Body Models IV

Javier Romero
Max Planck Institute for Intelligent Systems
Perceiving Systems

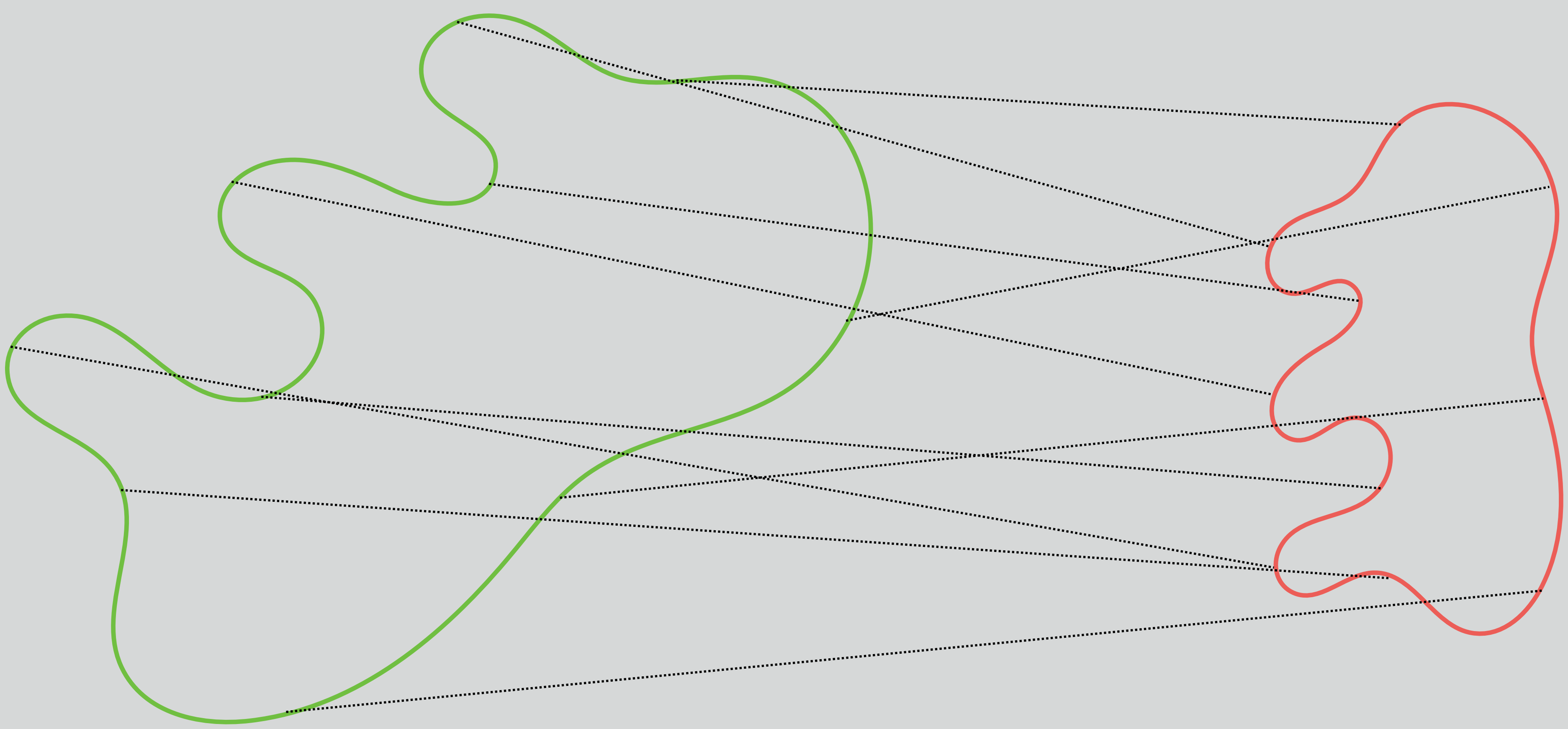
June 13, 2016

MAX-PLANCK-GESELLSCHAFT

11.04.2016	Introduction
18.04.2016	Graphical Models 1
25.04.2016	Graphical Models 2 (Sand 6/7)
02.05.2016	Graphical Models 3
09.05.2016	Graphical Models 4
23.05.2016	Body Models 1
30.05.2016	Body Models 2
06.06.2016	Body Models 3
13.06.2016	**Body Models 4**
20.06.2016	Stereo
27.06.2016	Optical Flow
04.07.2016	Segmentation
11.07.2016	Object Detection 1
18.07.2016	Object Detection 2

What have we learned so far about bodies?

- BM1: Procrustes for rigid alignment

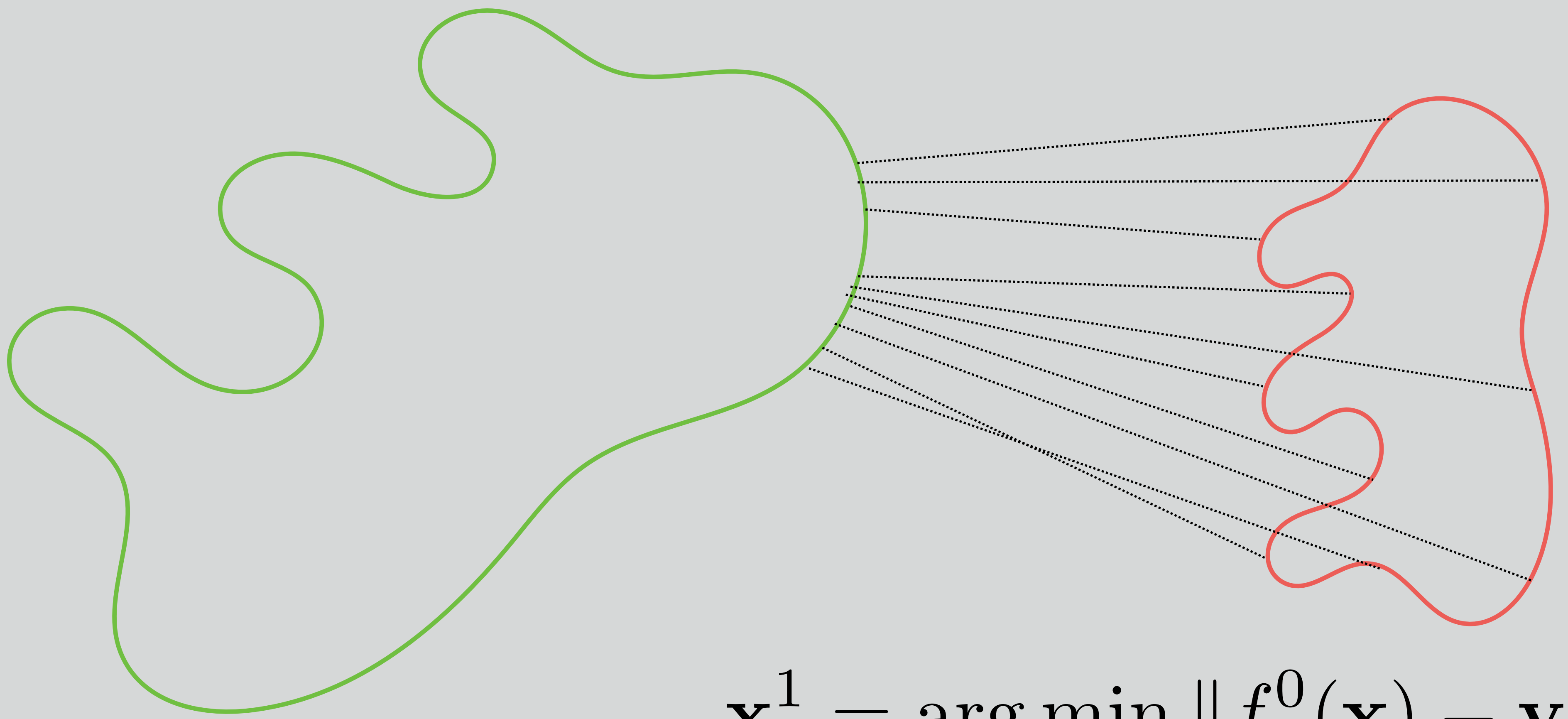


solve with procrustes
single step

$$f = \arg\min_{f} \sum_{i} \| f(\mathbf{x}_i) - \mathbf{y}_i \|^2$$

What have we learned so far about bodies?

- BM1: Procrustes for rigid alignment
- BM2: ICP, gradient-based ICP

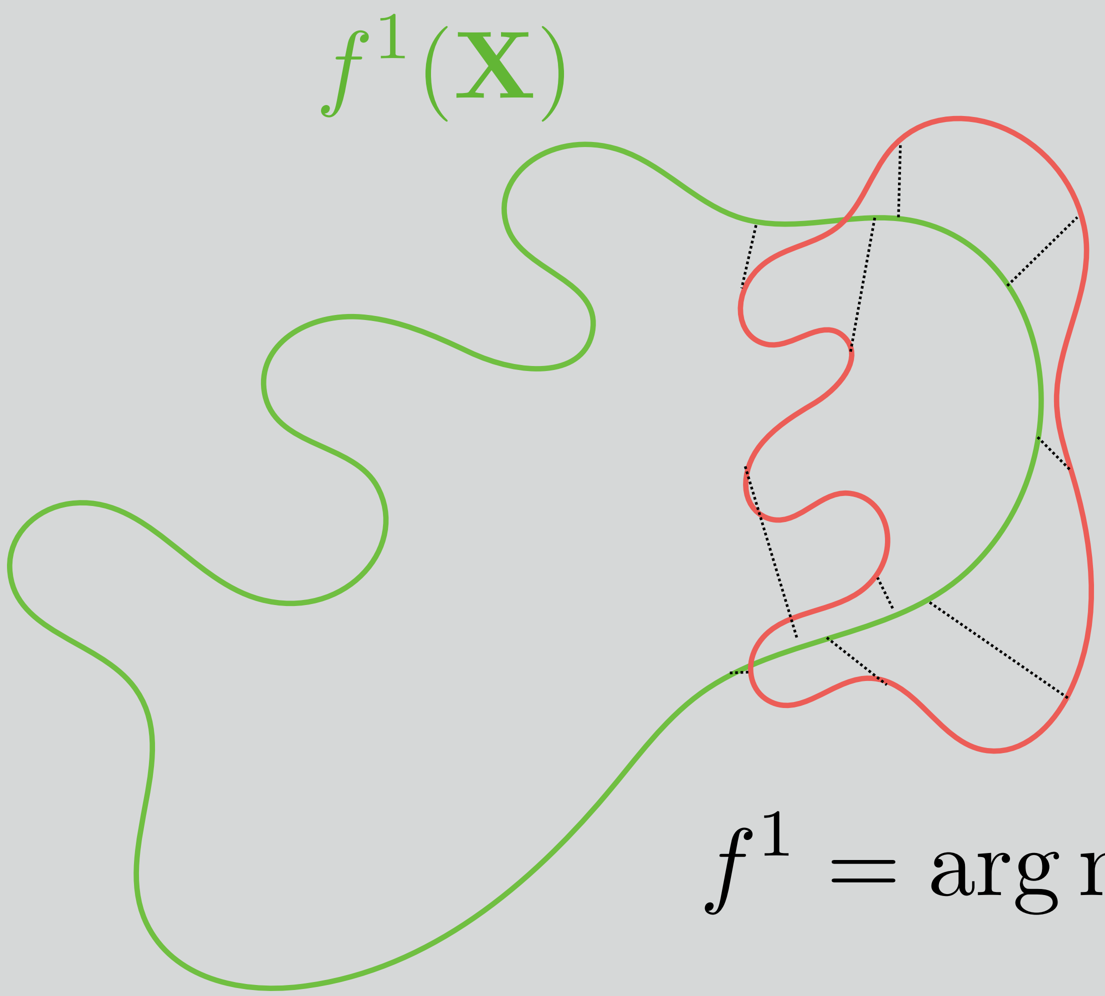


$$\mathbf{x}_i^1 = \arg\min_{\mathbf{x}\in\mathbf{X}} \|f^0(\mathbf{x}) - \mathbf{y}_i\|^2$$

solve with procrustes
or gradient-based

$$f^1 = \arg\min_f \sum_i \|f(\mathbf{x}_i^1) - \mathbf{y}_i\|^2$$

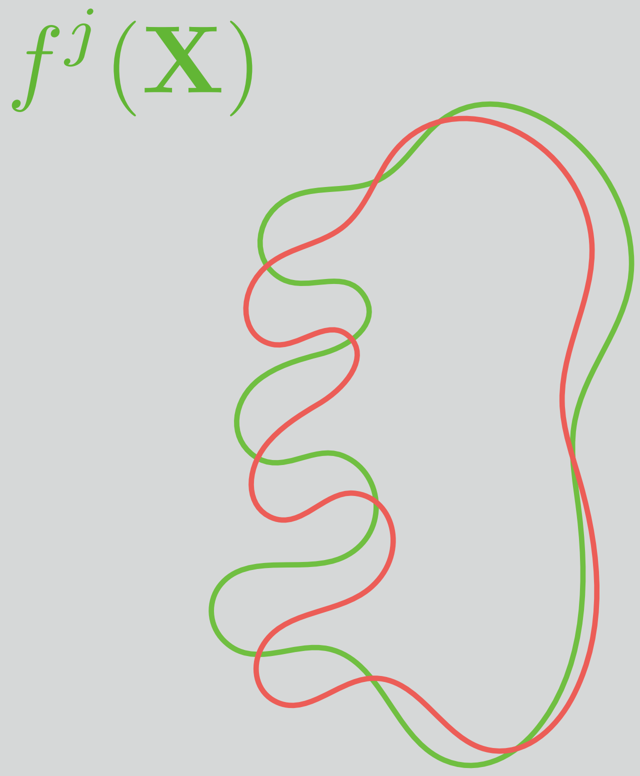
and iterate!



$$f^1 = \arg\min_f \sum_i \|f(\mathbf{x}_i^1) - \mathbf{y}_i\|^2$$

$$\mathbf{x}_i^2 = \arg\min_{\mathbf{x}\in\mathbf{X}} \|f^1(\mathbf{x}) - \mathbf{y}_i\|^2$$

and iterate!



$$f^j = \arg\min_f \sum_i \|f(\mathbf{x}_i^j) - \mathbf{y}_i\|^2$$

$$\mathbf{x}_i^{j+1} = \arg\min_{\mathbf{x}\in\mathbf{X}} \|f^j(\mathbf{x}) - \mathbf{y}_i\|^2$$

and iterate!

$$f^j = \arg\min_f \sum_i \| f(\mathbf{x}_i^j) - \mathbf{y}_i \|^2$$

$$\mathbf{x}_i^{j+1} = \arg\min_{\mathbf{x} \in \mathbf{X}} \| f^j(\mathbf{x}) - \mathbf{y}_i \|^2$$

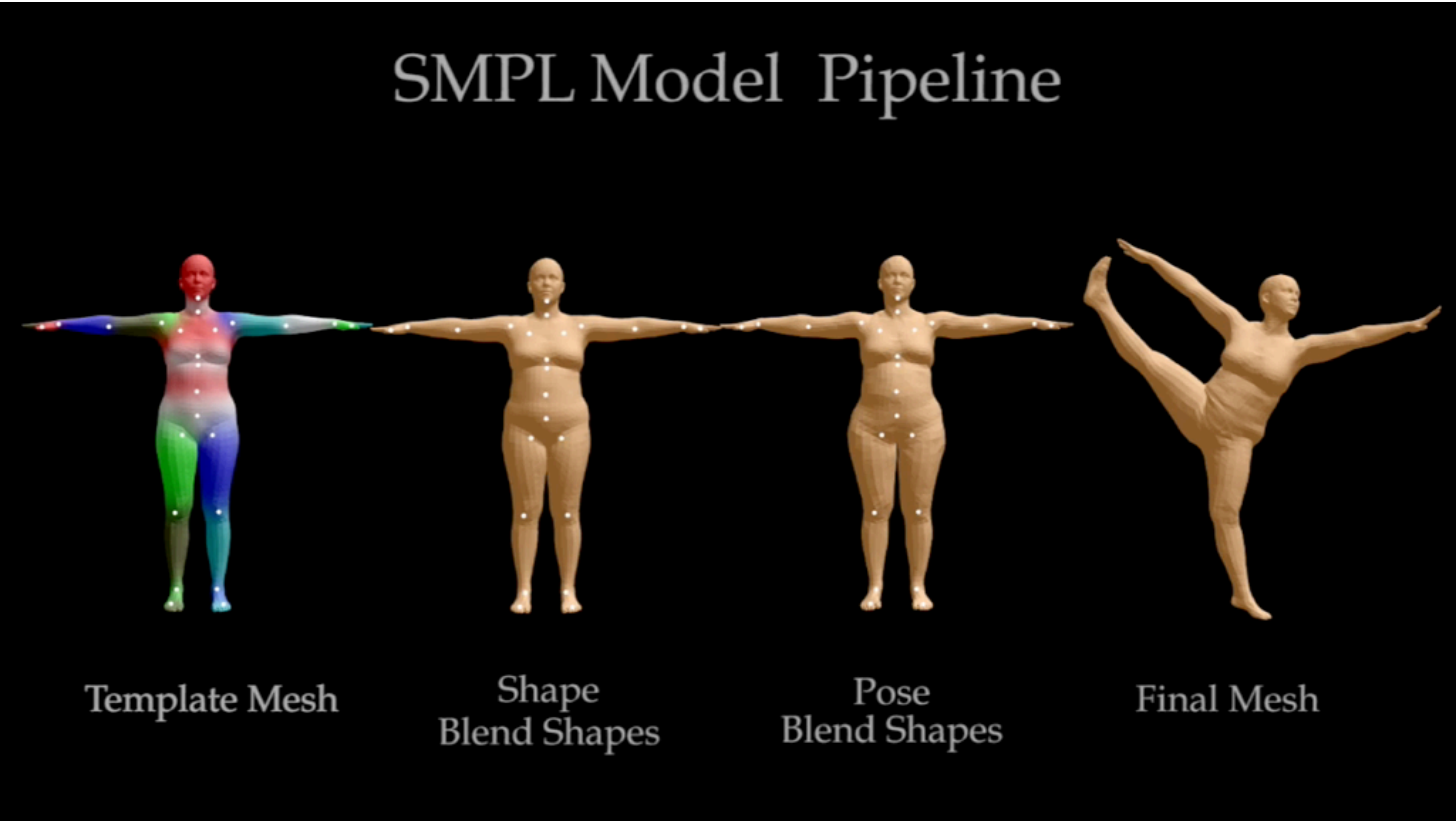
and iterate!

$$f^j = \arg\min_f \sum_i \|f(\mathbf{x}_i^j) - \mathbf{y}_i\|^2$$

$$\mathbf{x}_i^{j+1} = \arg\min_{\mathbf{x}\in\mathbf{X}} \|f^j(\mathbf{x}) - \mathbf{y}_i\|^2$$

What have we learned so far about bodies?

- BM1: Procrustes for rigid alignment
- BM2: ICP, gradient-based ICP
- BM3: Articulated models, Blendshapes, SMPL


SMPL Model Pipeline
Template Mesh
Shape
Blend Shapes
Pose
Blend Shapes
Final Mesh

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 $\vec{\beta}$

What is missing: today

- How do we fit SMPL to meshes without correspondences?
- This is a computer vision course.
  Where is the color in those meshes?
- Autodiff in images? OpenDR
- Fitting bodies to images

Fitting SMPL to a scan/mesh

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

some distance function between the two meshes

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

Model

Scan

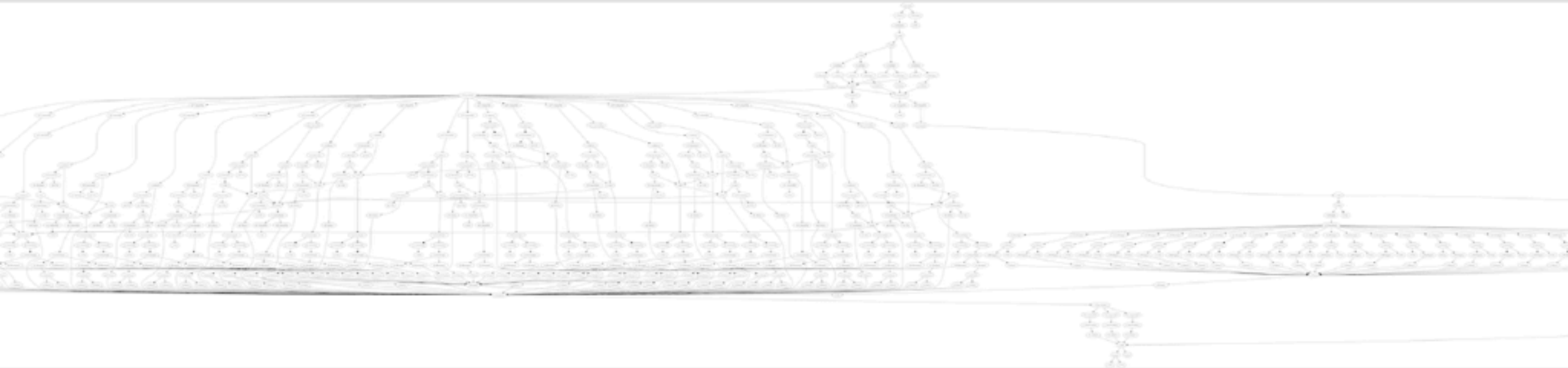
Fitting SMPL to a scan/mesh

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

```
from smpl.serialization import load_model
sm = load_model(path_to_downloaded_model)
ch.minimize(point2point_squared(dst_pts=sm, org_pts=Xch),
x0=[sm.betas, sm.pose])
```

Model Scan

SMPL tree: `sm.show_tree()`



- 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)

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

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

```
ipdb> p2p_yx = point2point_squared(org_pts=Xch, dst_pts=sm)
ipdb> print(p2p_yx)
[ 0.001  0.     0.001 ...,  0.012  0.012  0.012]
ipdb> p2p_yx.shape
(6890,)
```

as many elements as correspences between model and scan

Jacobian of the vector valued error function:

$$J_{\mathbf{e}}(\mathbf{x}) = \frac{d\mathbf{e}(\mathbf{x})}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{e}_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{e}_1}{\partial \mathbf{x}_P} \\ & \ddots & \\ \frac{\partial \mathbf{e}_N}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{e}_N}{\partial \mathbf{x}_P} \end{bmatrix}$$

N residuals

P parameters

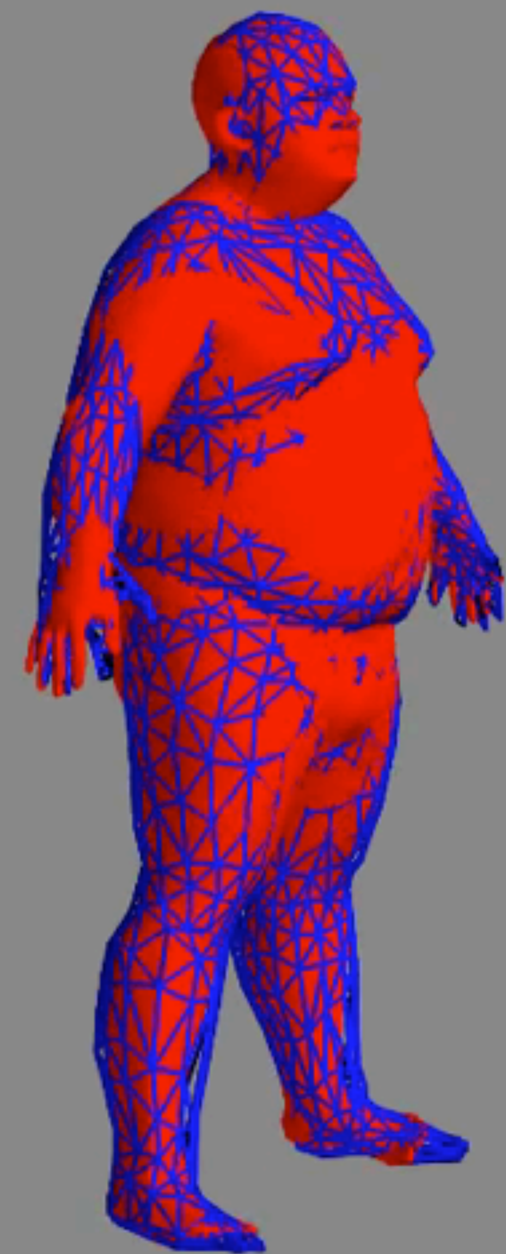
$$J_{\mathbf{e}}(\mathbf{x}) = \frac{d\mathbf{e}(\mathbf{x})}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{e}_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{e}_1}{\partial \mathbf{x}_P} \\ & \ddots & \\ \frac{\partial \mathbf{e}_N}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{e}_N}{\partial \mathbf{x}_P} \end{bmatrix}$$

N residuals

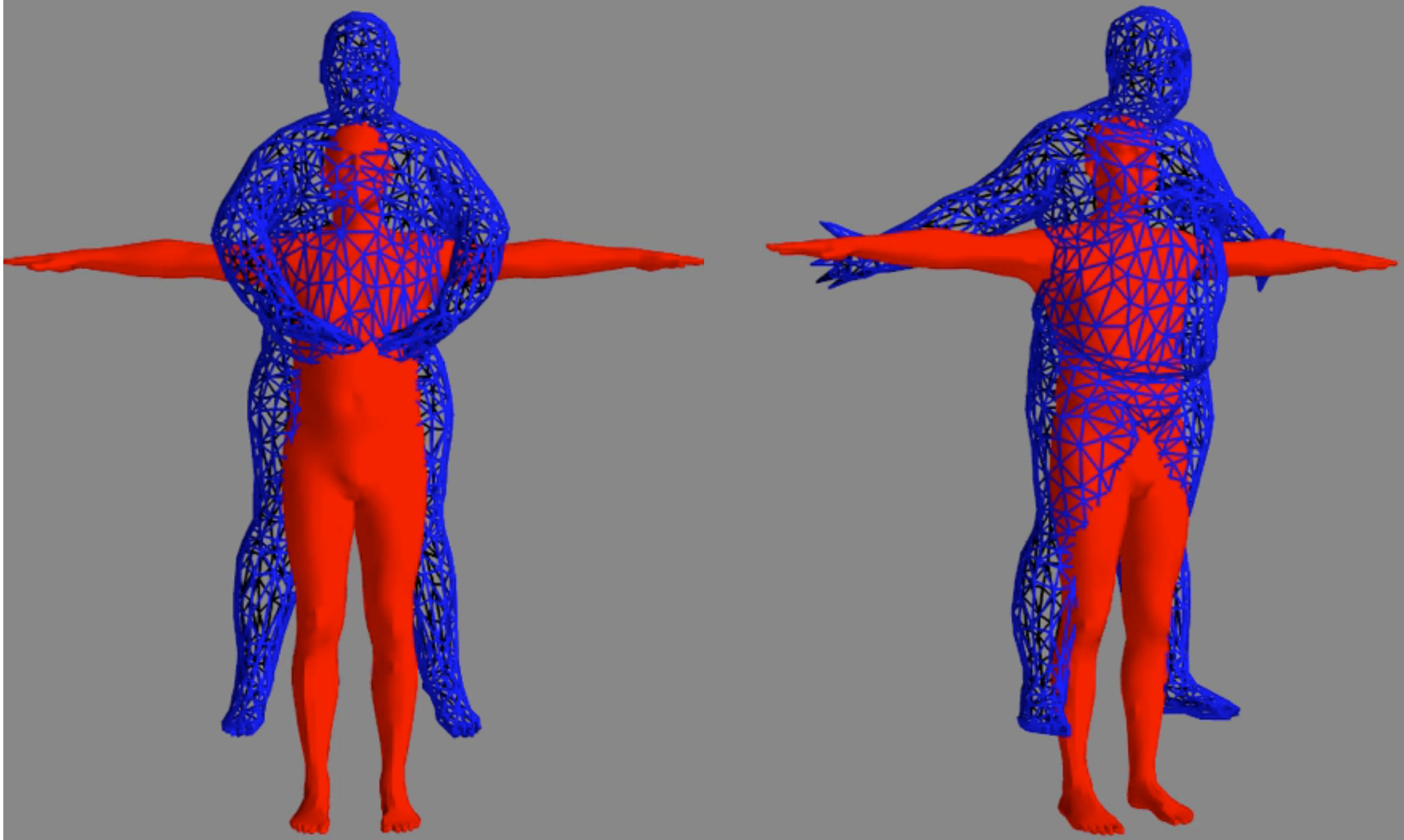
P parameters

```
ipdb> print(p2p_yx.dr_wrt(sm.betas).shape)
(6890, 10)
ipdb> print(p2p_yx.dr_wrt(sm.betas)[:5, :5].todense())
[[ -1.144e-04  -1.148e-04   3.350e-05  -2.048e-05   8.550e-06]
 [  3.490e-04  -4.617e-05  -1.243e-04  -7.371e-05   3.262e-05]
 [  5.642e-04  -1.518e-04  -2.017e-04  -1.487e-04   9.339e-05]
 [  2.437e-04  -2.448e-04  -9.368e-05  -1.272e-04   9.360e-05]
 [  8.284e-04  -1.090e-04  -2.925e-04  -1.700e-04   9.579e-05]]
```

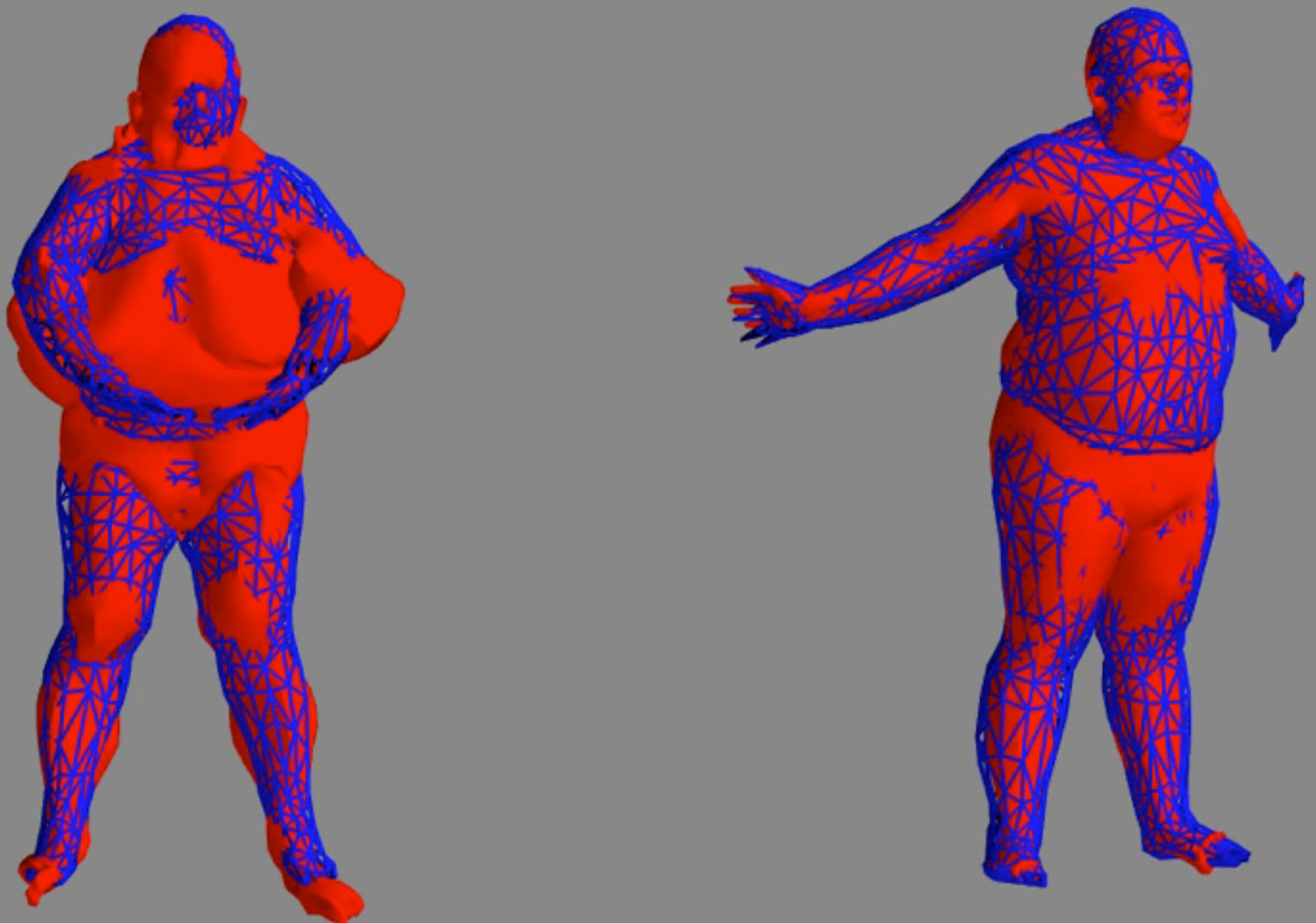
Try it!



Which one will fail?

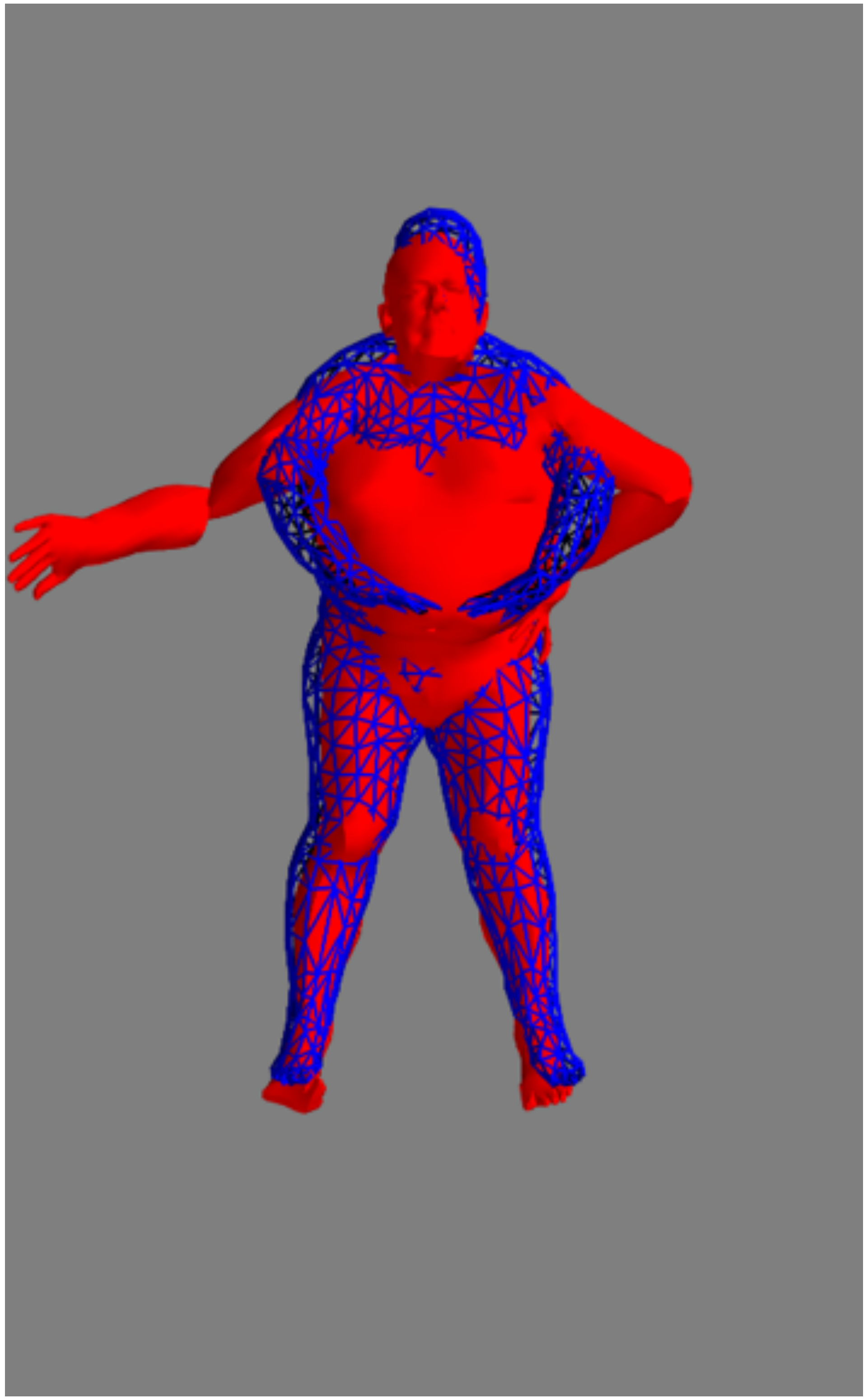


Which one will fail?



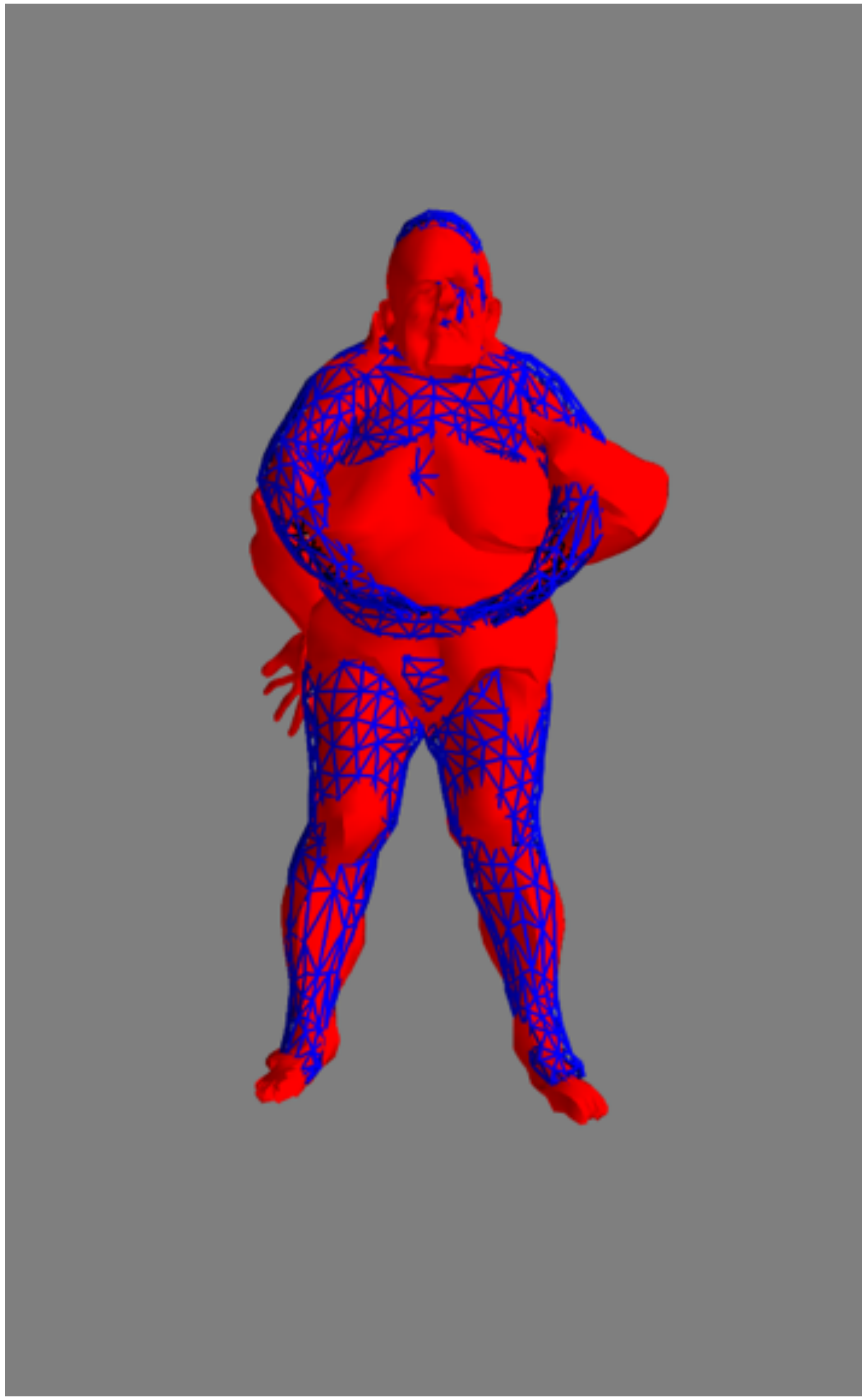
Problems?

- Unlikely pose



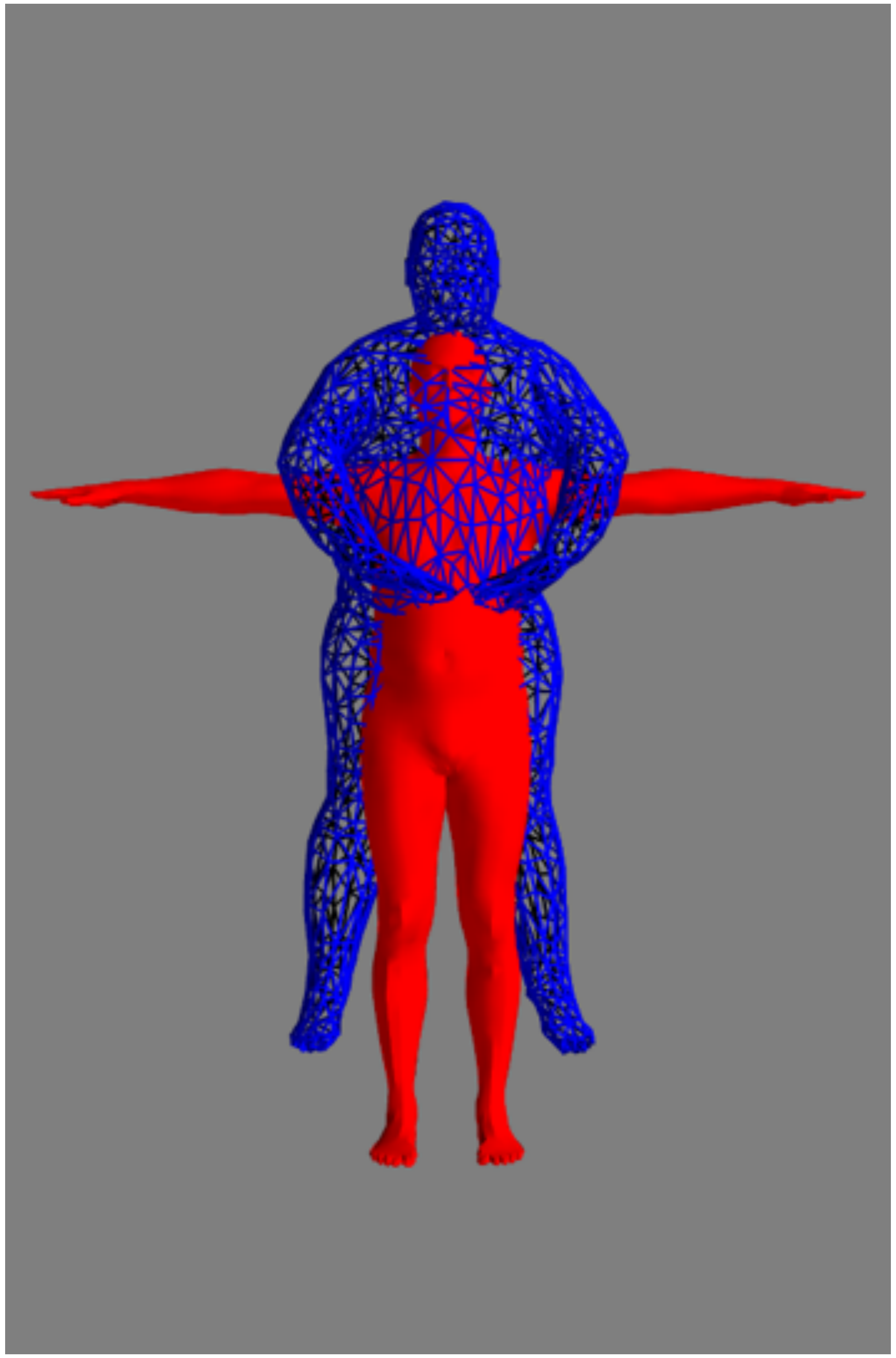
Problems?

- Unlikely pose
- Unlikely shape



Problems?

- Unlikely pose
- Unlikely shape
- Bad initialization



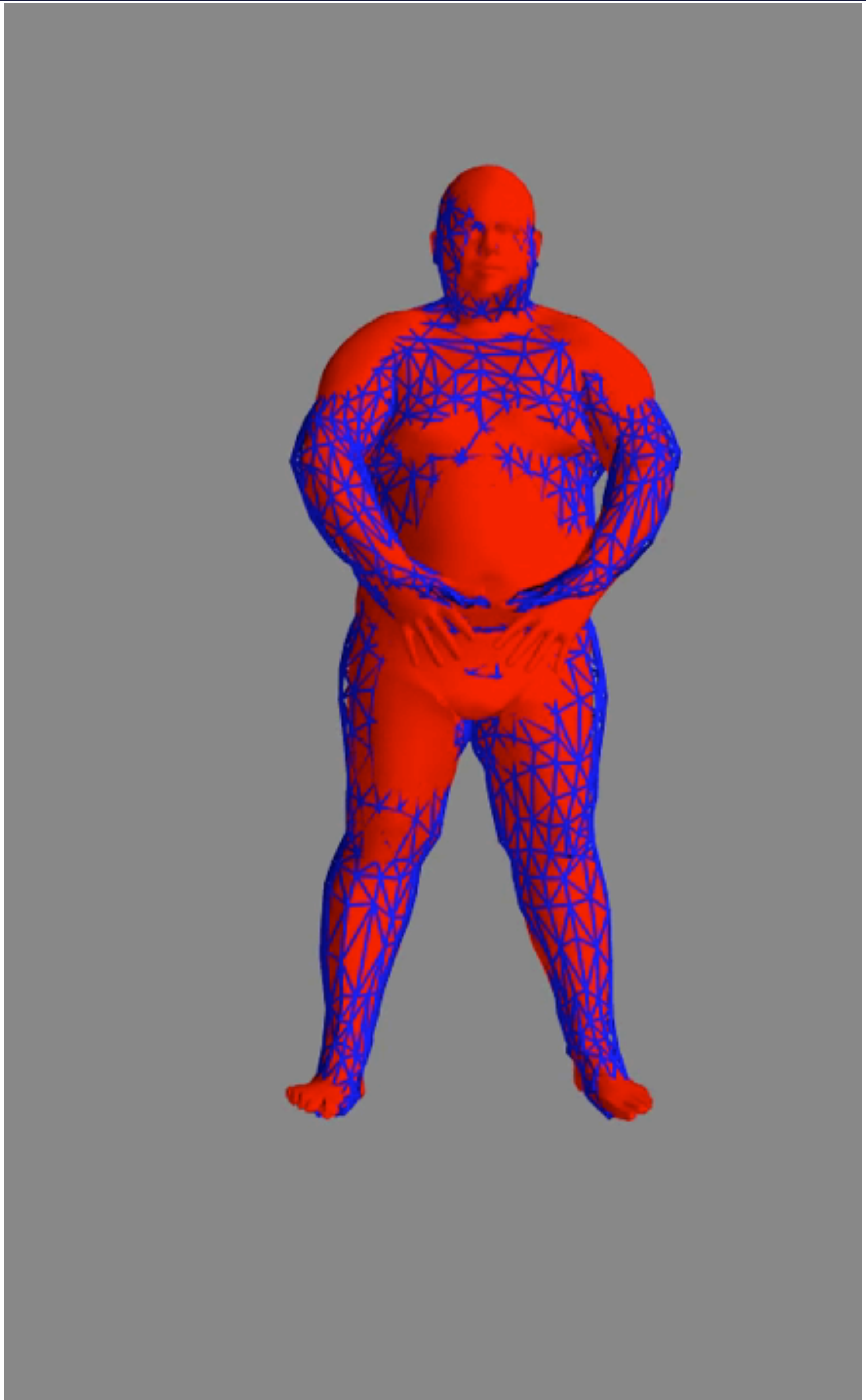
Fitting SMPL to a scan/mesh

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

Fitting SMPL to a scan/mesh

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

$$+ E_{\theta}(\vec{\theta})$$

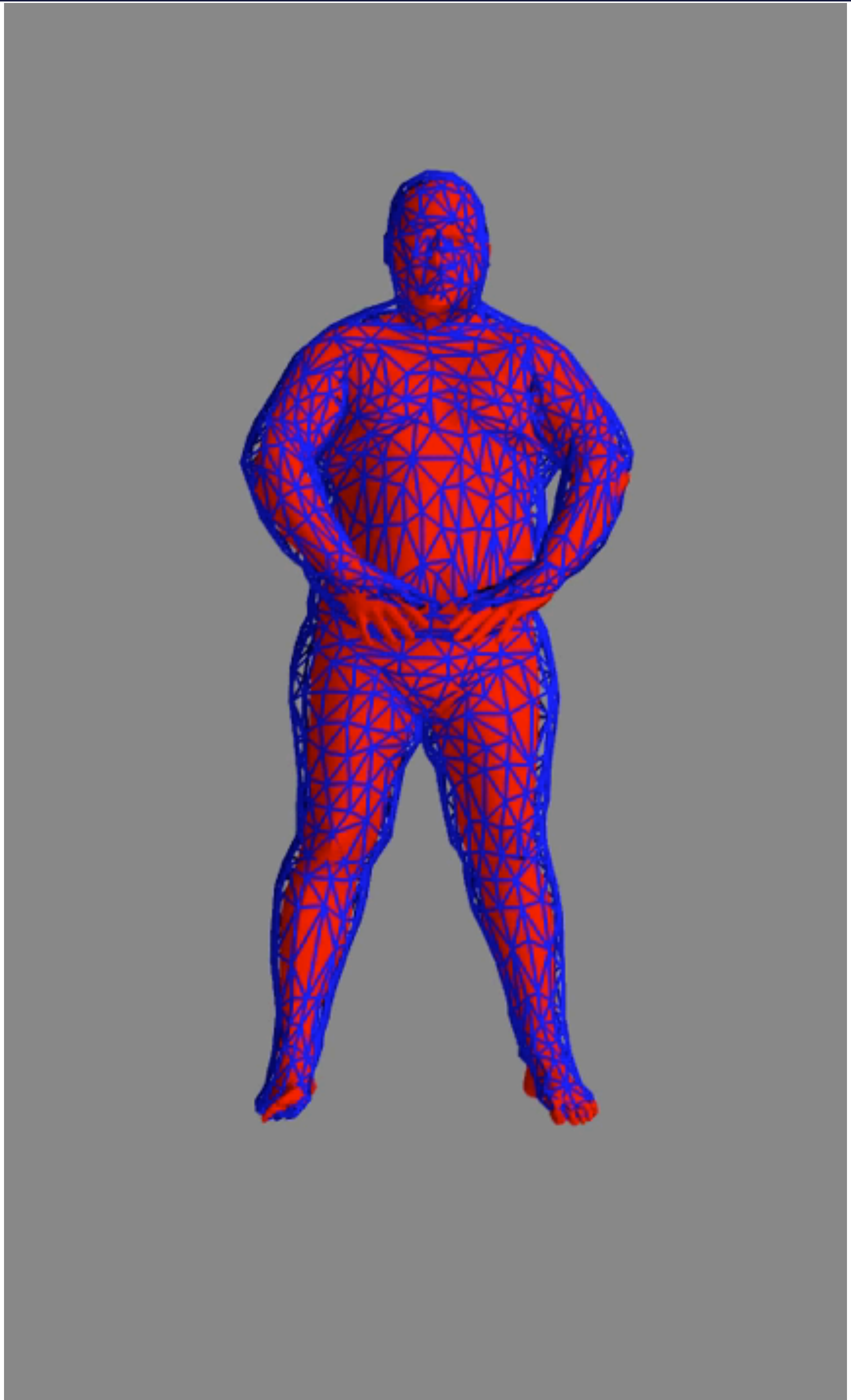


Fitting SMPL to a scan/mesh

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

$$+ E_\theta(\vec{\theta})$$

$$+ E_\beta(\vec{\beta})$$



Fitting SMPL to a scan/mesh

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

$$+ E_\theta(\vec{\theta})$$

$$+ E_\beta(\vec{\beta})$$

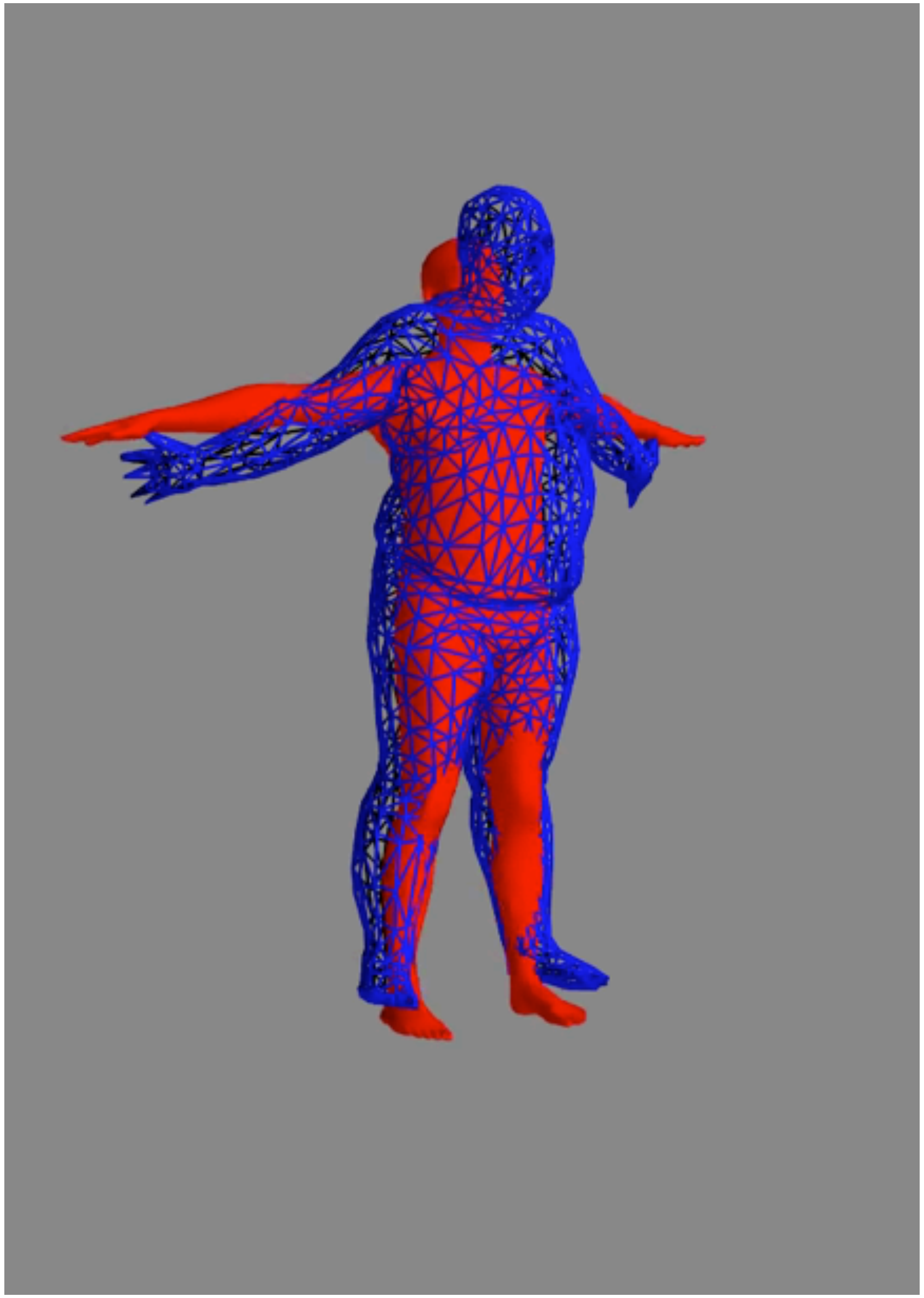
Mahalanobis distance
induced by distribution $\mathcal{N}(\vec{\mu}_\theta, \Sigma_\theta)$

$$E_\theta(\vec{\theta}) \equiv (\vec{\theta} - \vec{\mu_\theta})^T \Sigma_\theta^{-1} (\vec{\theta} - \vec{\mu_\theta})$$

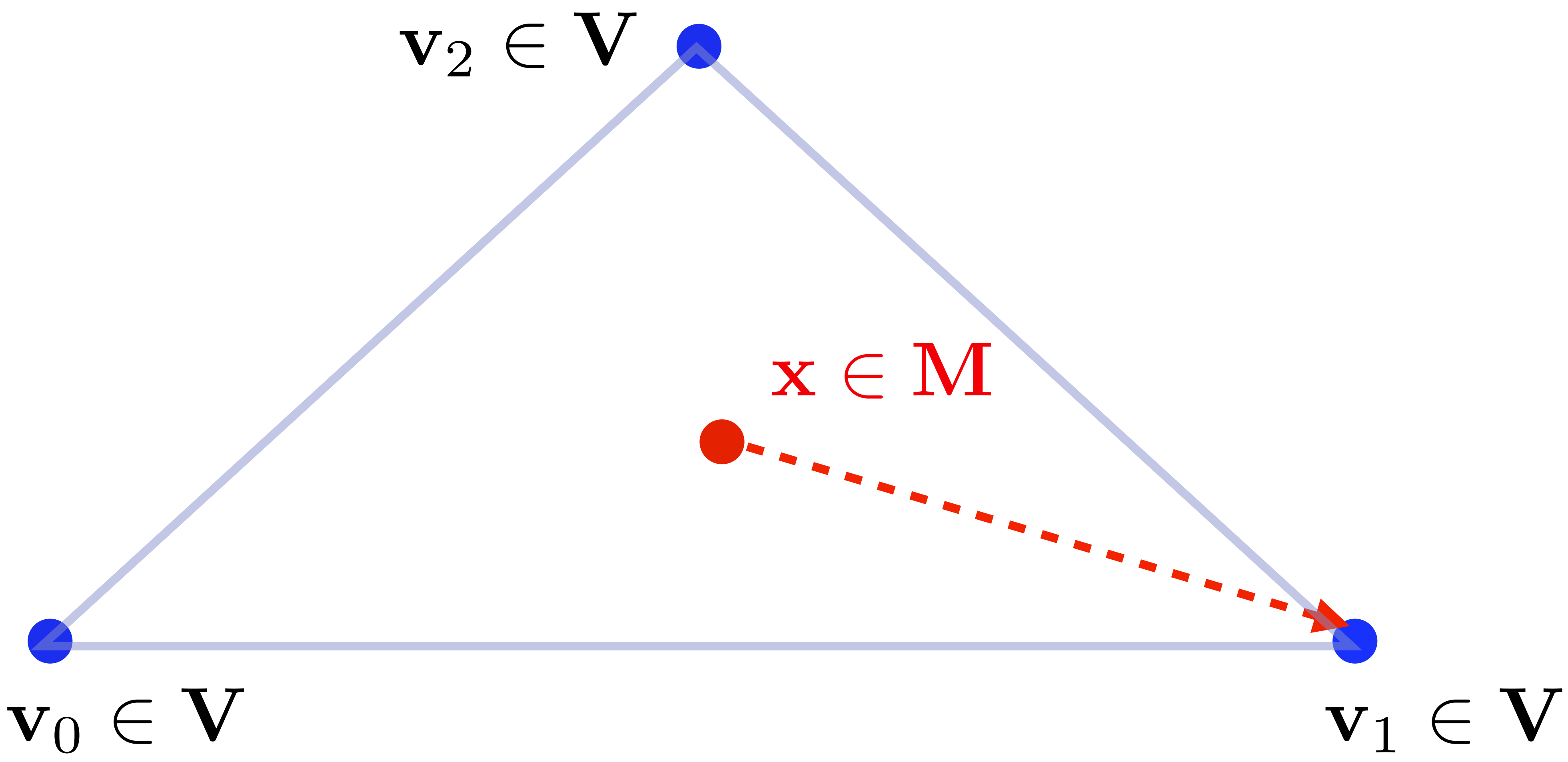
$$E_\beta(\vec{\beta}) \equiv (\vec{\beta} - \vec{\mu_\beta})^T \Sigma_\beta^{-1} (\vec{\beta} - \vec{\mu_\beta})$$

Fitting SMPL to a scan/mesh

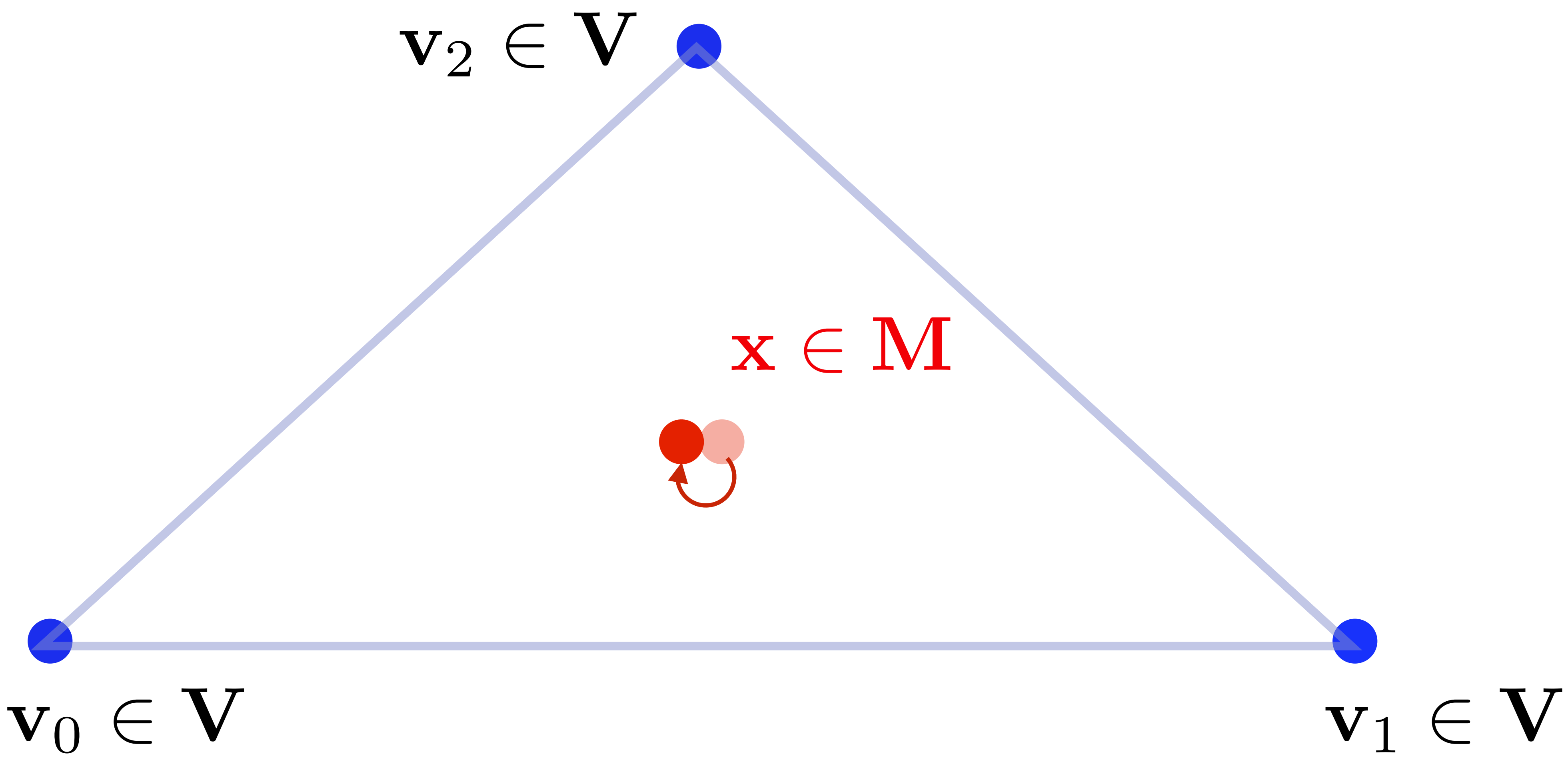
- What makes it so jumpy?
    - Correspondences change abruptly!



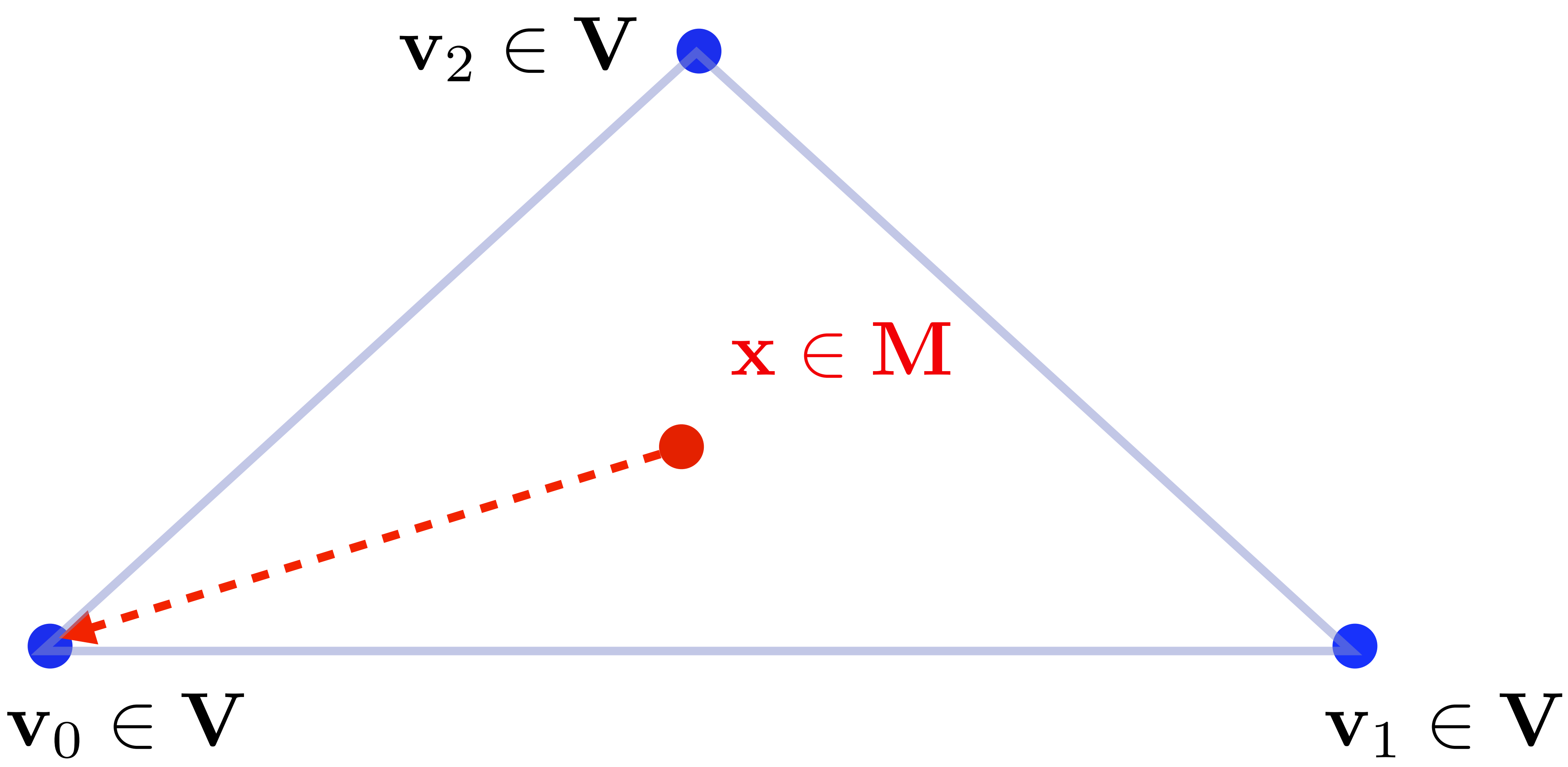
Point-to-point distance



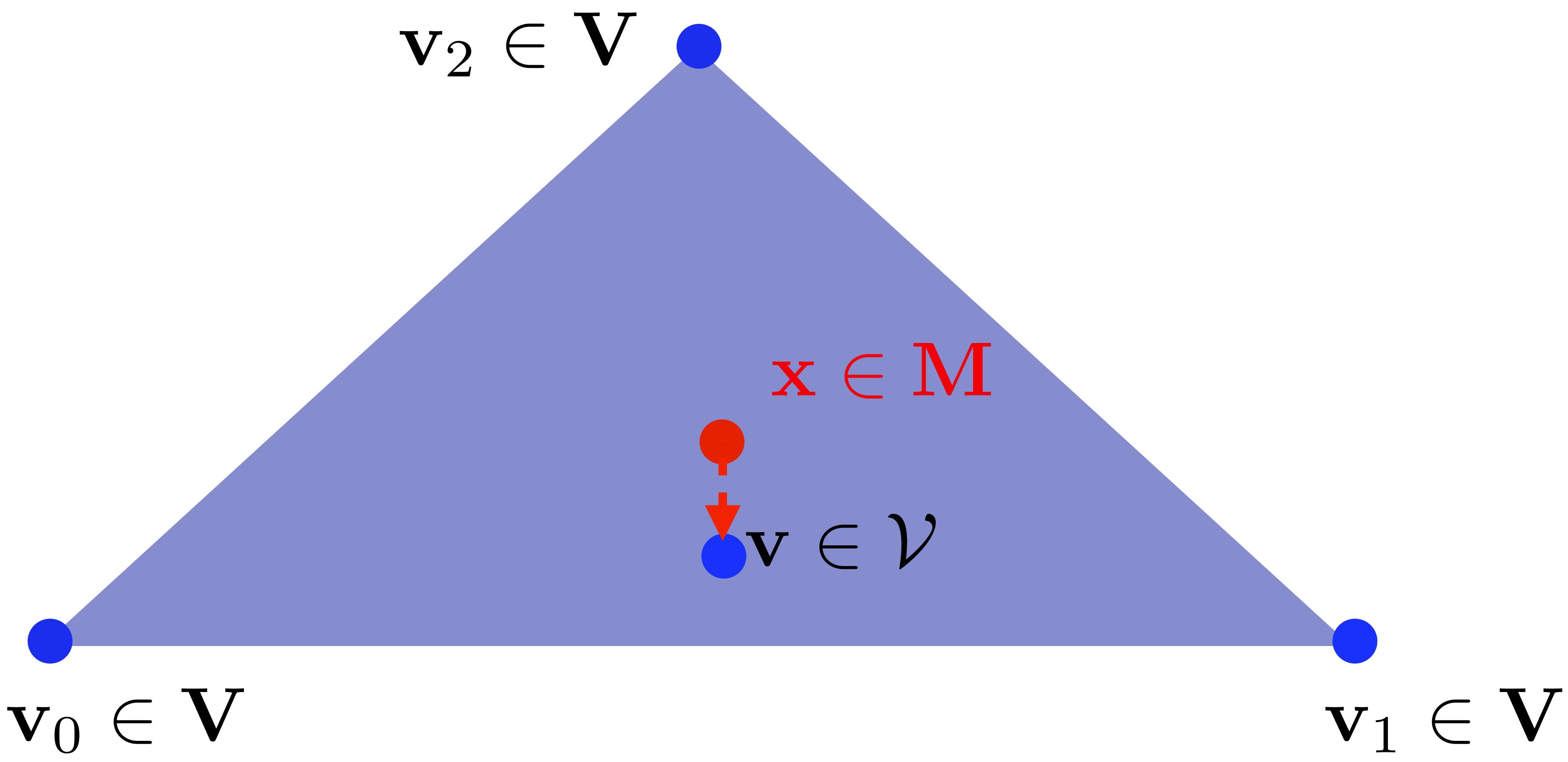
Point-to-point distance



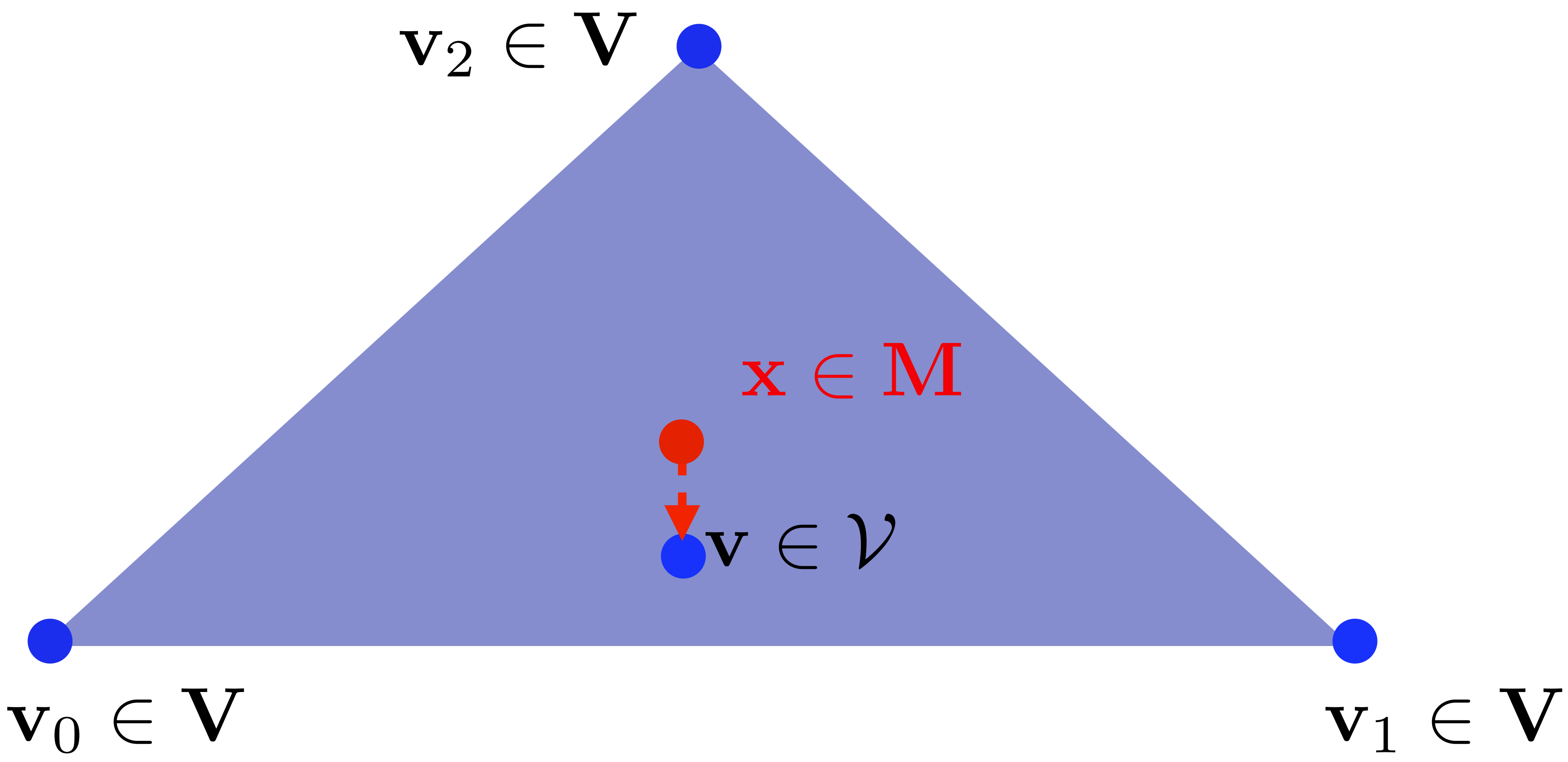
Point-to-point distance



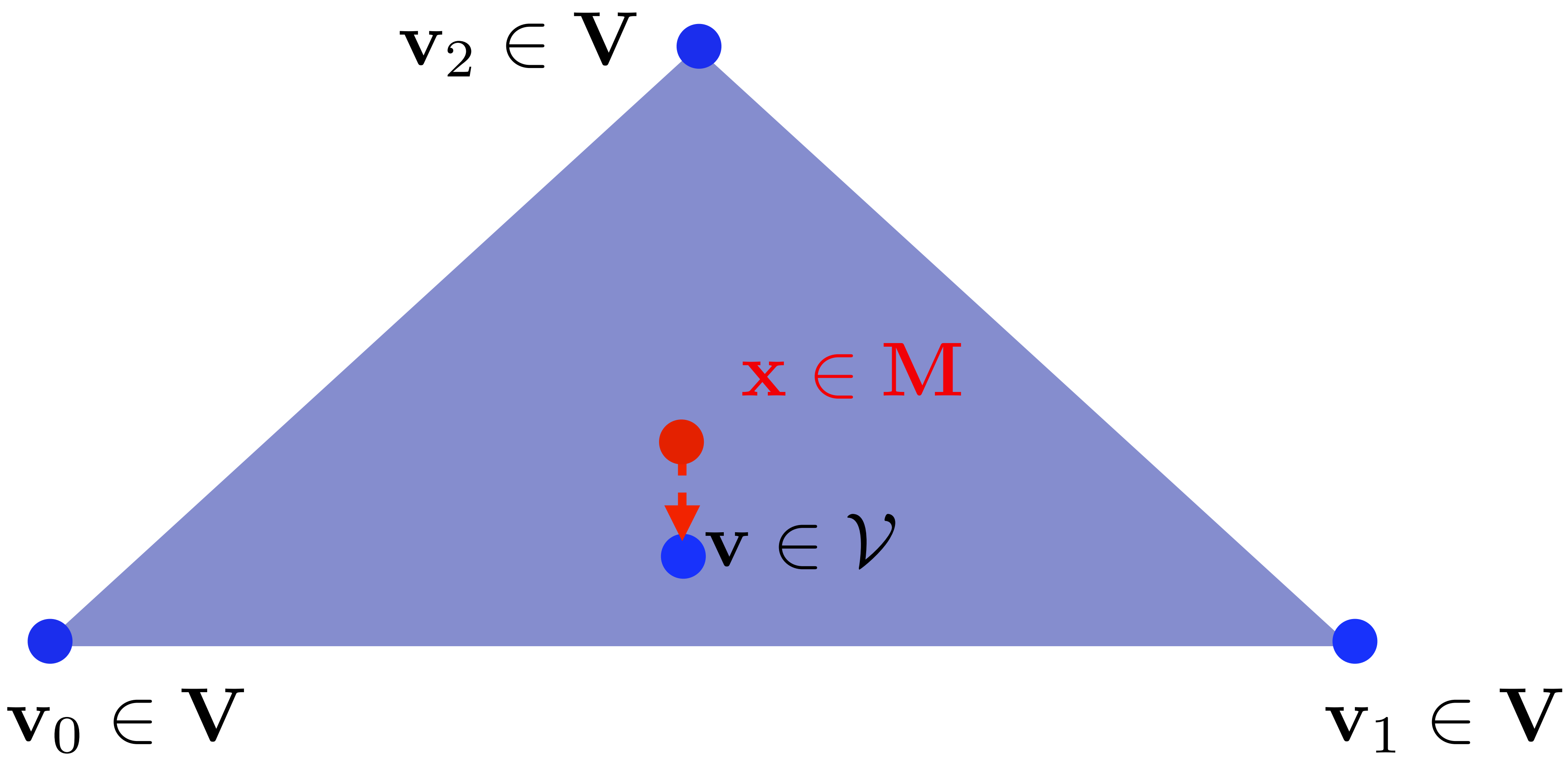
Point-to-surface distance



Point-to-surface distance



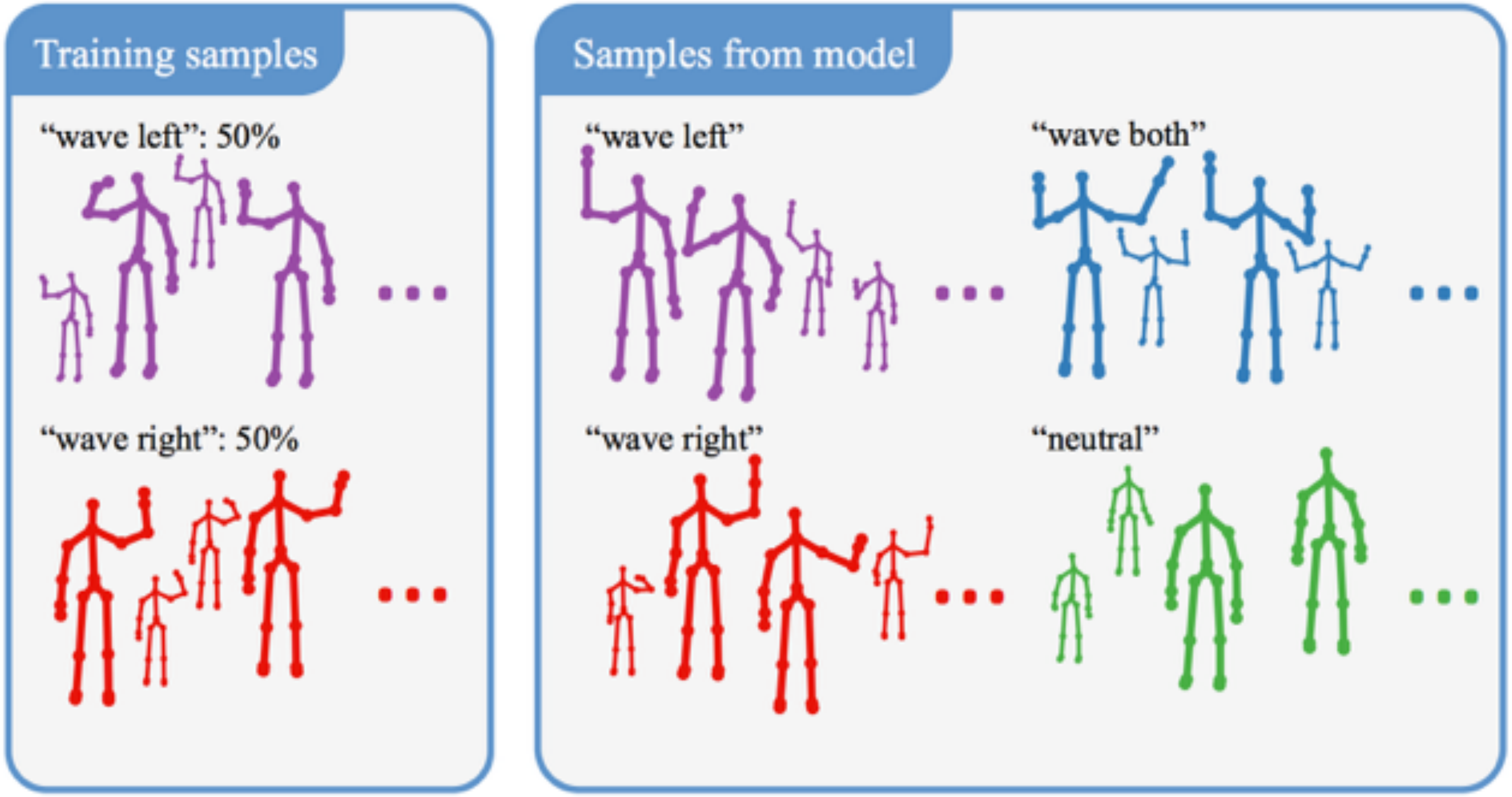
Point-to-surface distance



Implementation requires taking care of special cases when v falls in edges or points

Advanced registration

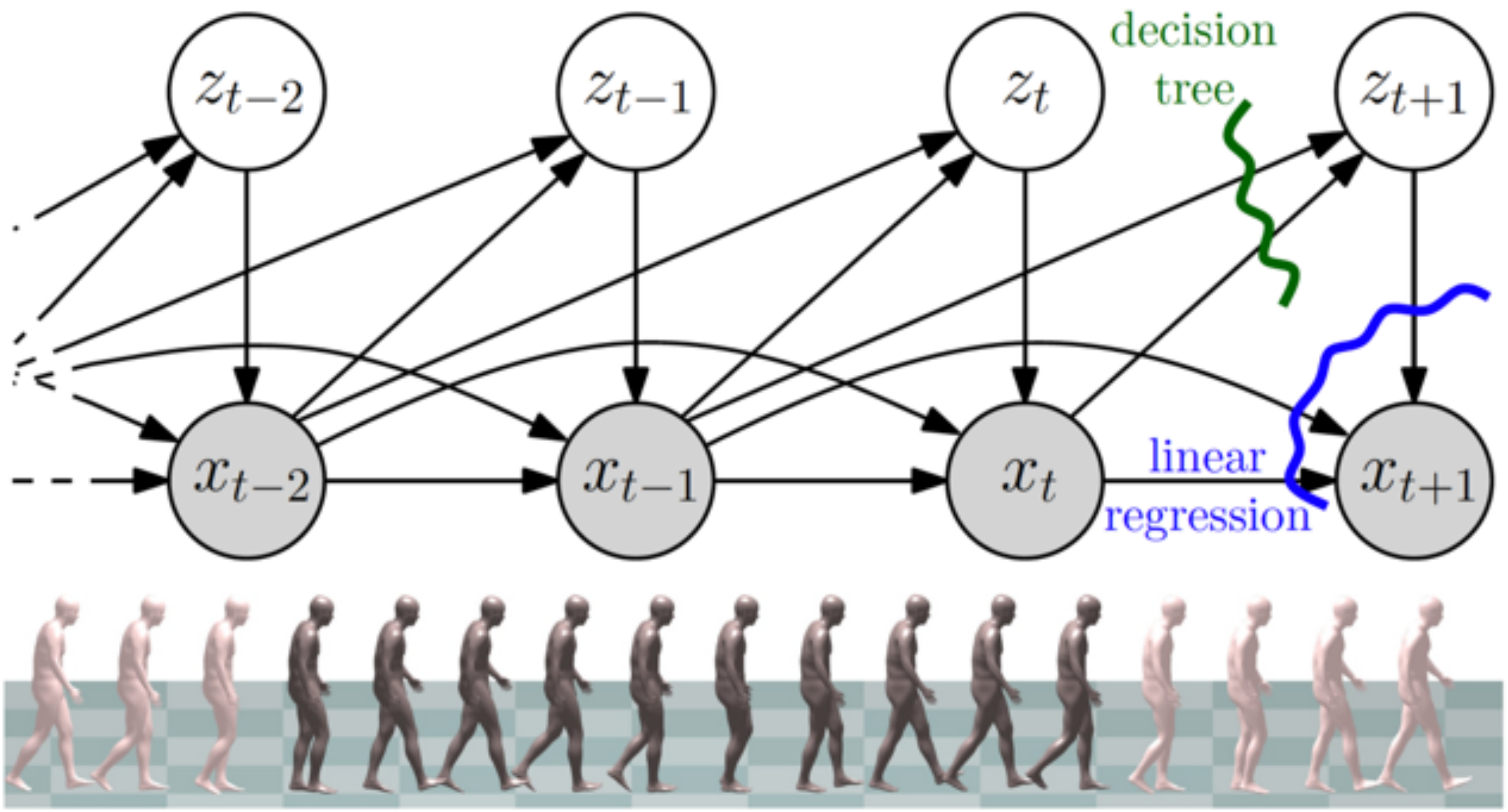
- Better pose priors
  - Non-parametric



A Non-parametric Bayesian Network Prior of Human Pose, Lehrman et al

Fitting SMPL to a scan/mesh

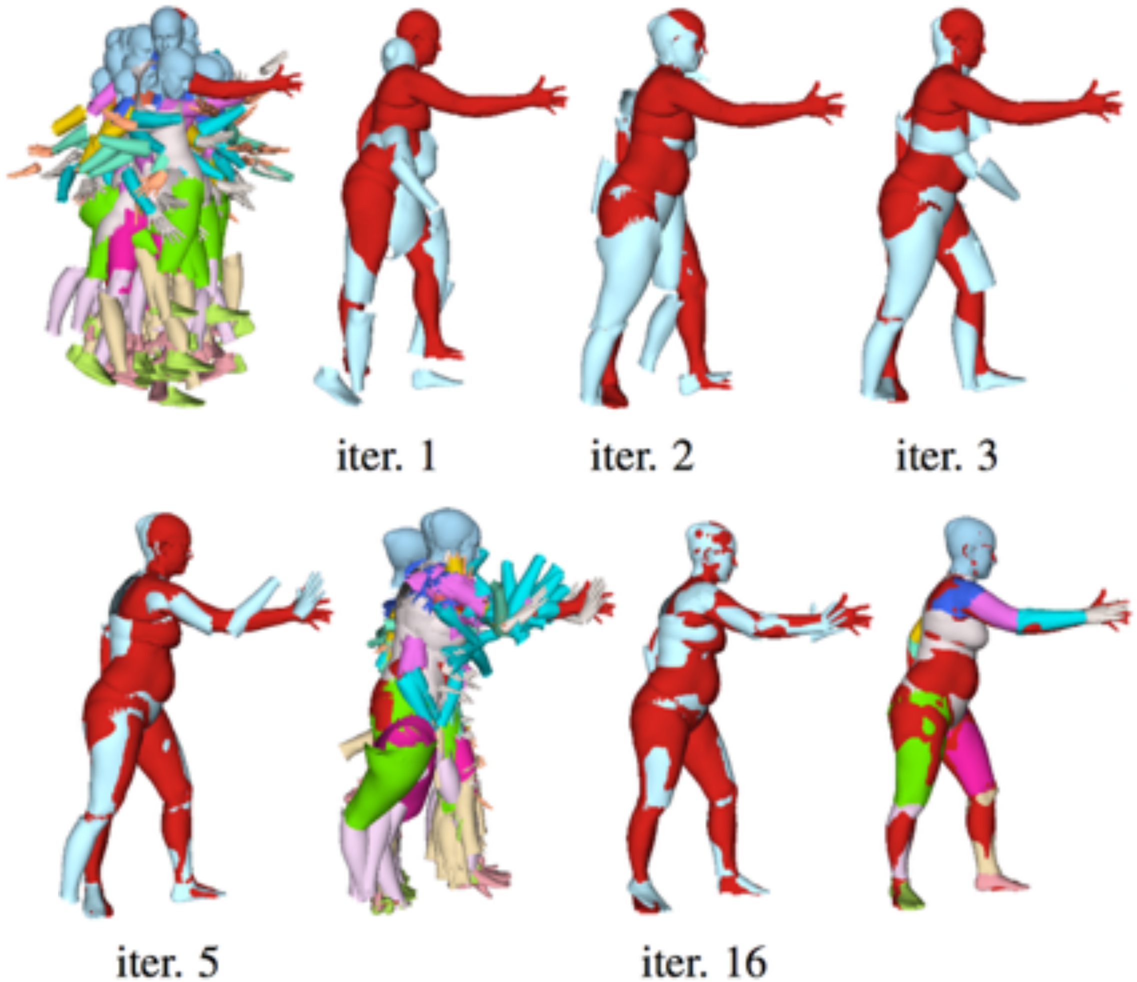
- Better pose priors
  - Non-parametric
  - Dynamic



Efficient Nonlinear Markov Models for Human Motion, Lehrman et al

Fitting SMPL to a scan/mesh

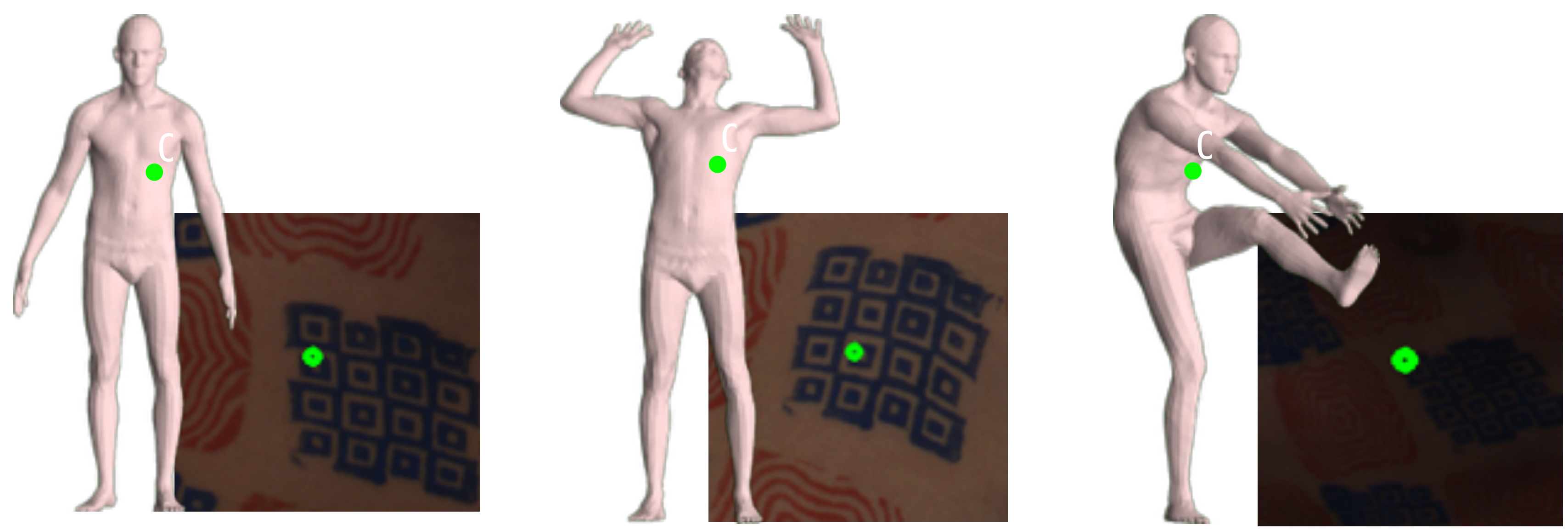
- Better pose priors
  - Non-parametric
  - Dynamic
- Better initialisation
  - From previous frame, from discriminative approaches, from graphical models



The Stitched Puppet: A Graphical Model of 3D Human Shape and Pose, Zuffi and Black

Fitting SMPL to a scan/mesh

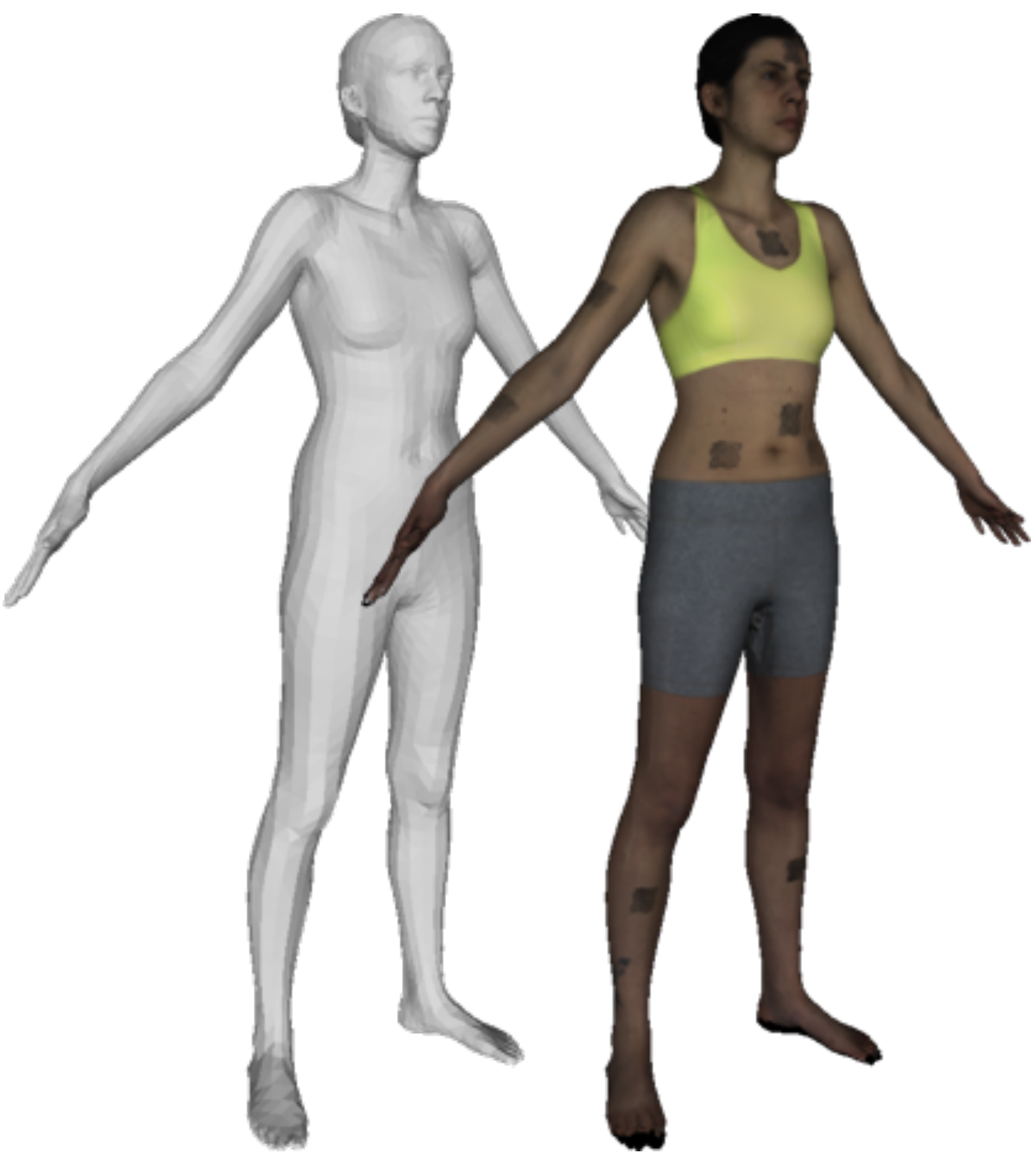
- Better pose priors
  - Non-parametric
  - Dynamic



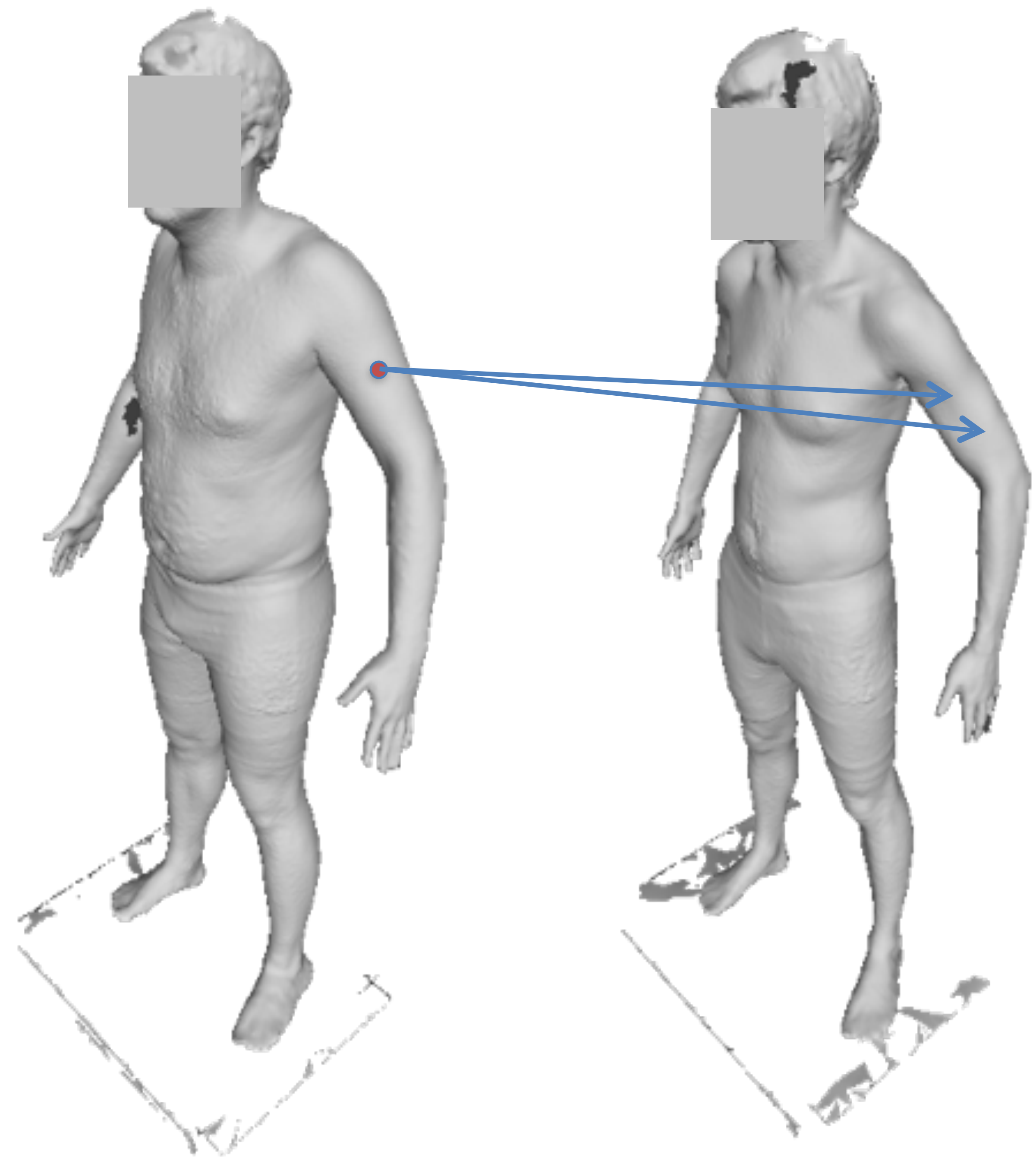
- Better initialisation
  - From previous frame, from discriminative approaches, from graphical models
- Other information: appearance (color)!

Why appearance

More realism

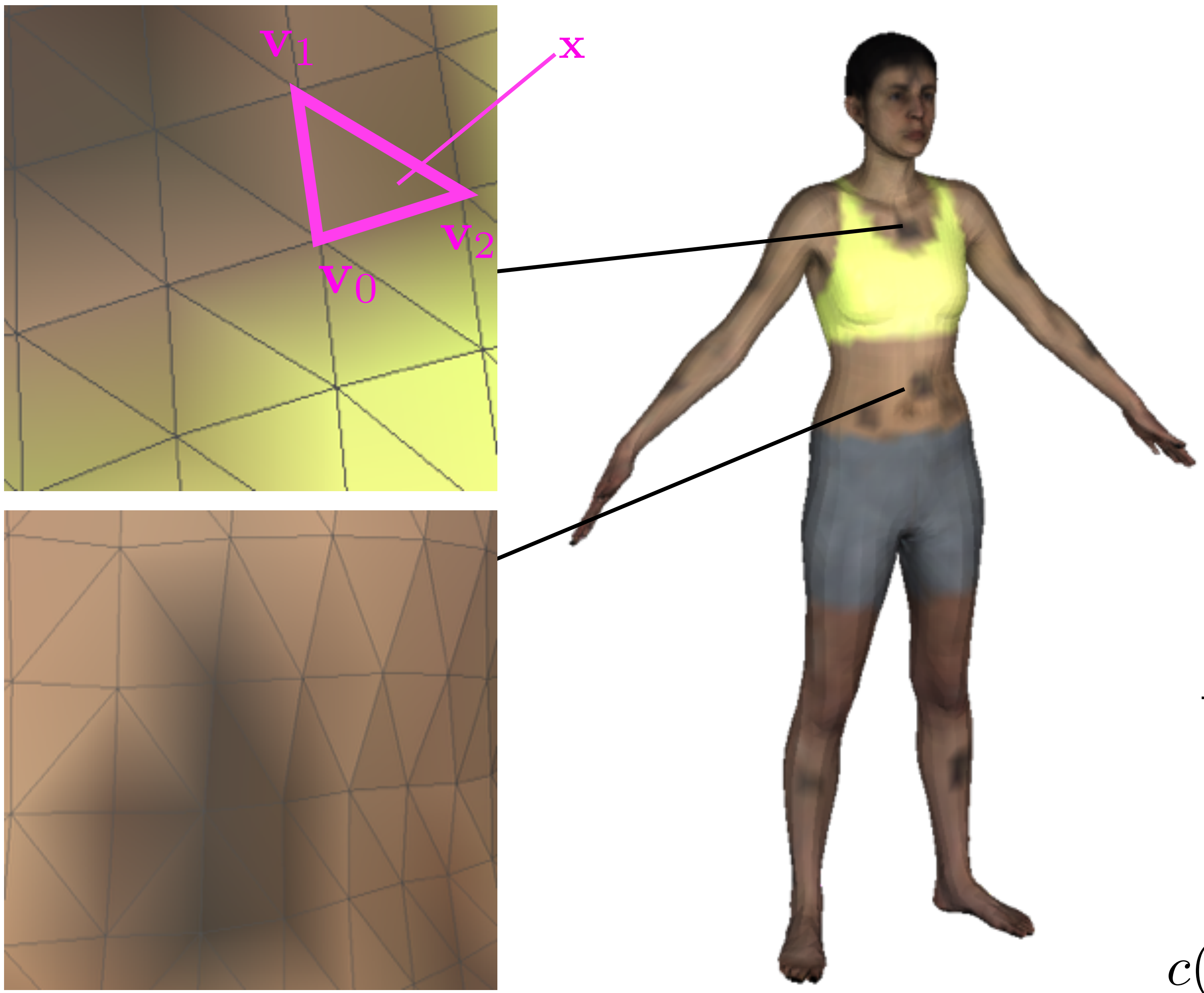


More accurate correspondences



Representing appearance

Vertex coloring



$$\mathbf{V} \in \mathbb{R}^{N\times 3}$$

$$\mathbf{F} \in \mathbb{N}^{M\times 3}, \mathbf{F}_{ij} \in [0, N)$$

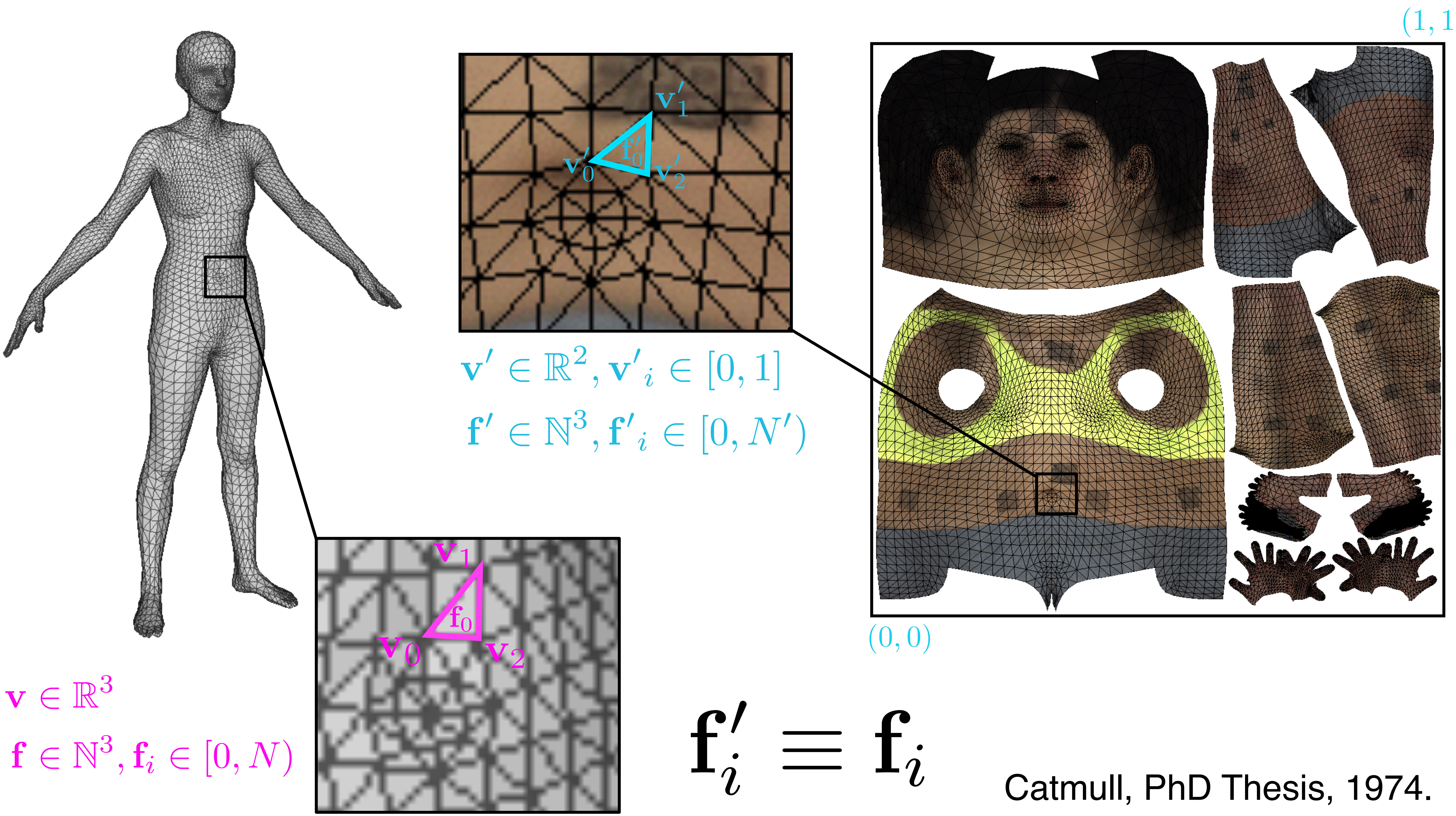
$$\mathbf{W} \in \mathbb{R}^{N\times 3}, \mathbf{W}_{ij} \in [0, 256)$$

$$\mathbf{f}_0 = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2]$$

$$\mathbf{x} \equiv \alpha_0\mathbf{v}_0 + \alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2$$

$$c(\mathbf{x}) = \alpha_0\mathbf{w}_0 + \alpha_1\mathbf{w}_1 + \alpha_2\mathbf{w}_2$$

Decouple geometry and appearance resolution

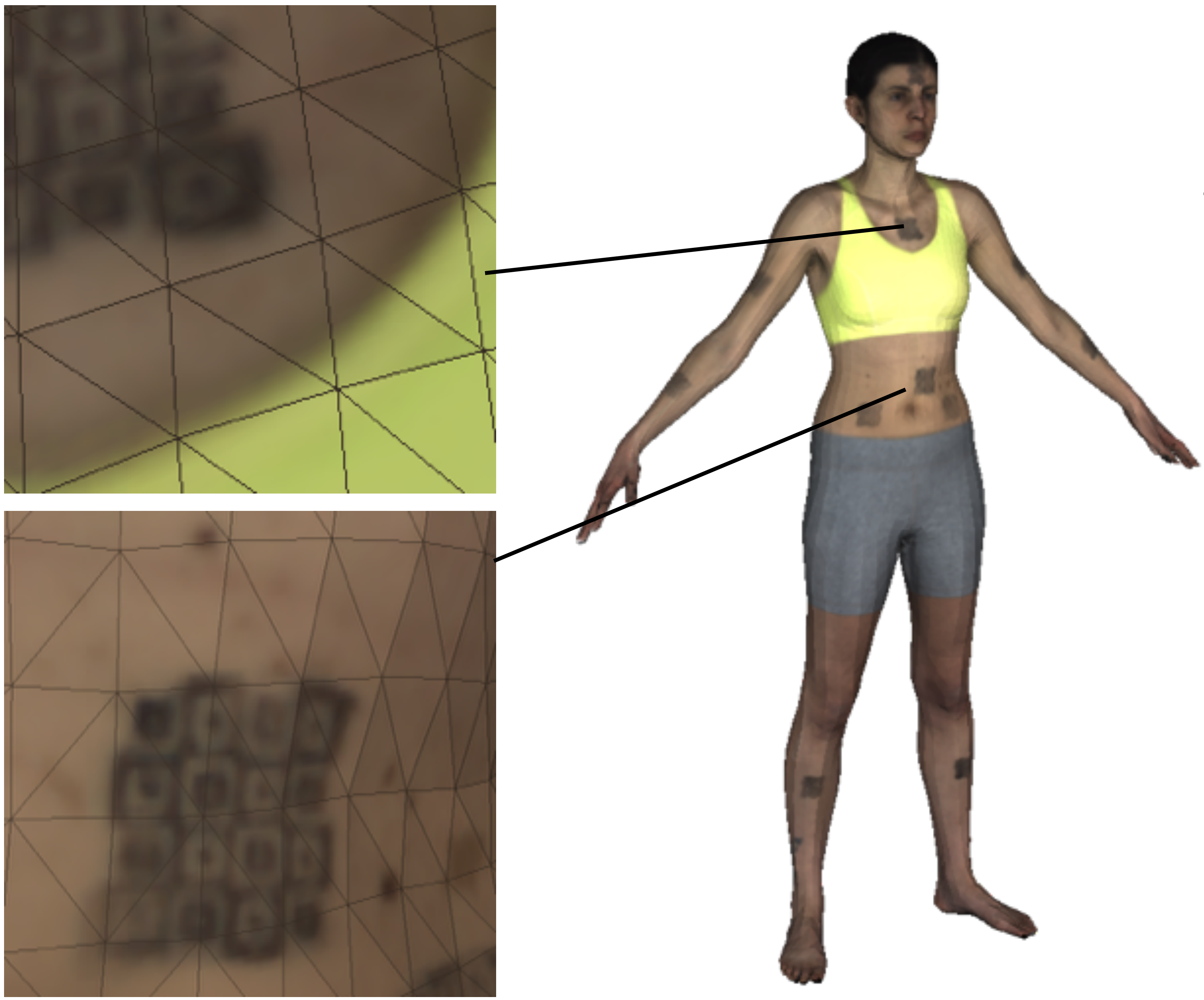


$$\mathbf{f}'_i \equiv \mathbf{f}_i$$

Catmull, PhD Thesis, 1974.

Representing appearance

Texture mapping



$$\mathbf{V} \in \mathbb{R}^{N \times 3}$$

$$\mathbf{F} \in \mathbb{N}^{M \times 3}, \mathbf{F}_{ij} \in [0, N)$$

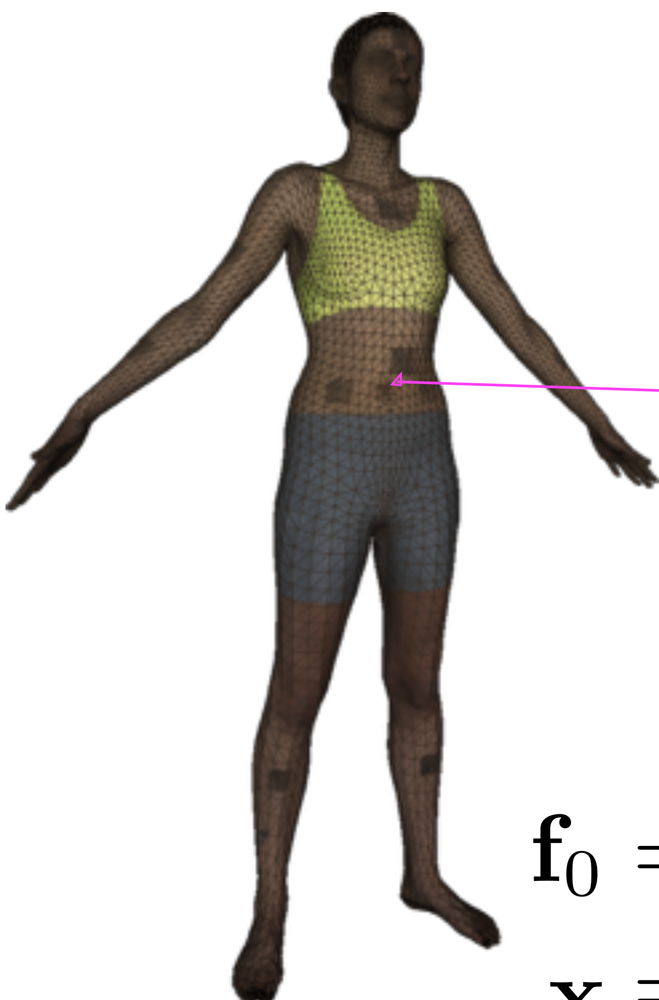
$$\mathbf{V}' \in \mathbb{R}^{N' \times 2}, \mathbf{V}'_{ij} \in [0, 1]$$

$$\mathbf{F}' \in \mathbb{N}^{M \times 2}, \mathbf{F}'_{ij} \in [0, N')$$

$$\mathbf{U} \in \mathbb{N}^{K \times K \times 3}, \mathbf{U}_{ijk} \in [0, 256)$$

Texture mapping

3D → UV



$$\mathbf{f}_0 = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2]$$

$$\mathbf{x} \equiv \alpha_0\mathbf{v}_0 + \alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2$$

$$\mathbf{f}'_0 = [\mathbf{v}'_0, \mathbf{v}'_1, \mathbf{v}'_2]$$

$$c(\mathbf{x}) = \mathbf{U}[\alpha_0\mathbf{v}'_0 + \alpha_1\mathbf{v}'_1 + \alpha_2\mathbf{v}'_2]$$

Catmull, PhD Thesis, 1974.

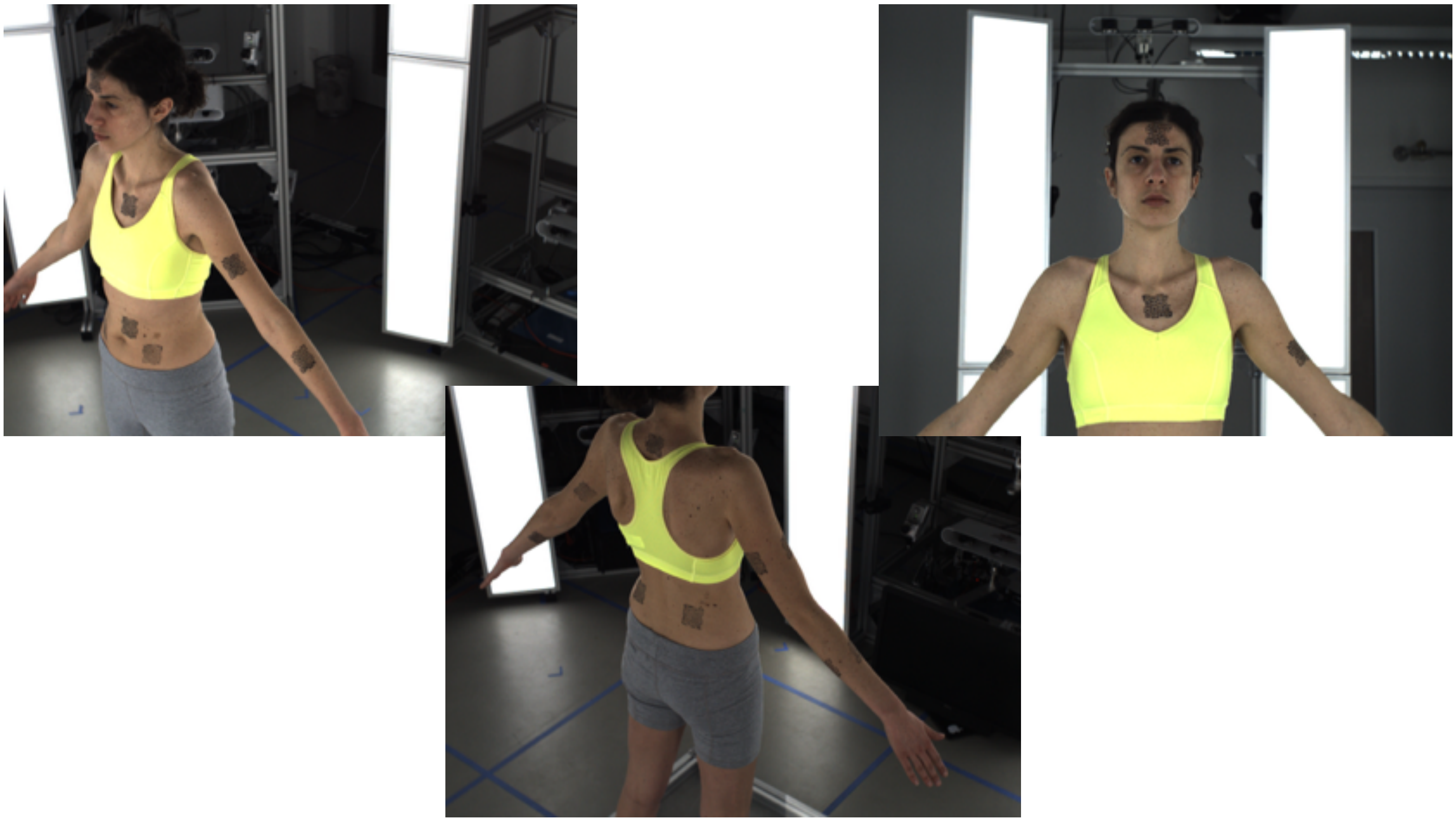
How do we create texture maps?

From 2D images to textures

Problem: combining multiple views of a 3D surface

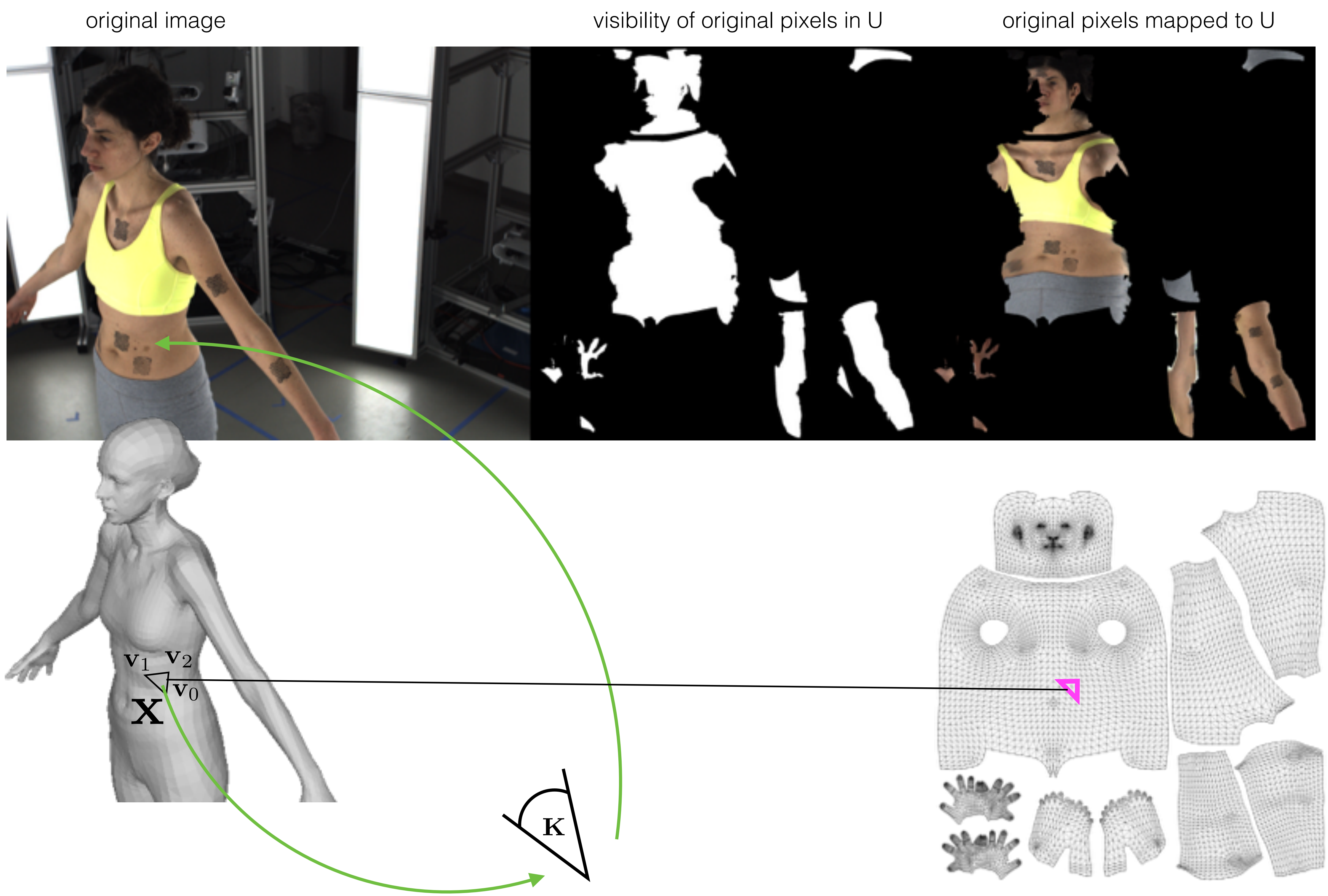
From 2D images to textures

Problem: combining multiple views of a 3D surface

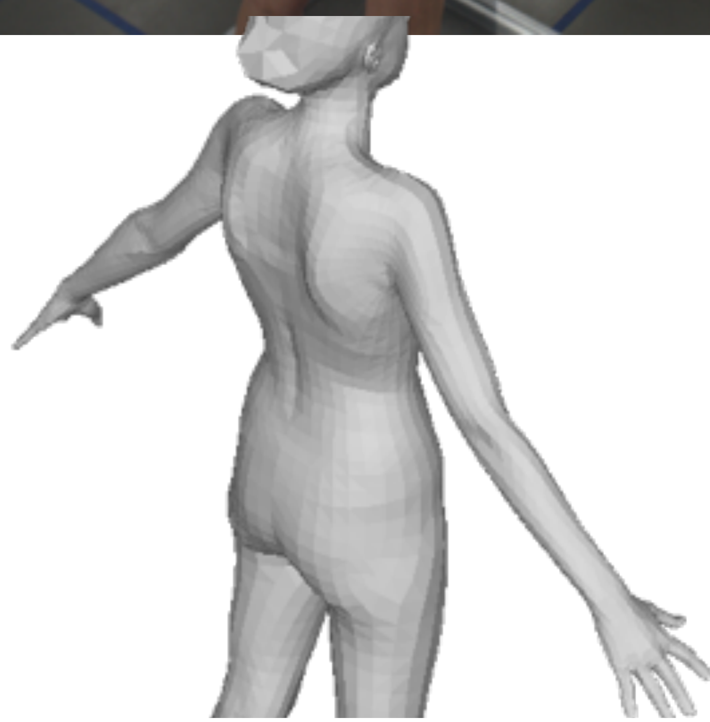


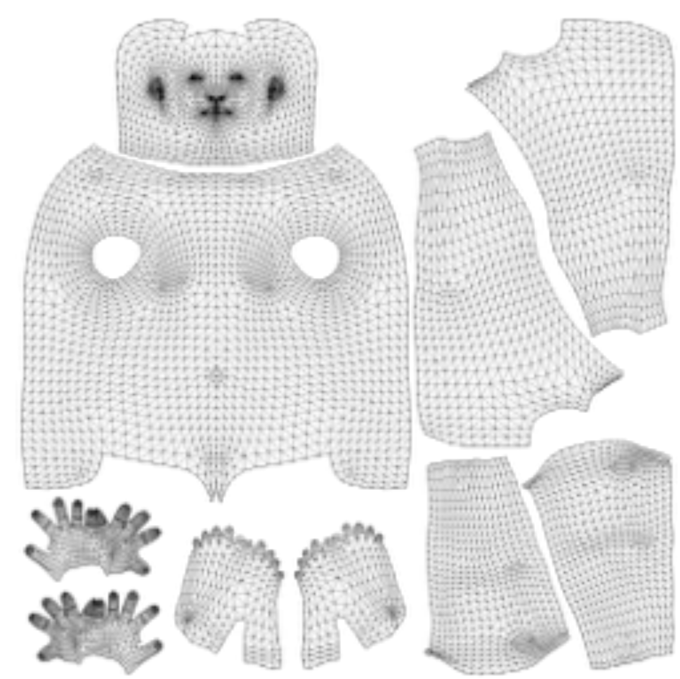
From 2D images to textures

From 2D images to textures

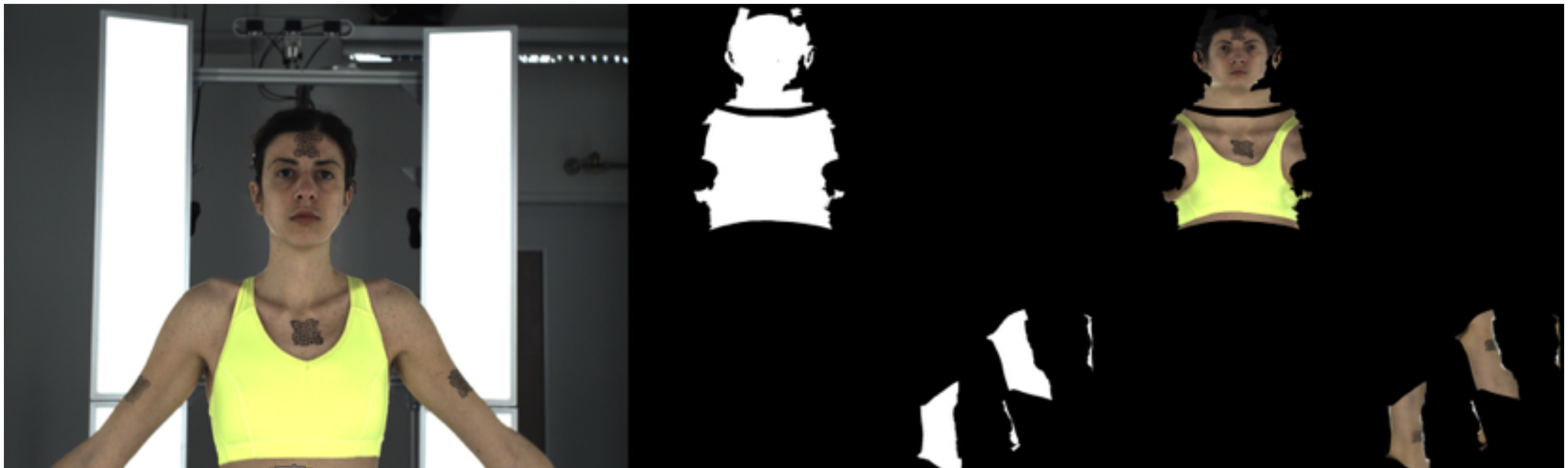


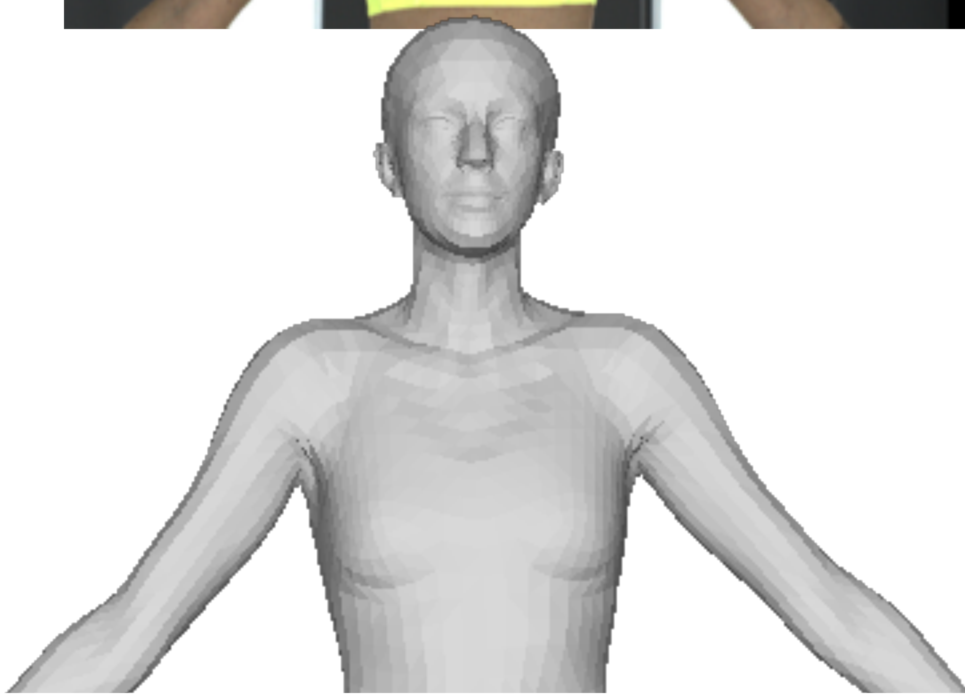
From 2D images to textures

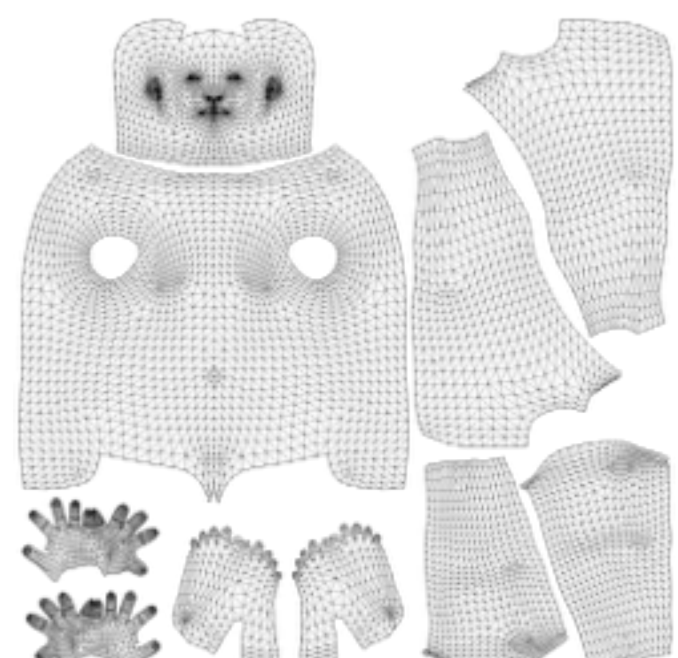




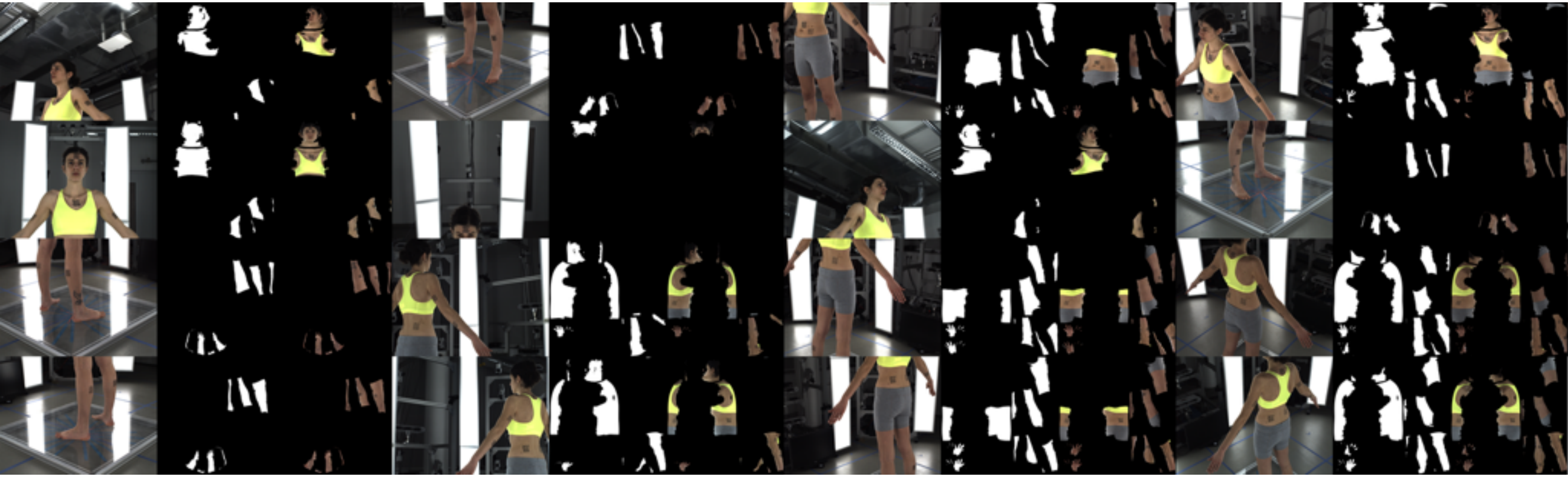
From 2D images to textures

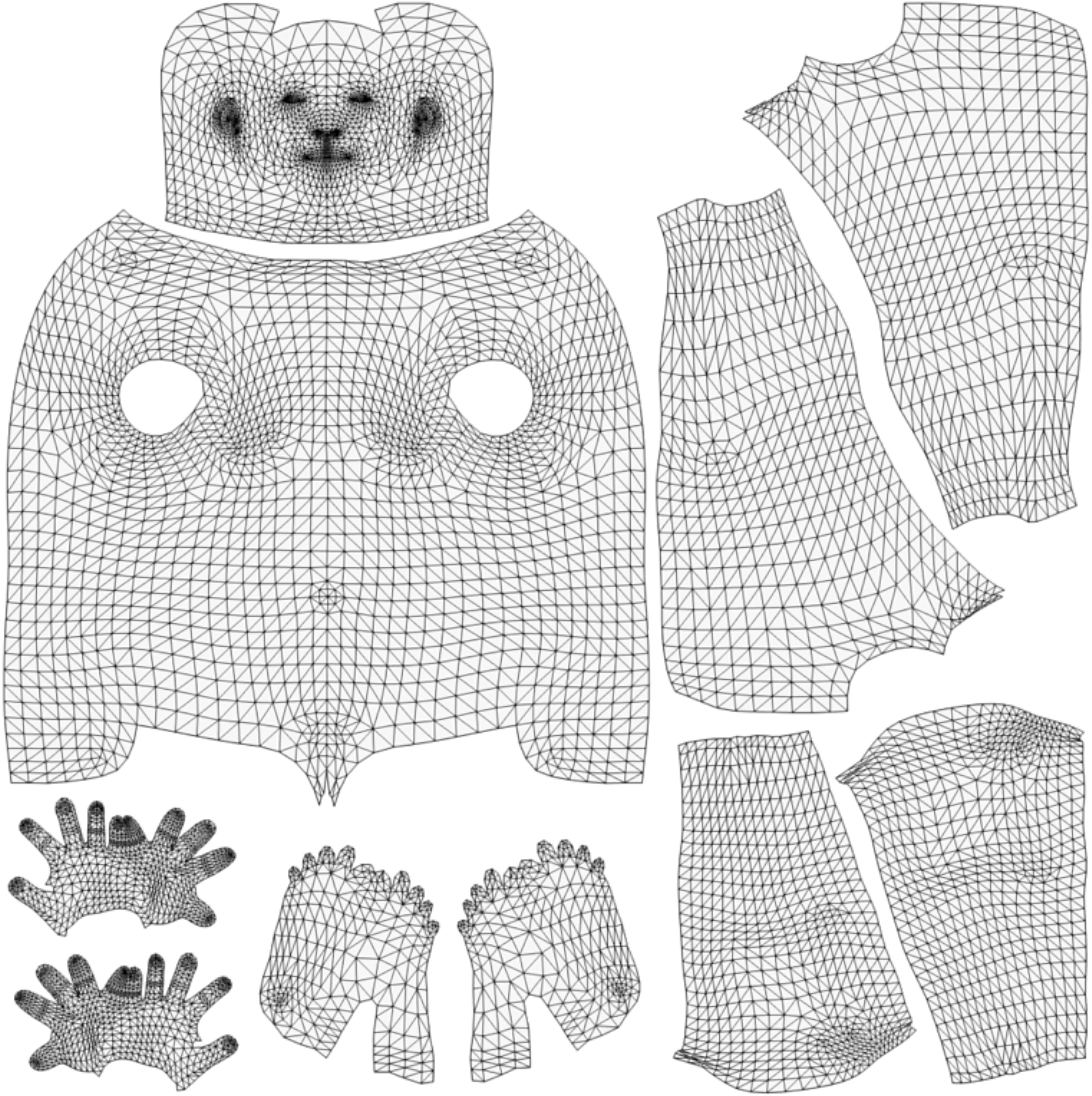




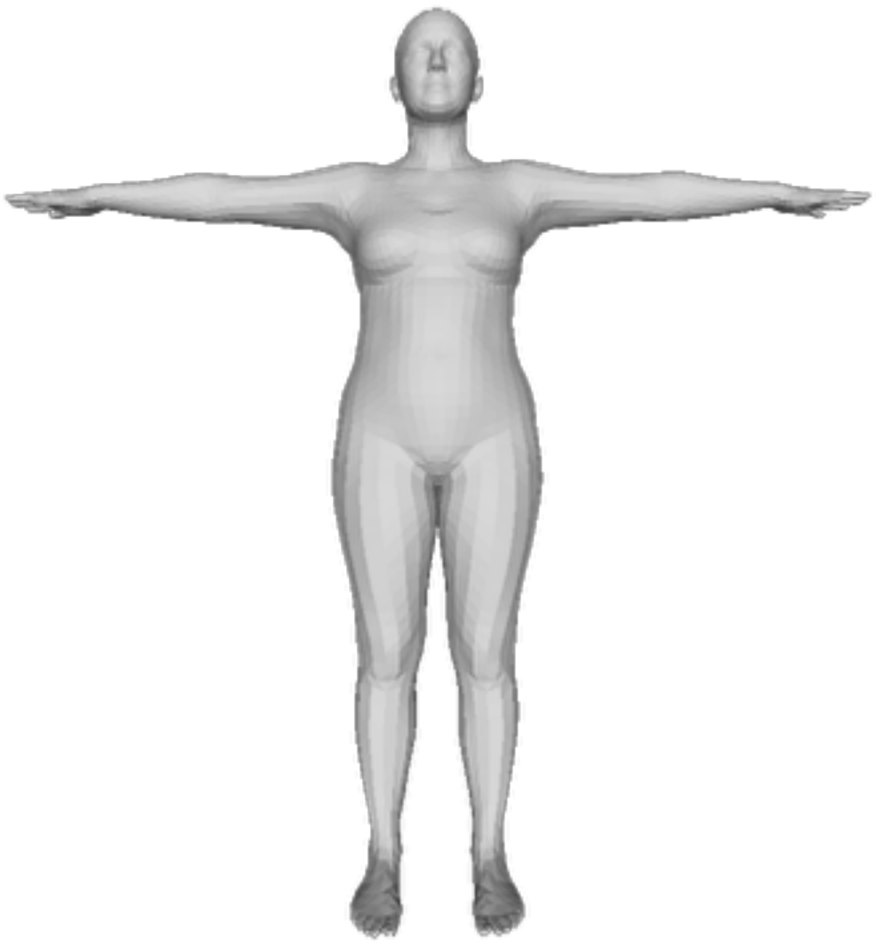


From 2D images to textures





Generating an image



image

Generating an image

shape

β

pose

θ

image

Generating an image

shape

β

pose

θ

camera

K

image

Generating an image

shape

β

pose

θ

UV map

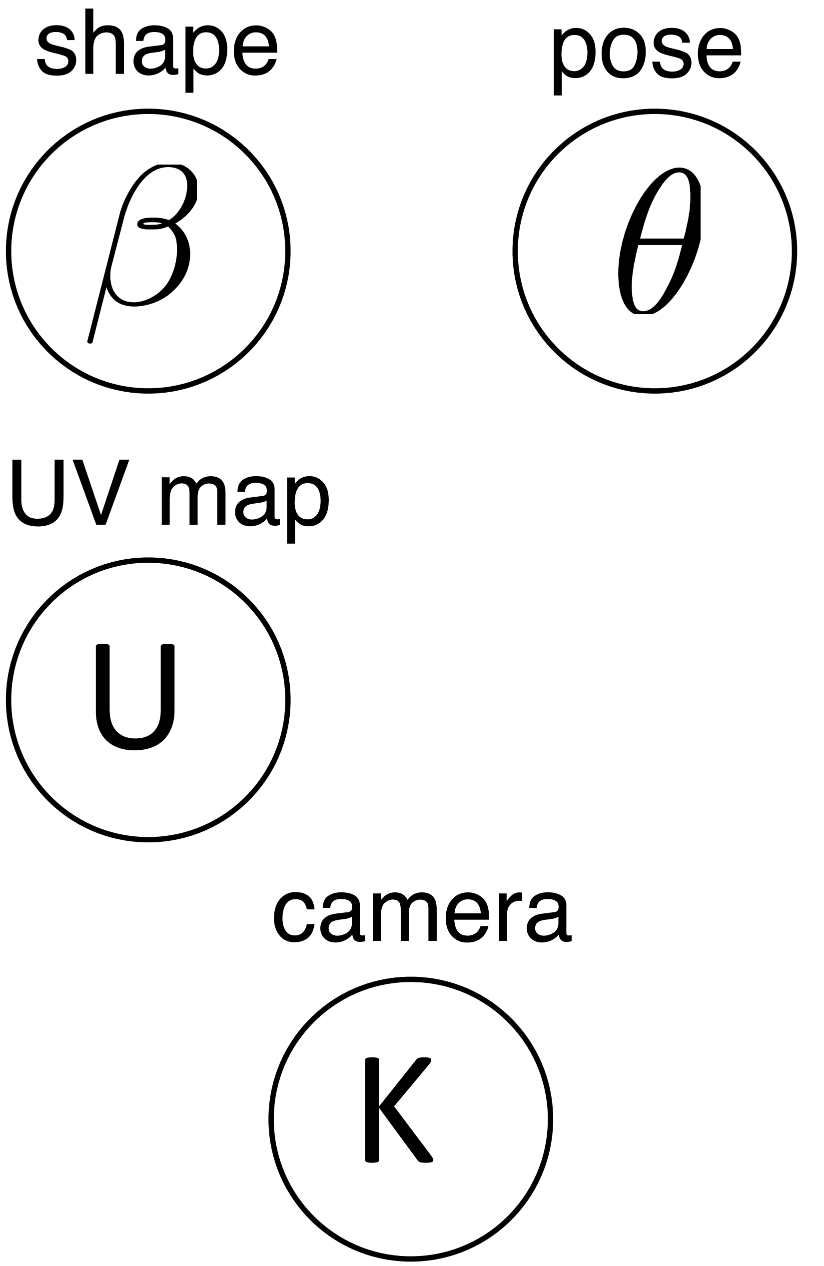
U

camera

K

image

That’s all, no?



image

This slide is wrong:

have all the vertices the same albedo?

shape

β

pose

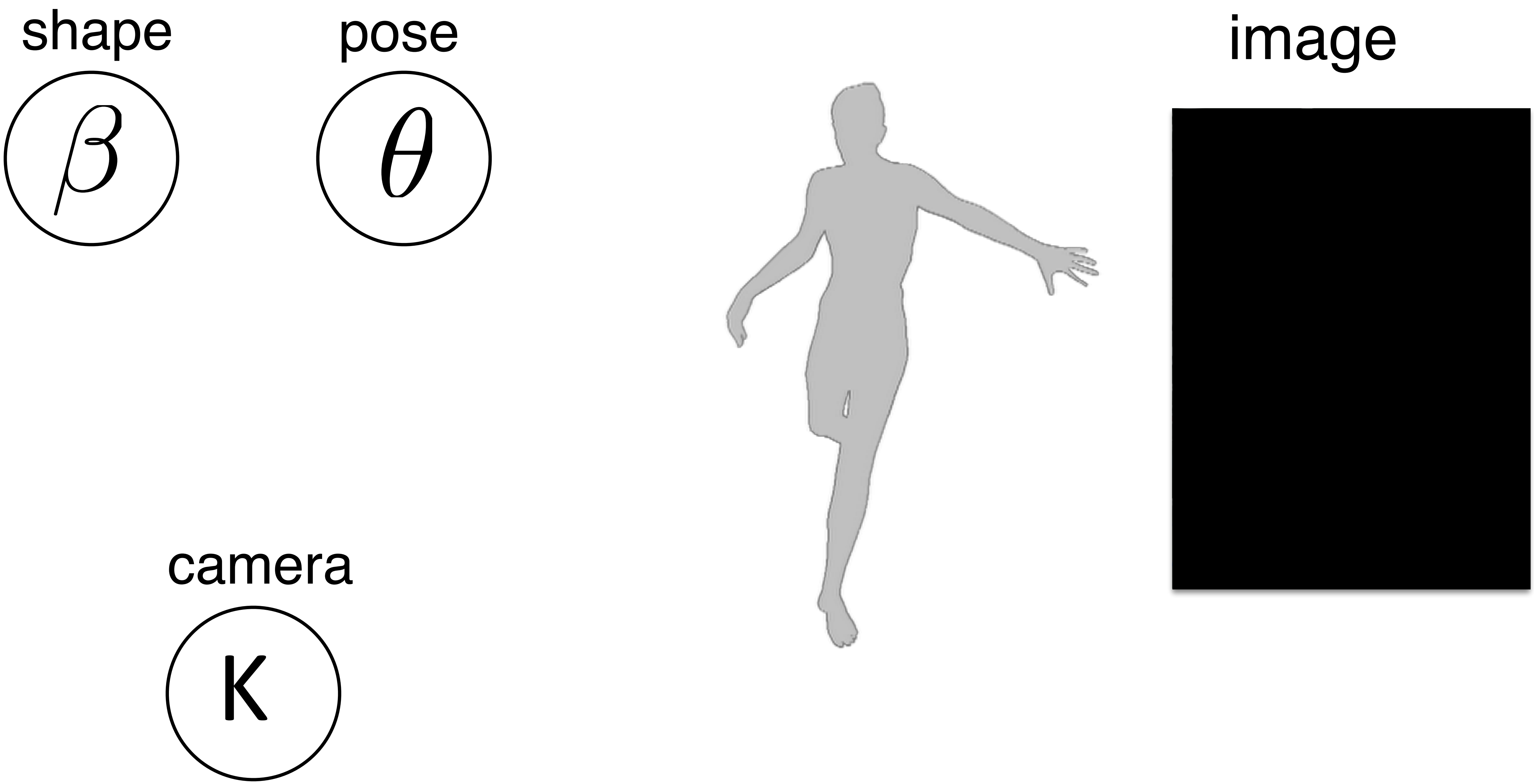
θ

camera

K

image

This one has a single albedo



Generating an image

shape

β

pose

θ

lighting

l

camera

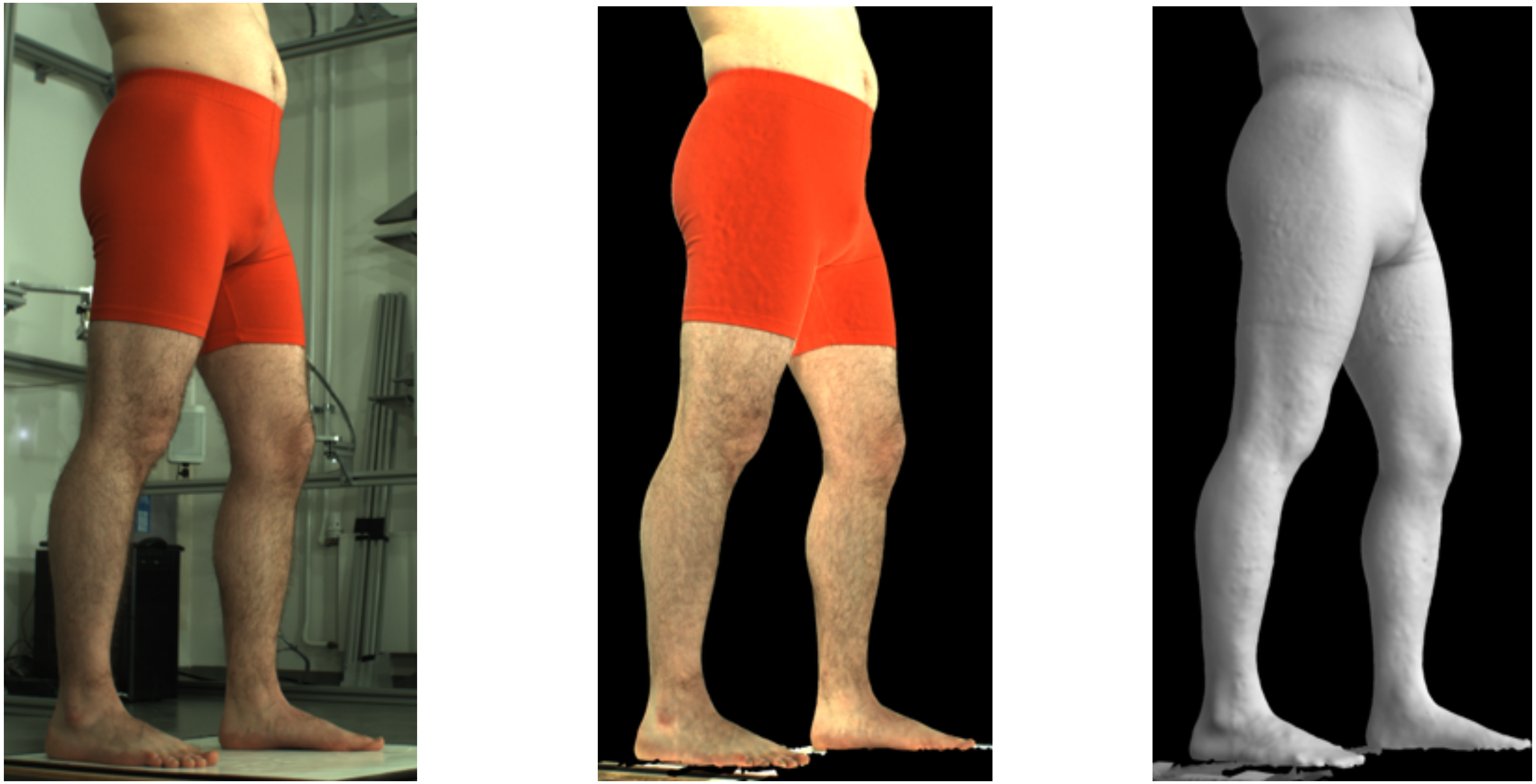
K

image

Albedo and shading

Albedo is constant: depends on physical properties of the surface

Shading is transient: given by the interplay between surface reflectance and lighting



real image albedo shading

Reflectance models

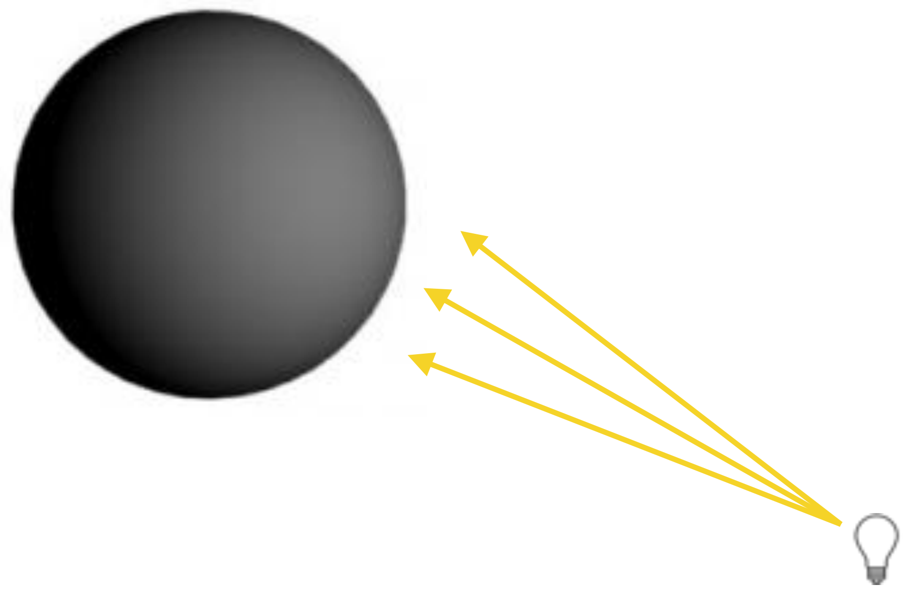
Lambertian reflectance

$$i_x = (\boldsymbol{n}_x \cdot \boldsymbol{l}_x) a_x l$$

- i_x: surface color
- $\boldsymbol{n}_x$: surface normal
- $\boldsymbol{l}_x$: direction from *x* to light source
- a_x: albedo
- l: light intensity

Lighting models

Point light sources

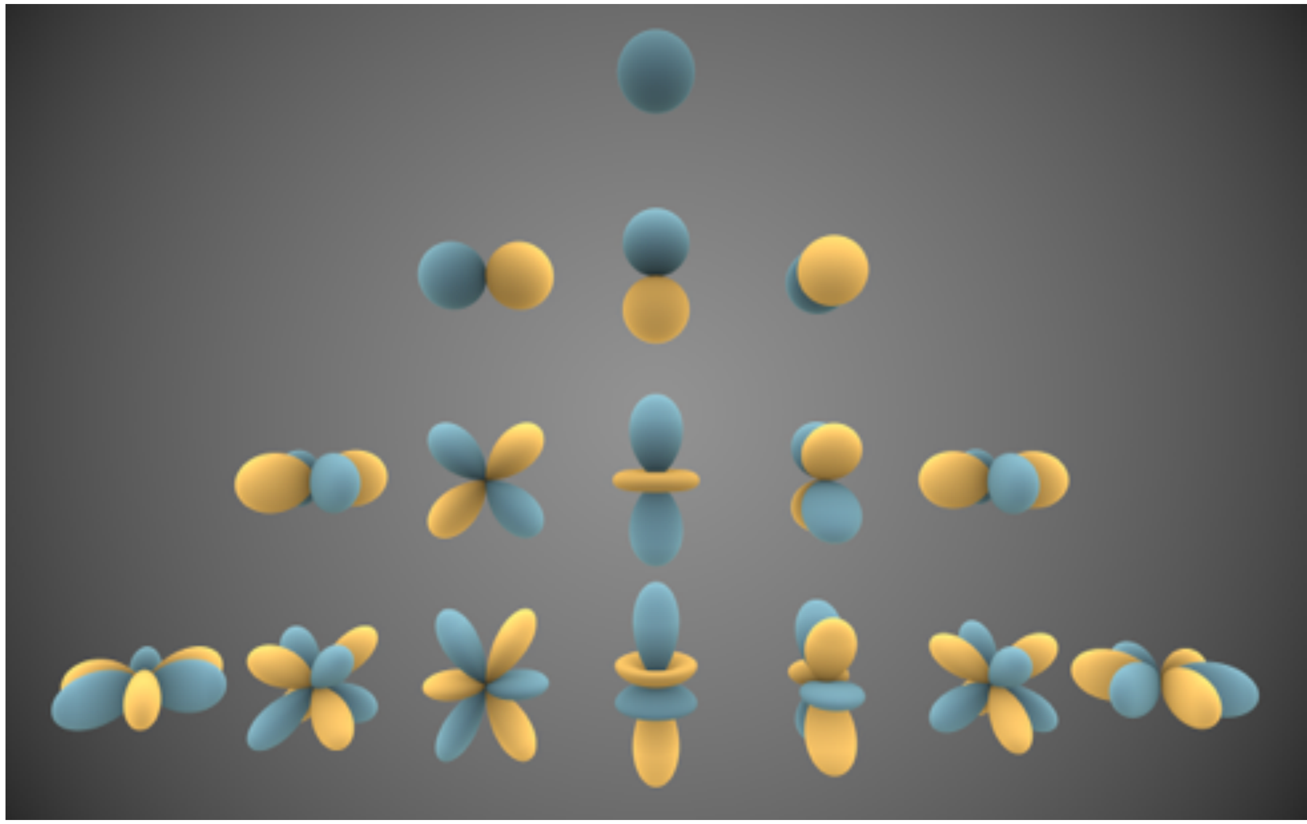


Lighting models

Spherical Harmonics (SH)

Lighting as a function over the sphere, projected onto a low-order SH basis

Simple and efficient for diffuse environments



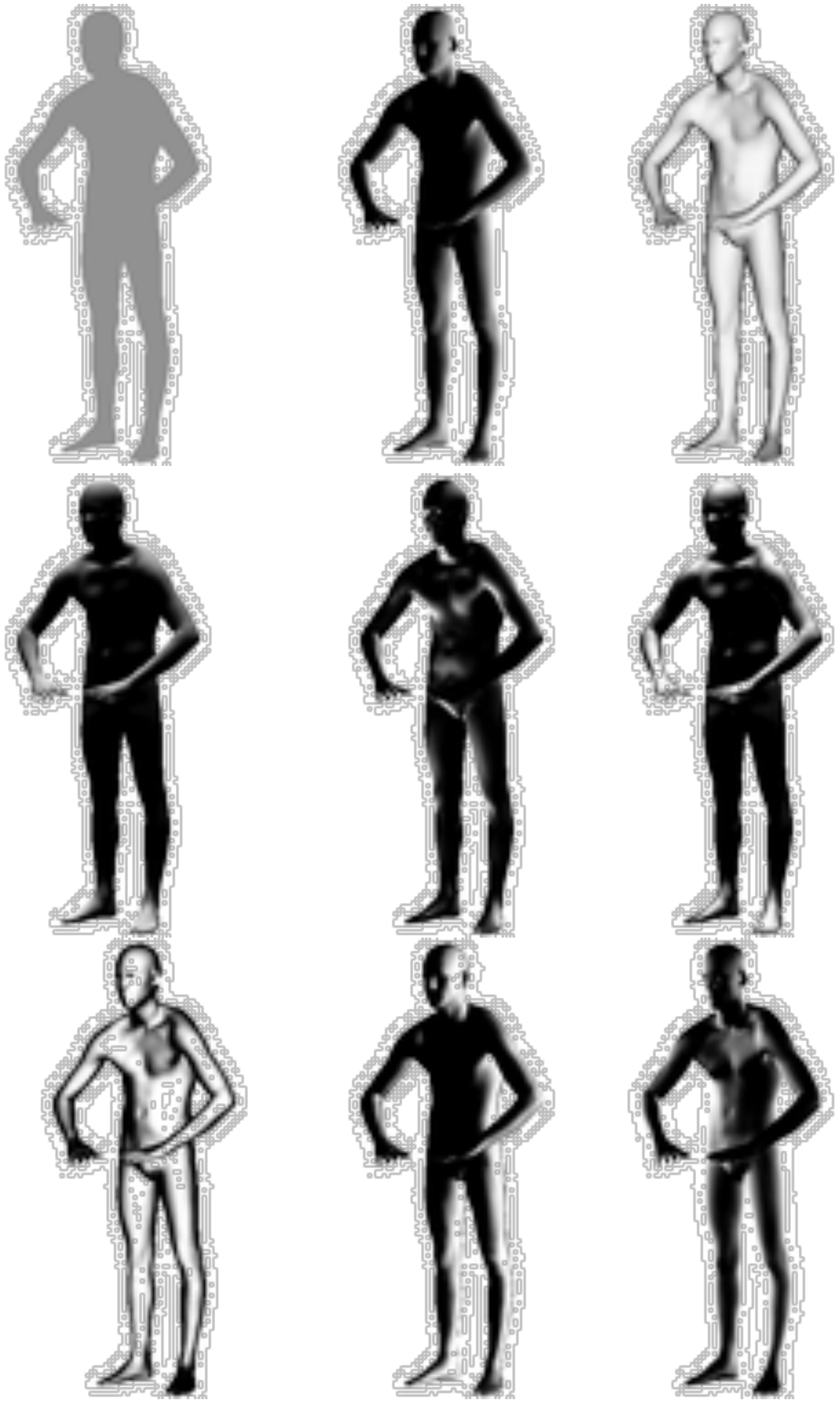
Sloan et al., SIGGRAPH 2002.
Basri et al., IEEE TPAMI, 2003.

Lighting models

Spherical Harmonics (SH)

Lighting as a function over the sphere, projected onto a low-order SH basis

Simple and efficient for diffuse environments



Sloan et al., SIGGRAPH 2002.
Basri et al., IEEE TPAMI, 2003.

Modeling all together

shape β

pose θ

UV map U

lighting l

camera K

images

Forward rendering process

shape β

pose θ

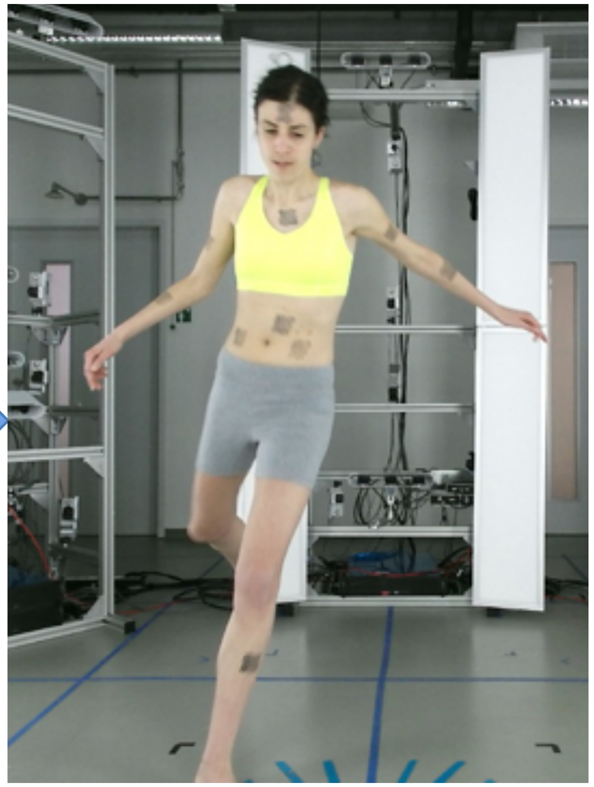
UV map U

lighting l

camera K

$f(\beta, \theta, U, l, K)$

images



Rendering takes model parameters and produces images.

Gradient-based optimization?

- We want to exploit images to obtain better registrations
- We saw that we can optimise a function given its derivatives
- Most of the functions involved in the rendering are linear operators
- Anybody wants to write the jacobians by hand?

OpenDR

An open source differentiable rendering framework for:

- approximating a rendering process
- differentiating this approximation
- finding parameter estimates

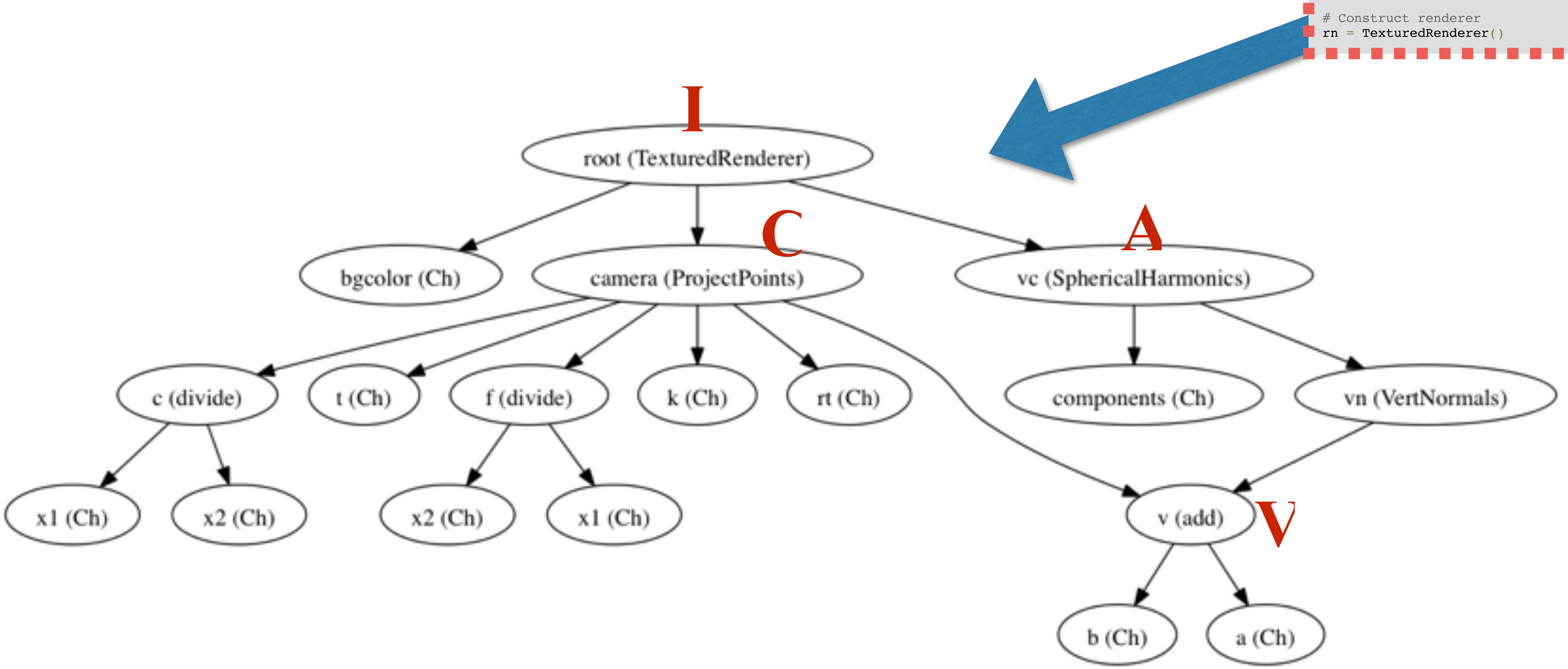
OpenDR differentiable renderer

http://open-dr.org

Loper and Black, ECCV 2014.

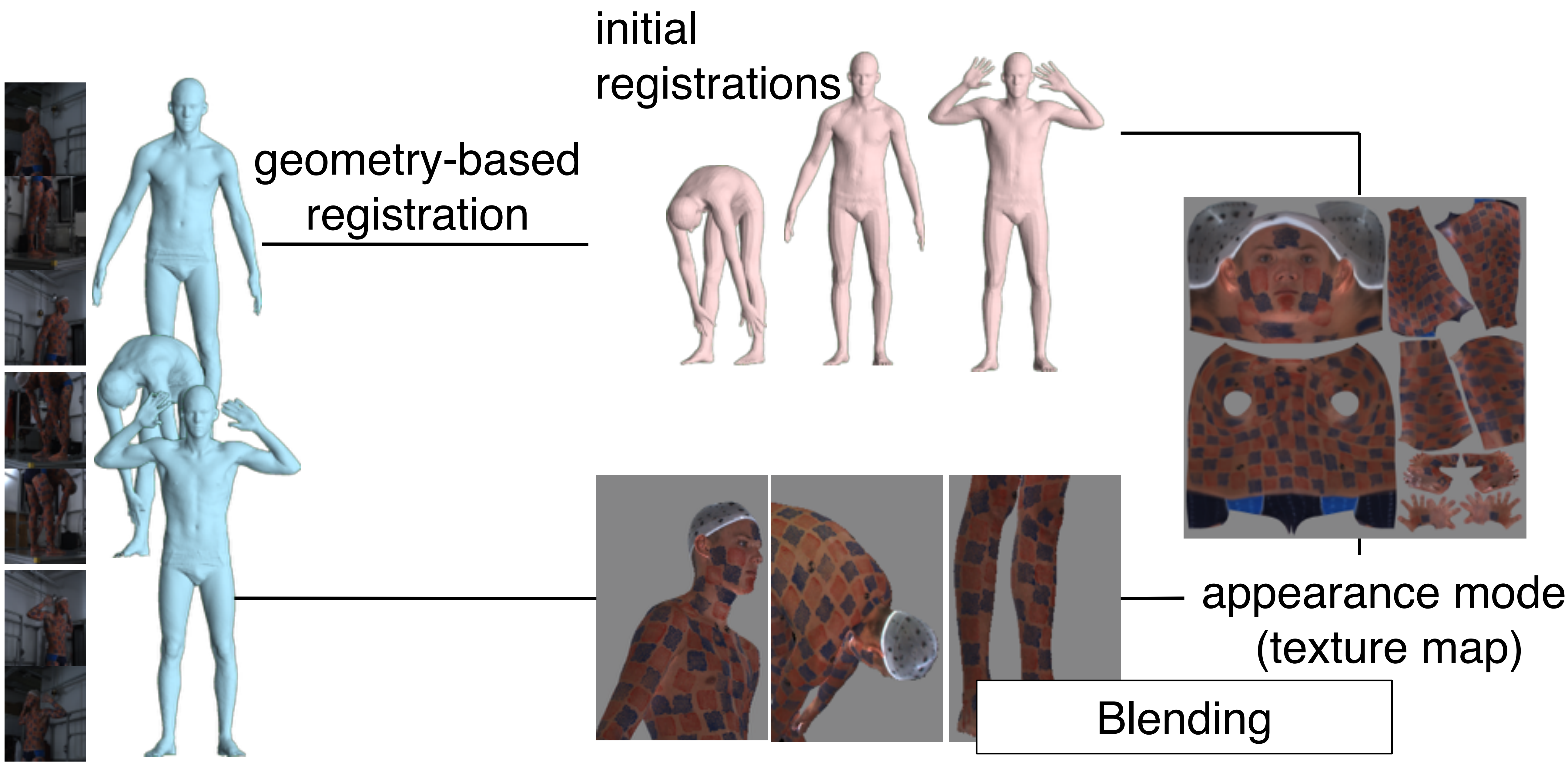
OpenDR

```python
import chumpy as ch
from opendr.everything import *

# Load mesh
m = load_mesh('/Users/matt/geist/OpenDR/test_dr/nasa_earth.obj')
m.v += ch.array([0,0,4])
w, h = (320, 240)
trans = ch.array([[0,0,0]])

# Construct renderer
rn = TexturedRenderer()
```

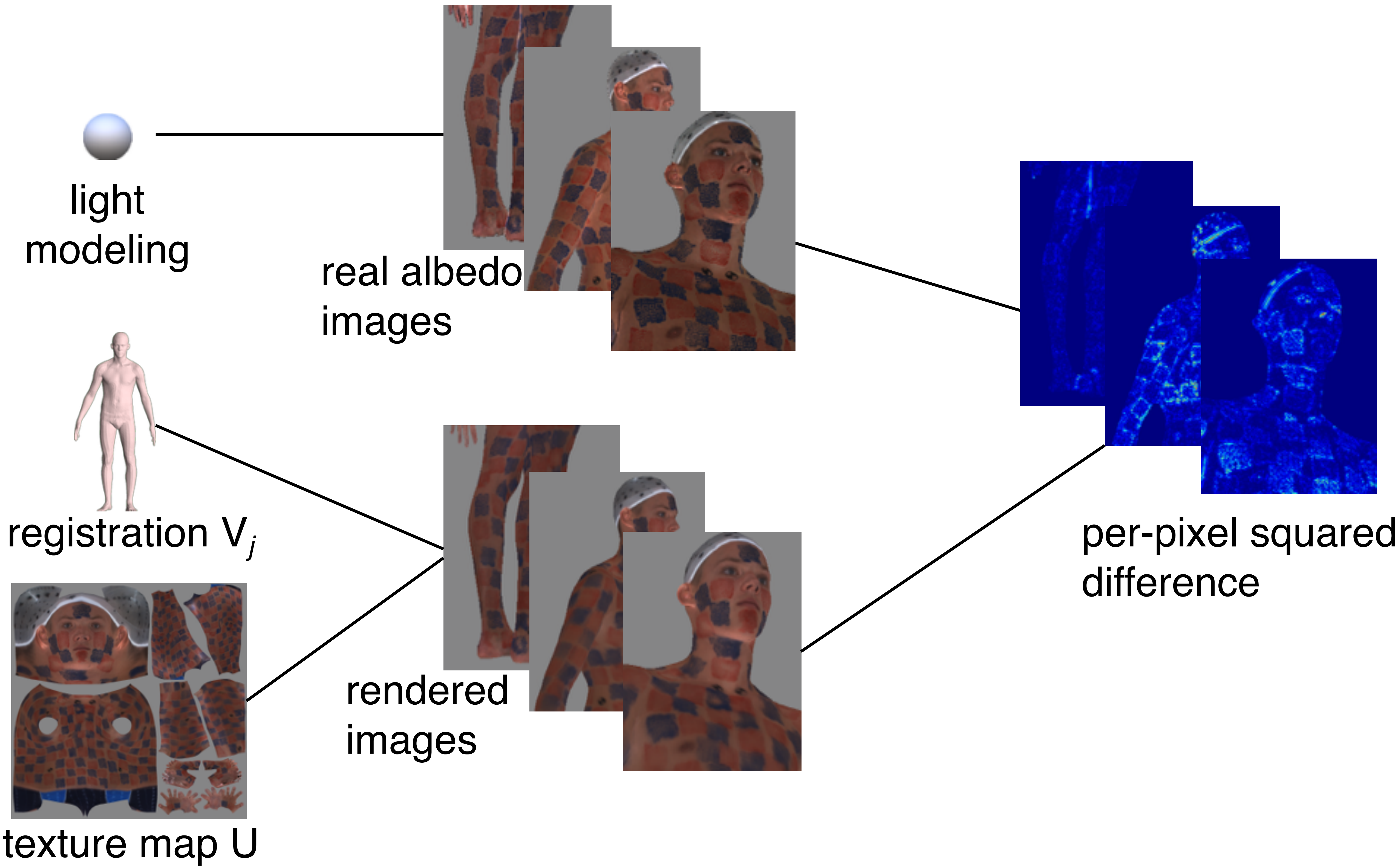


Appearance-based registration

Building an appearance model



Appearance-based error term



New registration objective

$$\vec{\theta}, \vec{\beta} = \arg\min_{\vec{\theta}, \vec{\beta}} \|M(\vec{\theta}, \vec{\beta}) - \mathbf{V}\|^2 + E_\theta(\vec{\theta}) + E_\beta(\vec{\beta}) + E_U(\mathbf{I}, \mathbf{K}, \mathbf{U}, M(\vec{\theta}, \vec{\beta}))$$

$$E_U \equiv \sum_i \|\mathbf{I}_i - r(M(\vec{\theta}, \vec{\beta}), \mathbf{U}, \mathbf{K}_i)\|^2$$

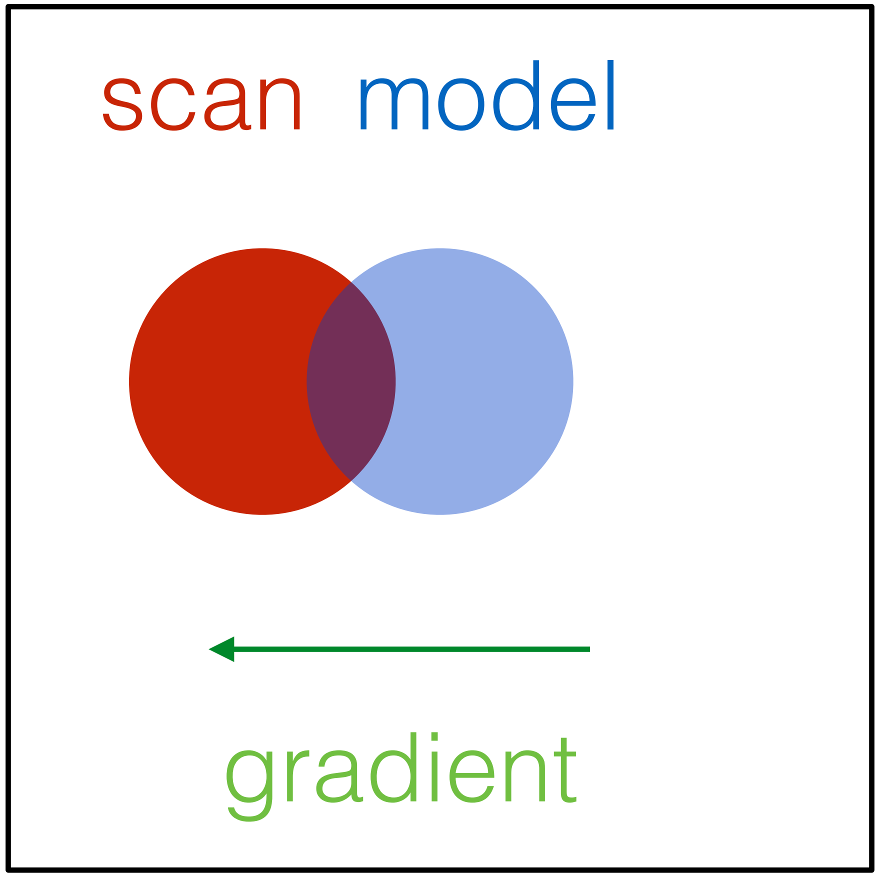
With OpenDR…

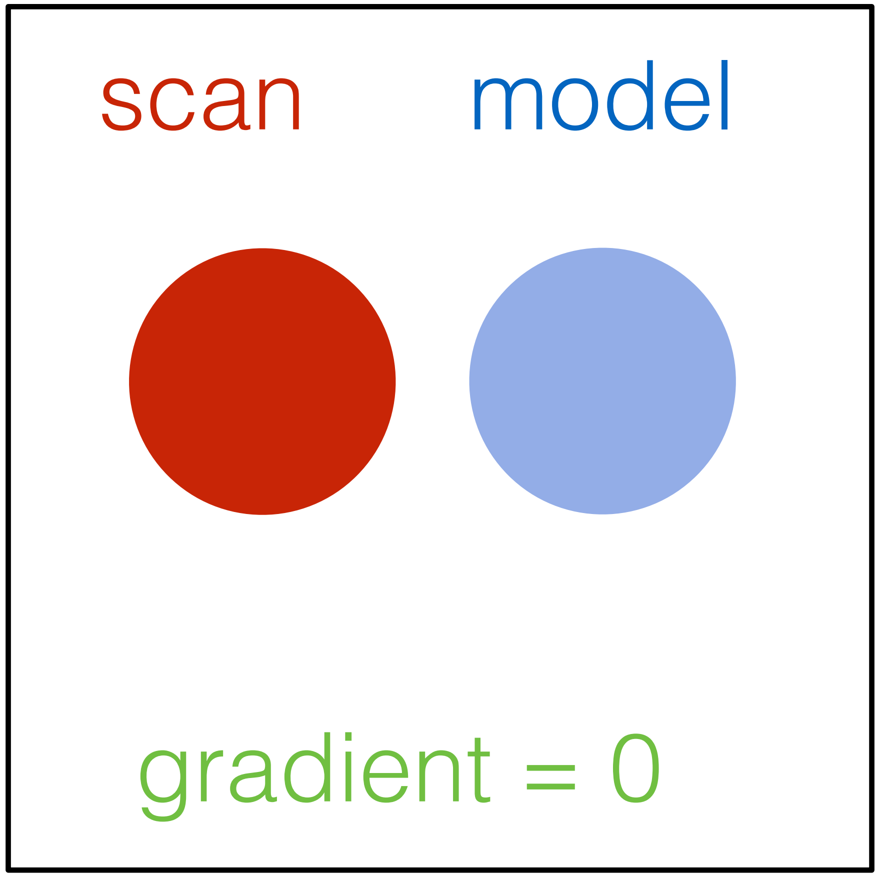
```
import chumpy as ch
import cv2
from opendr.camera import ProjectPoints
from opendr.renderers import TexturedRenderer

# Load meshes, create other objectives…
# …

# Construct renderer
rn = TexturedRenderer()
rn.camera = ProjectPoints(v=m.v, vc=m.vc, rt=ch.zeros(3), t=ch.zeros(3),
             f=ch.array([w,w])/2., c = ch.array([w,h])/2., k=ch.zeros(5))
rn.frustum = {'near': 1., 'far': 10., 'width': w, 'height': h}
rn.set(f=m.f, texture_image=m.texture_img, ft=m.ft, vt=m.vt, bgcolor=ch.zeros(3))

# Define the error term
obj = rn – cv2.imread(real_img_path)

# Minimize
ch.minimize(obj, x0=[m.v], method=‘dogleg’)
```

lighting encoded in vc
appearance encoded in

Texture-based registration

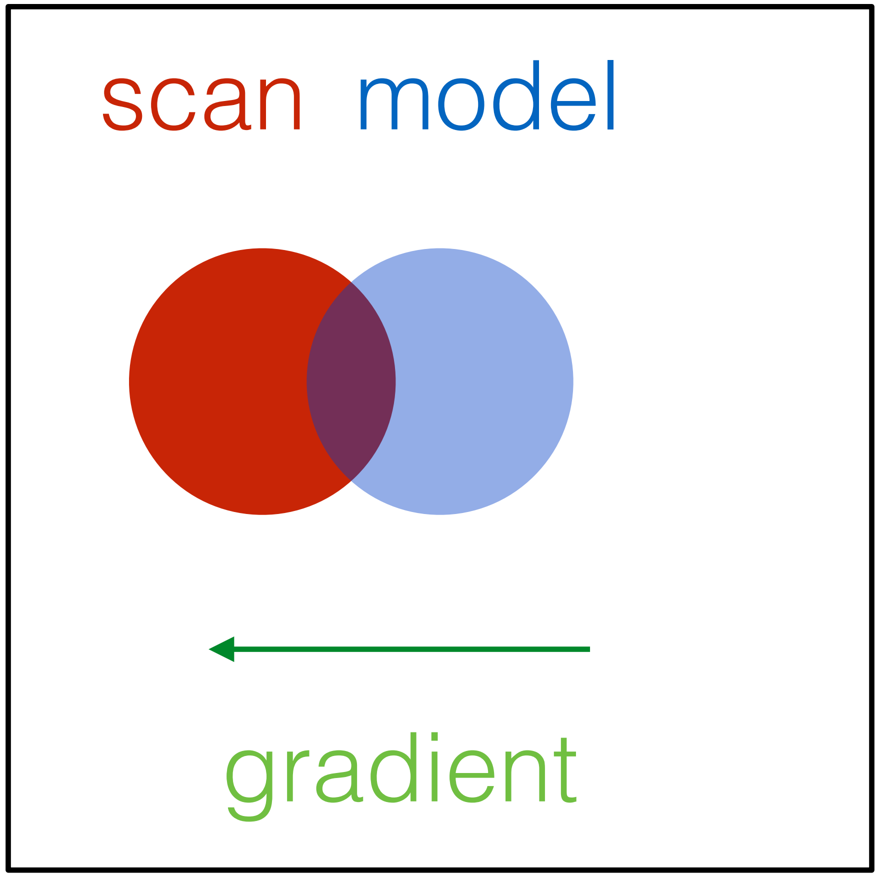
- The appearance objective function has MANY local minima

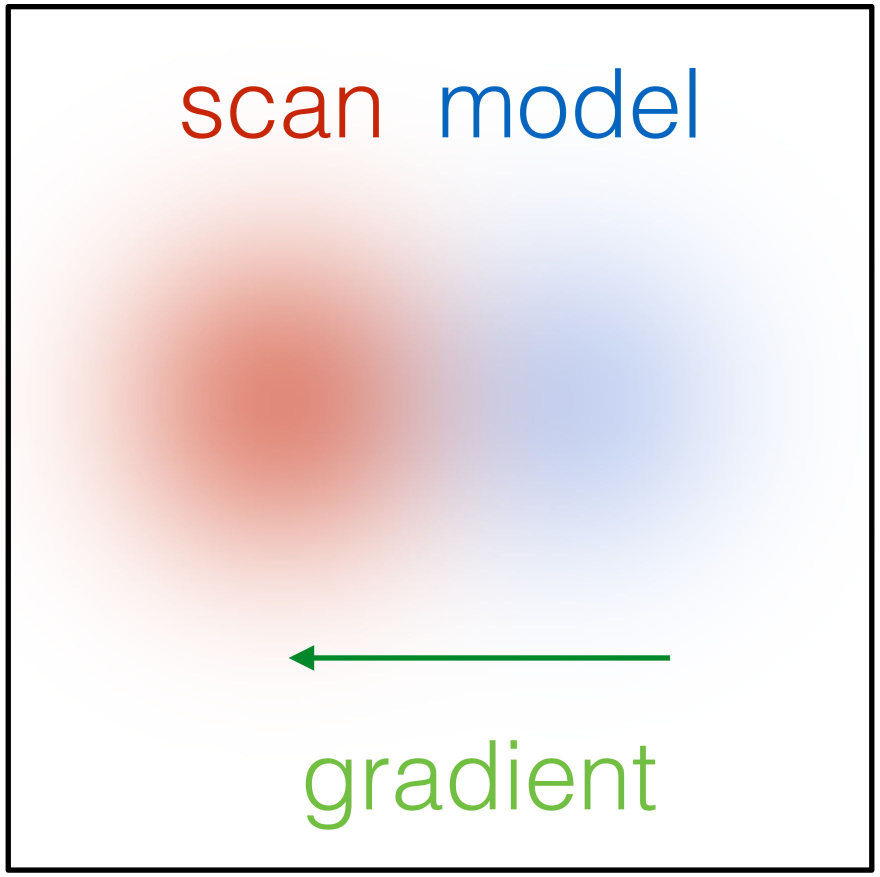




Texture-based registration

- The appearance objective function has MANY local minima
  - Pyramids of blurred images help





Texture-based registration

- The appearance objective function has MANY local minima
  - Pyramids of blurred images help
- The dimensionality of this objective is much bigger than the geometric one
  - Optimisation will be slower

Texture-based registration

- The appearance objective function has MANY local minima
  - Pyramids of blurred images help
- The dimensionality of this objective is much bigger than the geometric one
  - Optimisation will be slower
- Open problems: Lighting optimisation? Occlusions?

Take-home message

- Optimising SMPL pose and shape with chumpy is easy
  - But the devil is in the details: point2surface, regularisers
- We can add color to our model either with per-vertex colors, or texture maps
- Apart from making the model match the scan geometrically, we can make it match in terms of COLOR
- OpenDR differentiates the rendering process for us