomposition



Dynamics of Soft Tissue



DMPL exaggeration

Applications 1

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

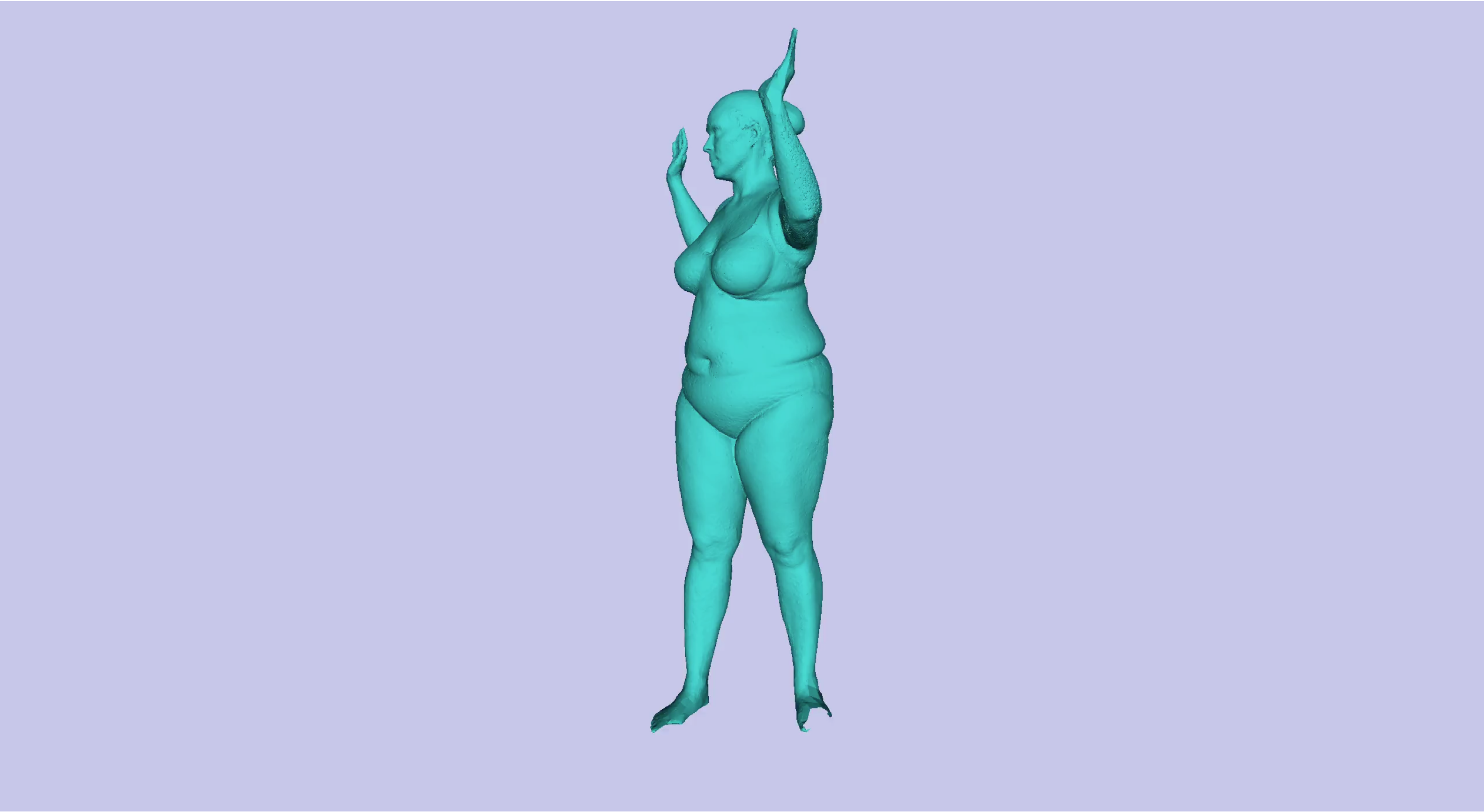
Fitting SMPL to a scan/mesh

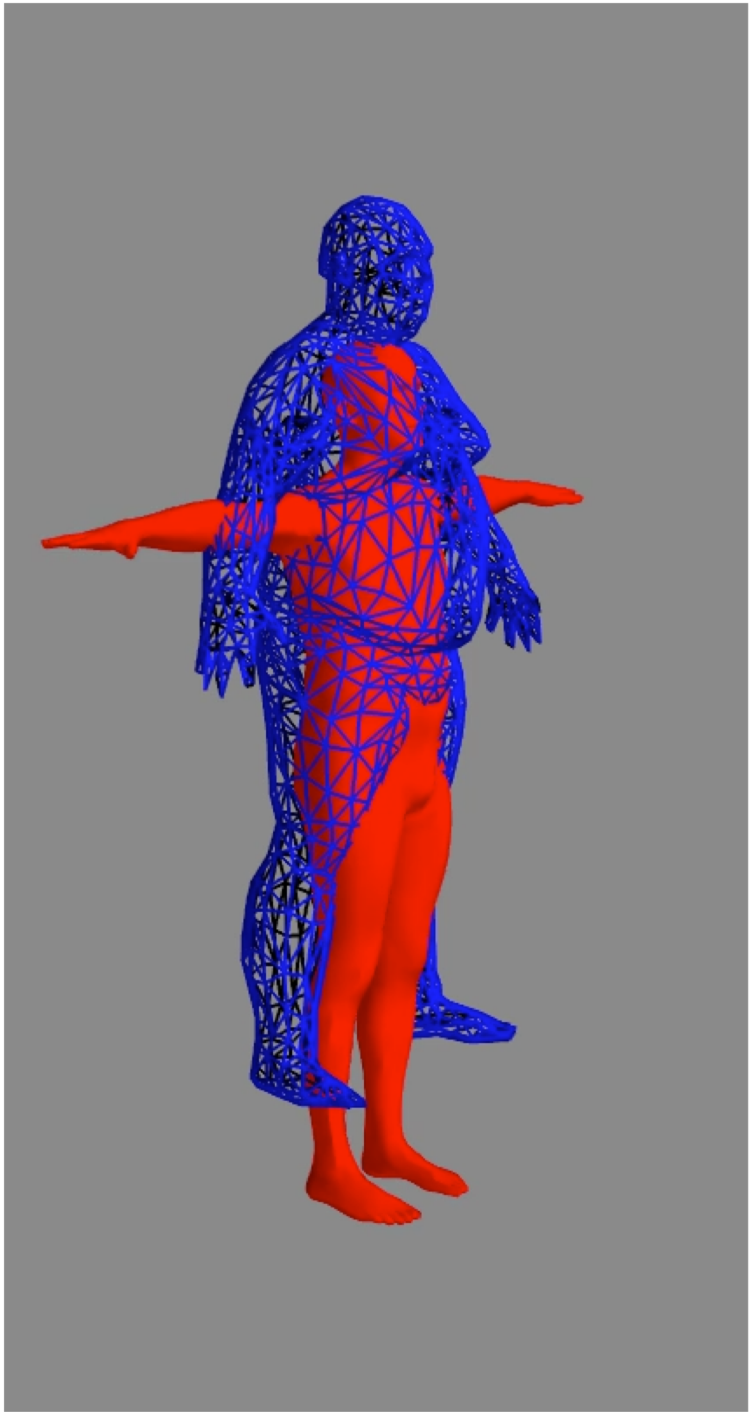
- Problem: Given a registration, find the model pose and shape.

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

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:

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

Gradient

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

Gradient of sum of squares

Jacobian of vector valued error function

Gauss Newton method

$$e(\mathbf{x}+\Delta\mathbf{x}) = \mathbf{e}(\mathbf{x}+\Delta\mathbf{x})^T\mathbf{e}(\mathbf{x}+\Delta\mathbf{x}) \simeq (e(\mathbf{x})+\mathbf{J}\Delta\mathbf{x}))^T(e(\mathbf{x})+\mathbf{J}\Delta\mathbf{x}))$$

$$\underbrace{\mathbf{J}^T\mathbf{J}}_{\text{Hessian Approximation}}\Delta\mathbf{x} = \underbrace{-\mathbf{J}^T\mathbf{e}}_{\text{Gradient of the sum of squares}}$$

Levenberg-Marquadt method

$$e(\mathbf{x}+\Delta\mathbf{x}) = \mathbf{e}(\mathbf{x}+\Delta\mathbf{x})^T\mathbf{e}(\mathbf{x}+\Delta\mathbf{x}) \simeq (e(\mathbf{x})+\mathbf{J}\Delta\mathbf{x}))^T(e(\mathbf{x})+\mathbf{J}\Delta\mathbf{x}))$$

$$(\mathbf{J}^T\mathbf{J}+\lambda\mathbf{I})\Delta\mathbf{x} = -\mathbf{J}^T\mathbf{e}$$

Hessian Approximation

Gradient of the sum of squares

When do we need to compute 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 requires computing Jacobians

If optimization takes too long, or breaks etc.

Ask yourself the following:

- What is the dimension of the Jacobian?
- Is it dense? (sparsity is exploited for speed).
- Is the 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