Body Models II

Javier Romero
Max Planck Institute for Intelligent Systems
Perceiving Systems

May 24, 2016

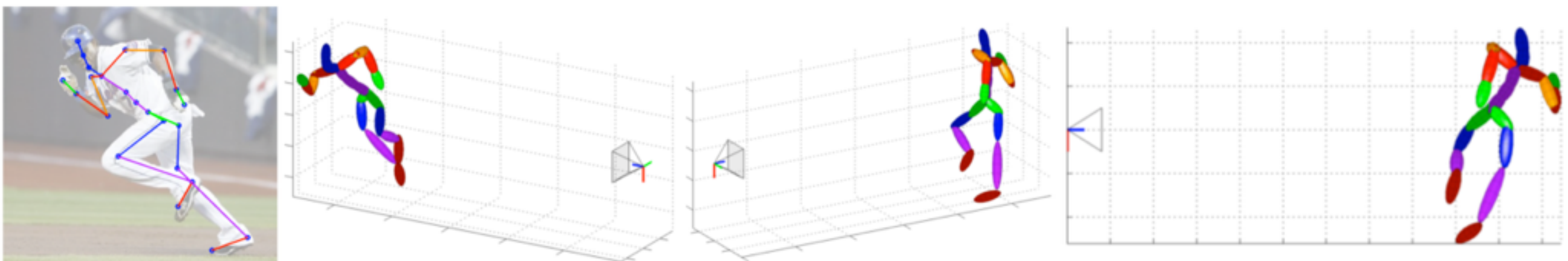
MAX-PLANCK-GESELLSCHAFT

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?

- Holistic vs part-based models
- Translations and rotations as basic building blocks
- Procrustes: algorithm for computing optimal similarity (+mirroring) transformation between two point sets

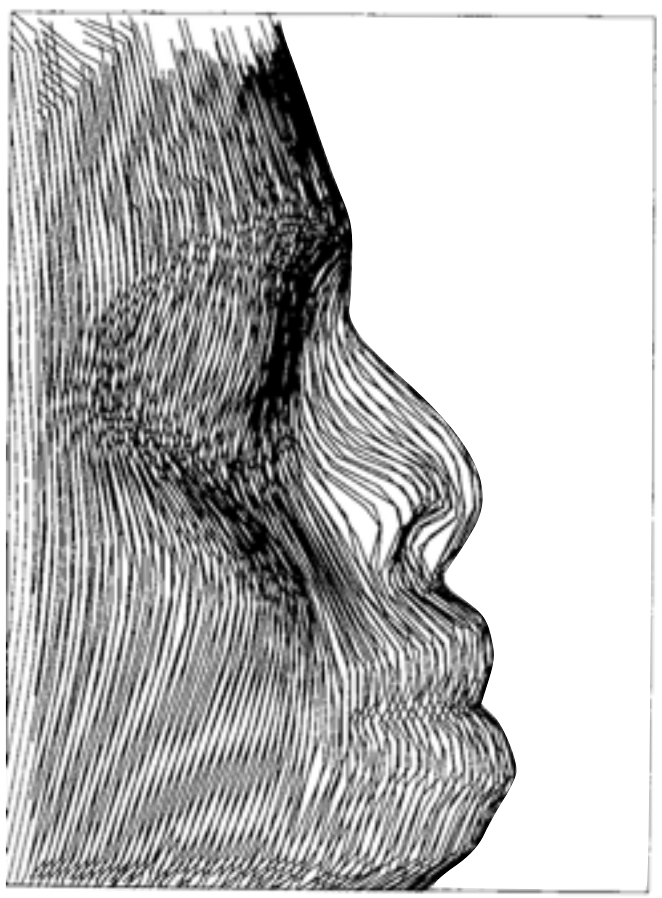
Why are we doing this?

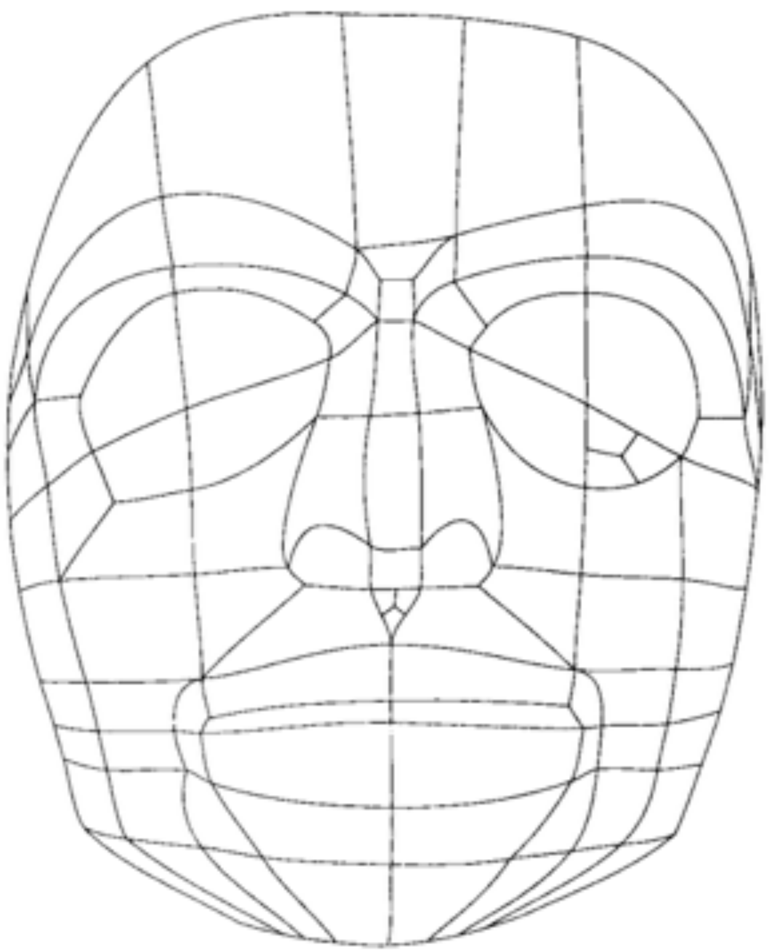


Solve for the camera parameters!

Reconstructing 3D Human Pose from 2D Image Landmarks, Ramakrishna et al.

Why are we doing this?

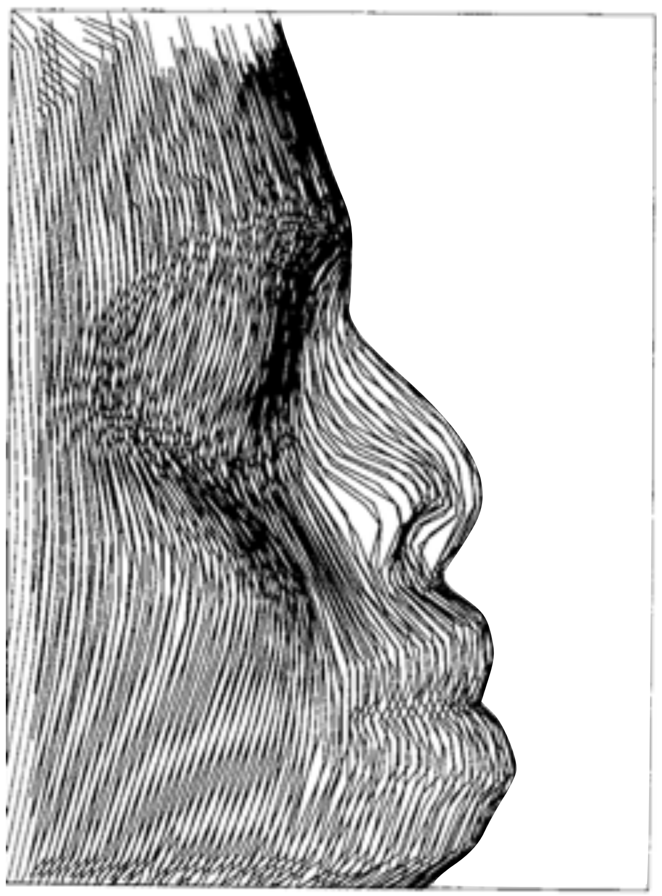


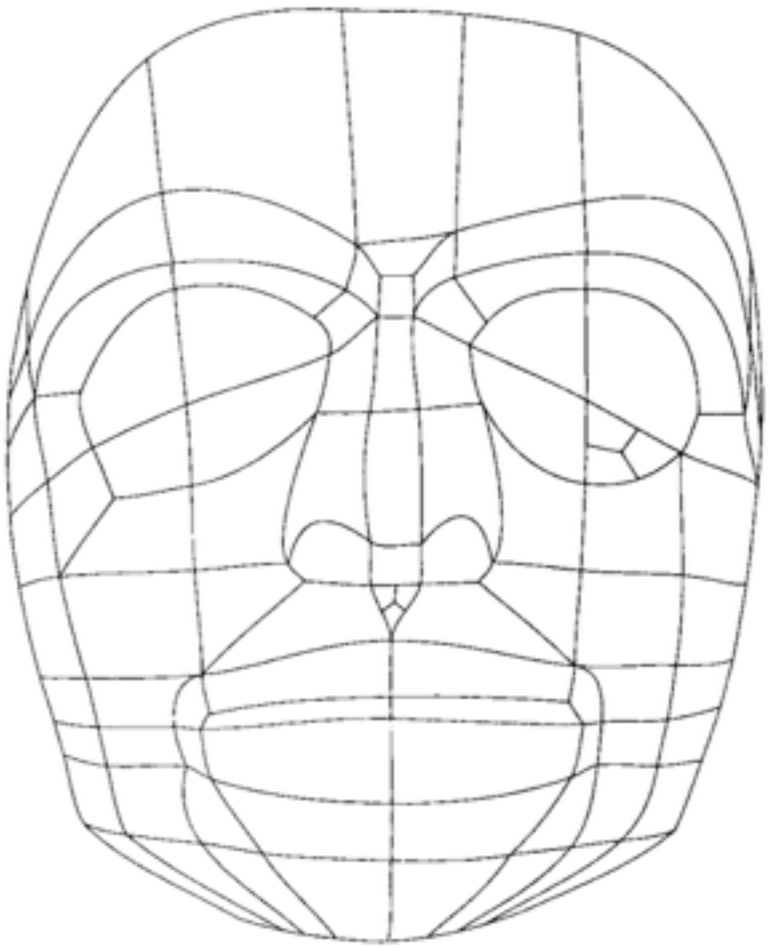


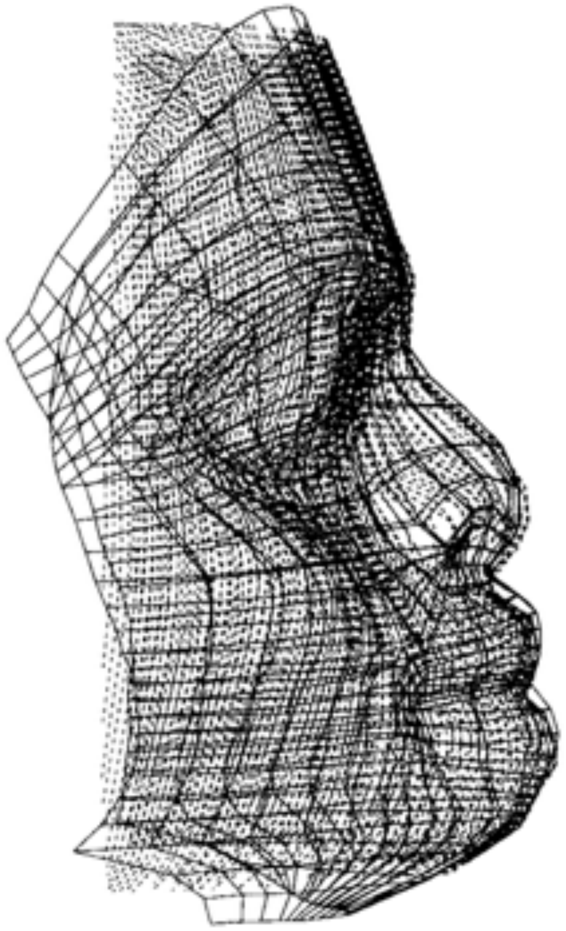
Transfer information from one point set to another

A Method for Registration of 3D Shapes, Besl and McKay

Why are we doing this?



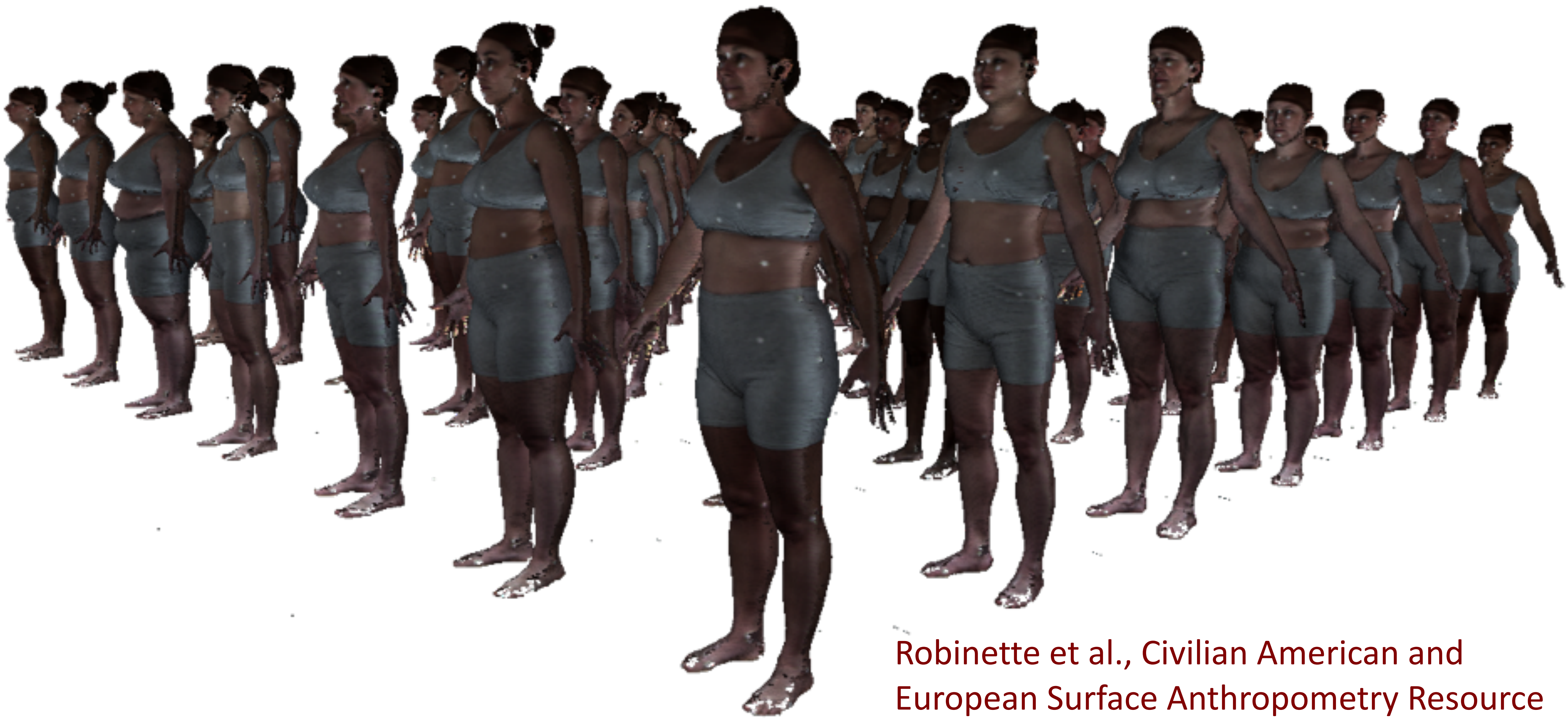




Transfer information from one point set to another

A Method for Registration of 3D Shapes, Besl and McKay

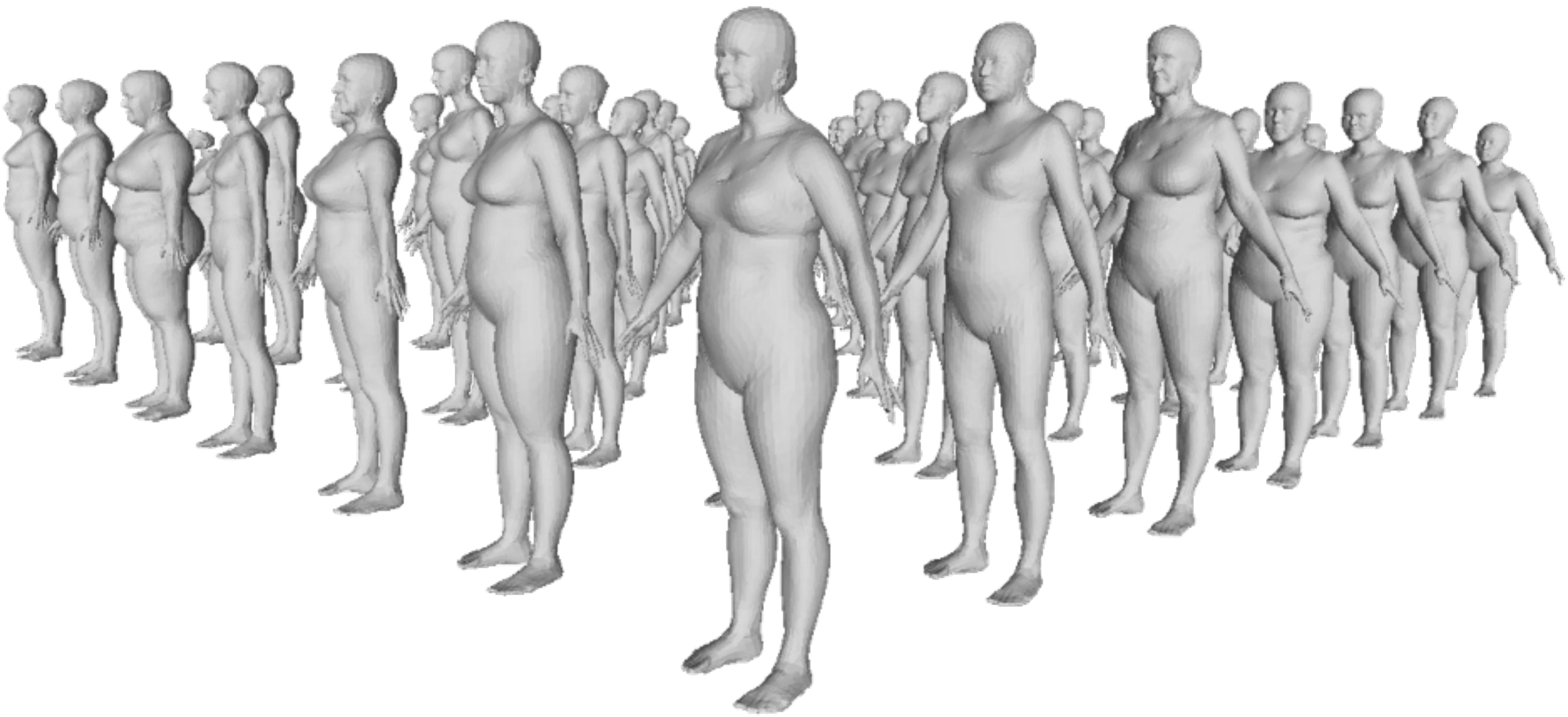
Why are we doing this?



Robinette et al., Civilian American and European Surface Anthropometry Resource (CAESAR) final report, 2002.

Extract common information across point sets

Why are we doing this?



Extract common information across point sets

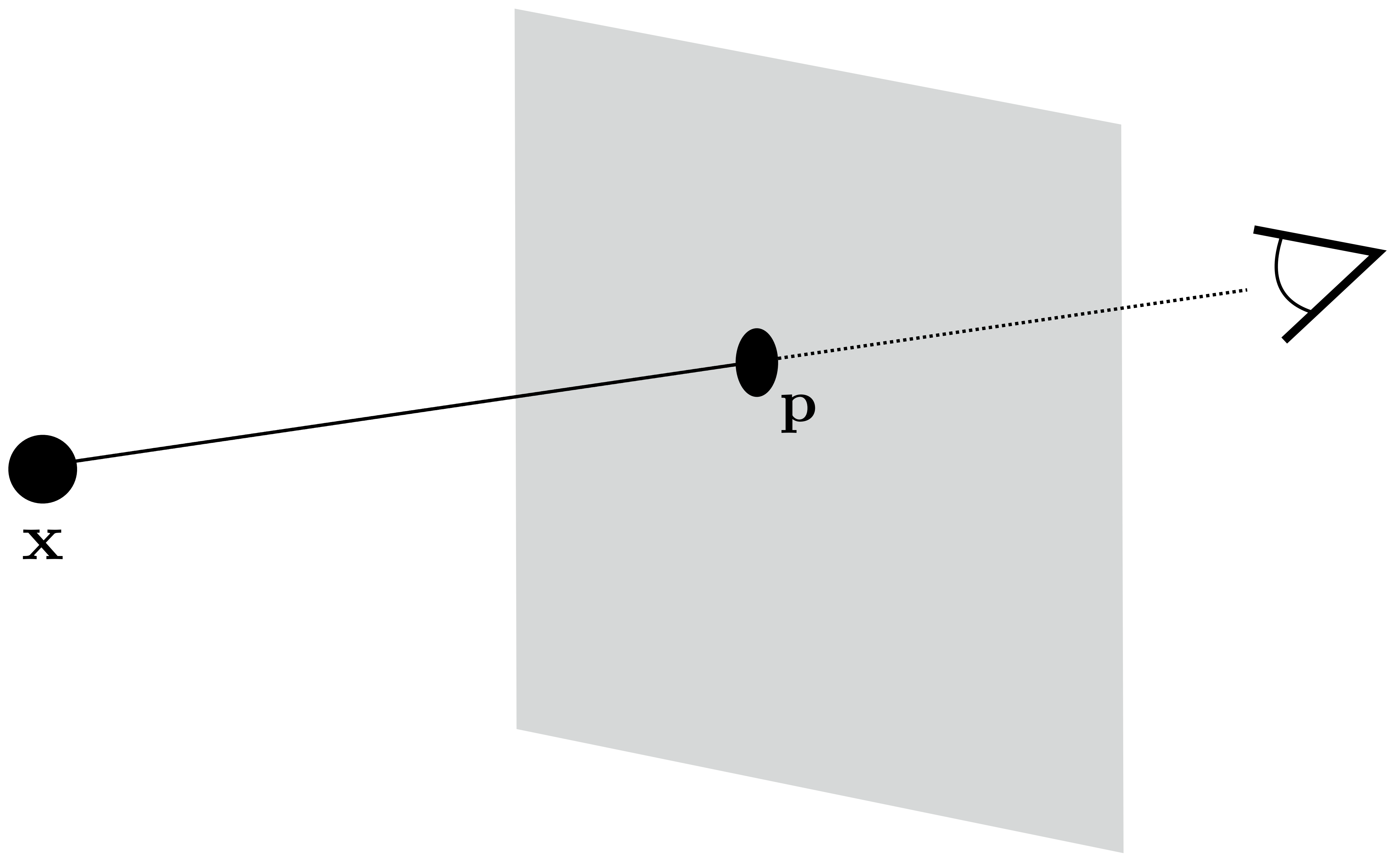
What is missing

- How do we map 3D to 2D?
- If we don't have information about those shapes, how do we find correspondences?
- Rigid deformations do not work, what do we do?
    - next week, a complete articulated body model

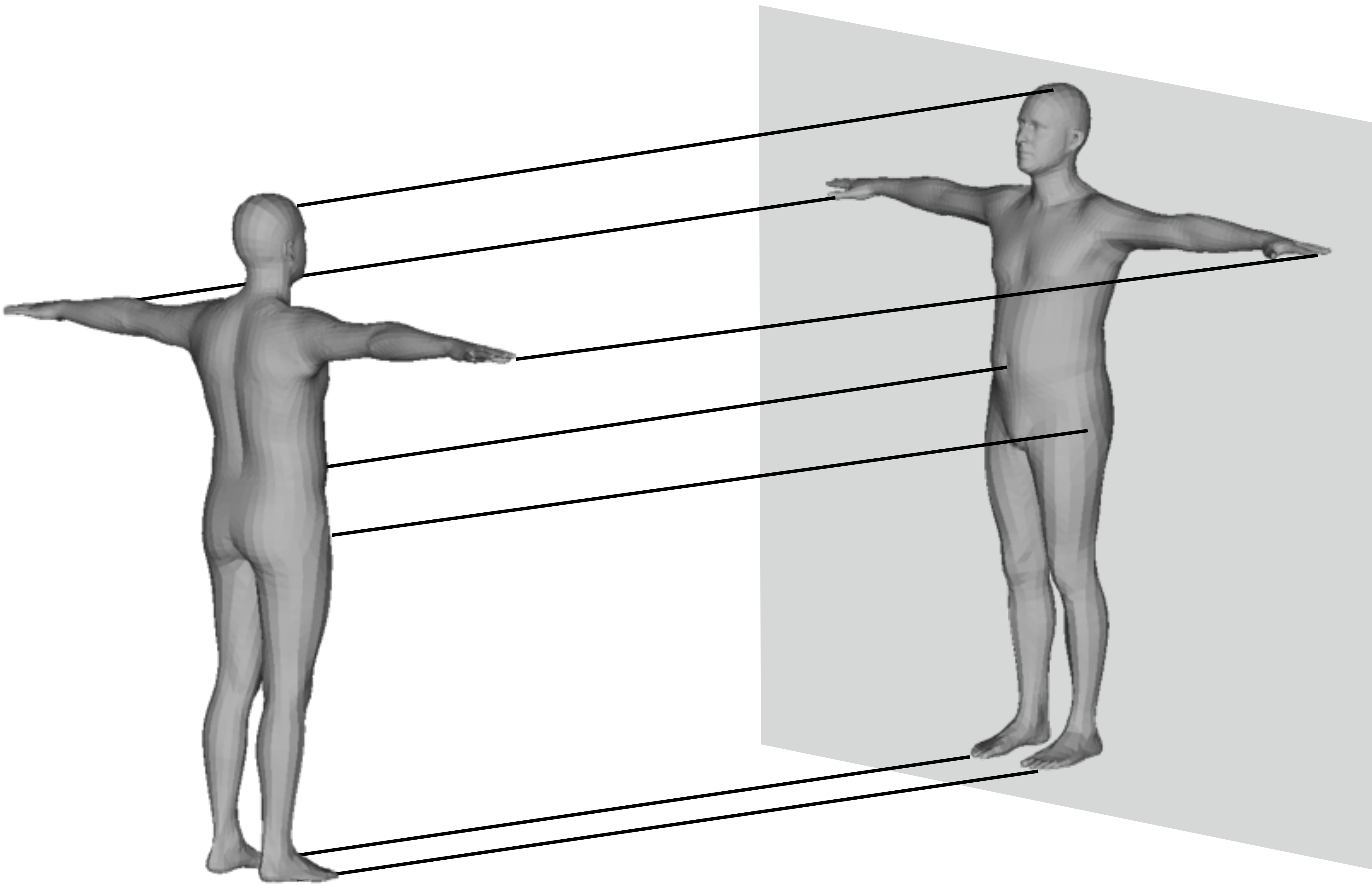
Today

- Mapping the 3D world to 2D: camera models
- Optimise rigid 3D -> 2D correspondences
- Optimising alignment and correspondences: ICP
- Alignment through gradient descent

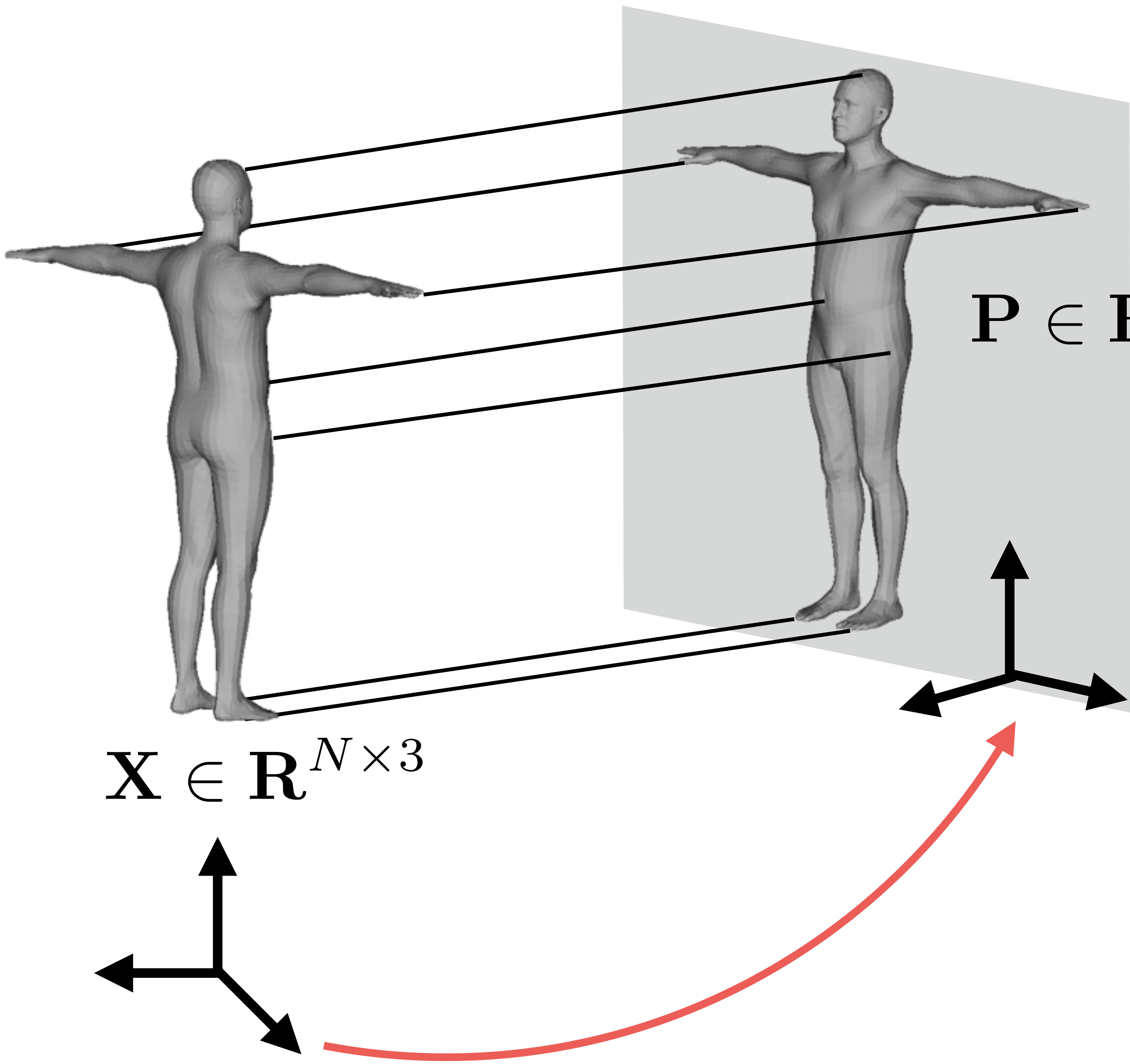
Mapping 3D to 2D: what is an image?



Orthographic projection

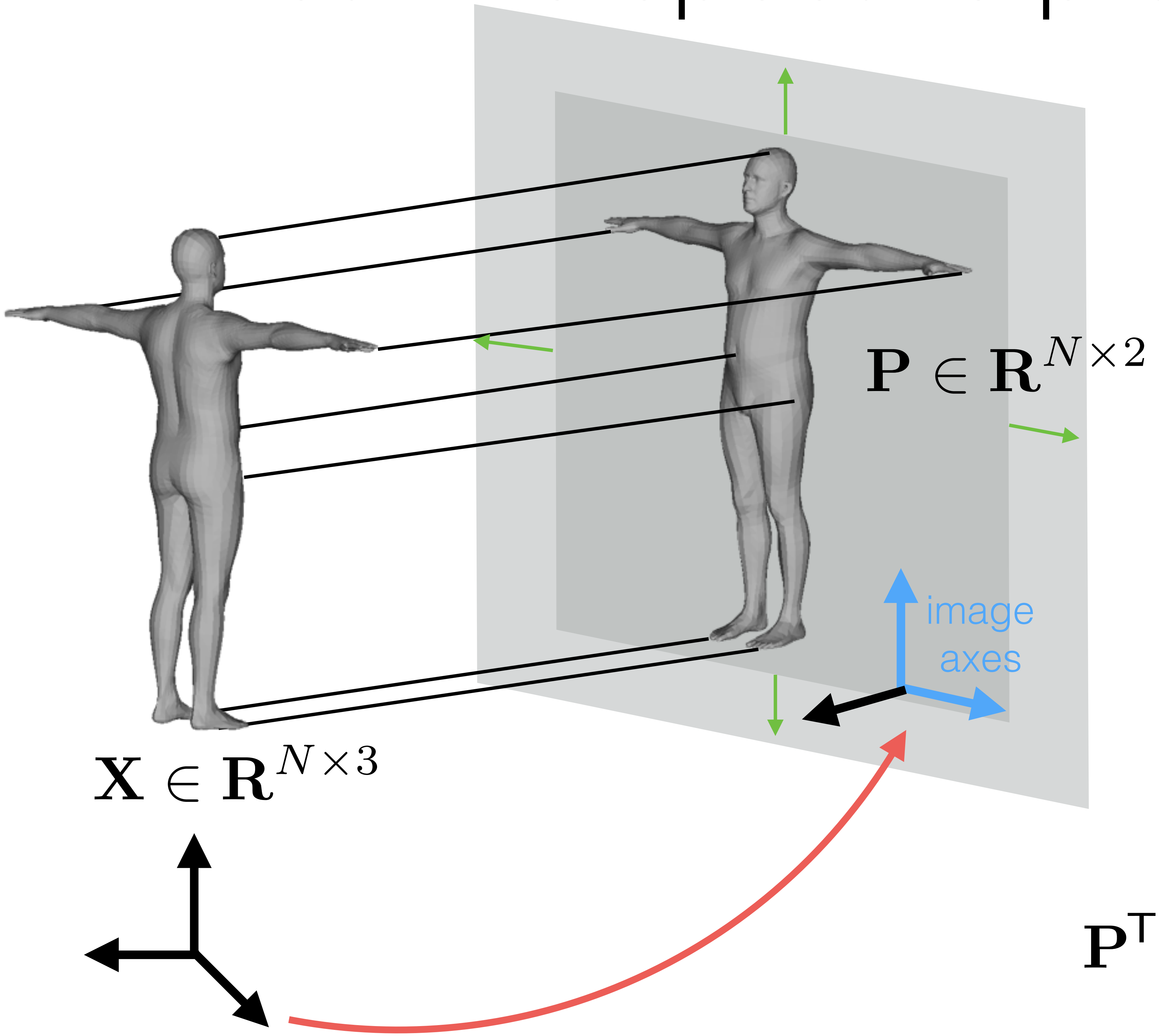


Orthographic projection



$$\mathbf{P}^{\mathsf{T}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{R}\mathbf{X}^{\mathsf{T}}$$

Weak Perspective projection



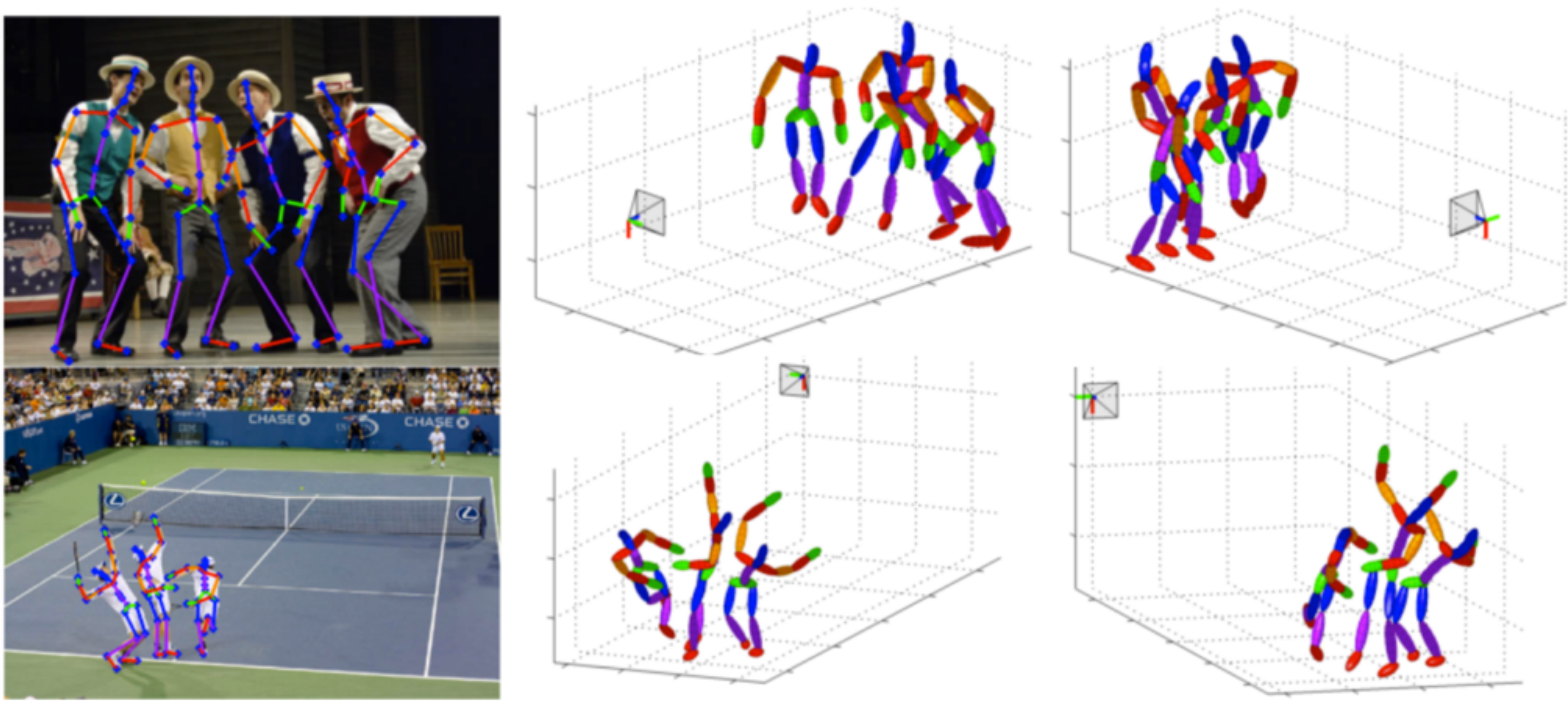
$$\mathbf{P}^{\mathsf{T}} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \end{bmatrix} \mathbf{R} \mathbf{X}^{\mathsf{T}}$$

2D-3D Procrustes with weak perspective

- Weak perspective: depth of 3D points has no effect on projection
    - Procrustes: pad the projections with a column of zeros and solve for the 3D-3D procrustes problem!
- The case of anisotropic scale (different scale for x and y) complicates the rotation optimisation
    - See [1] for more information

[1]: Procrustes Problems, Gower and Dijksterhuis, Chapter 8

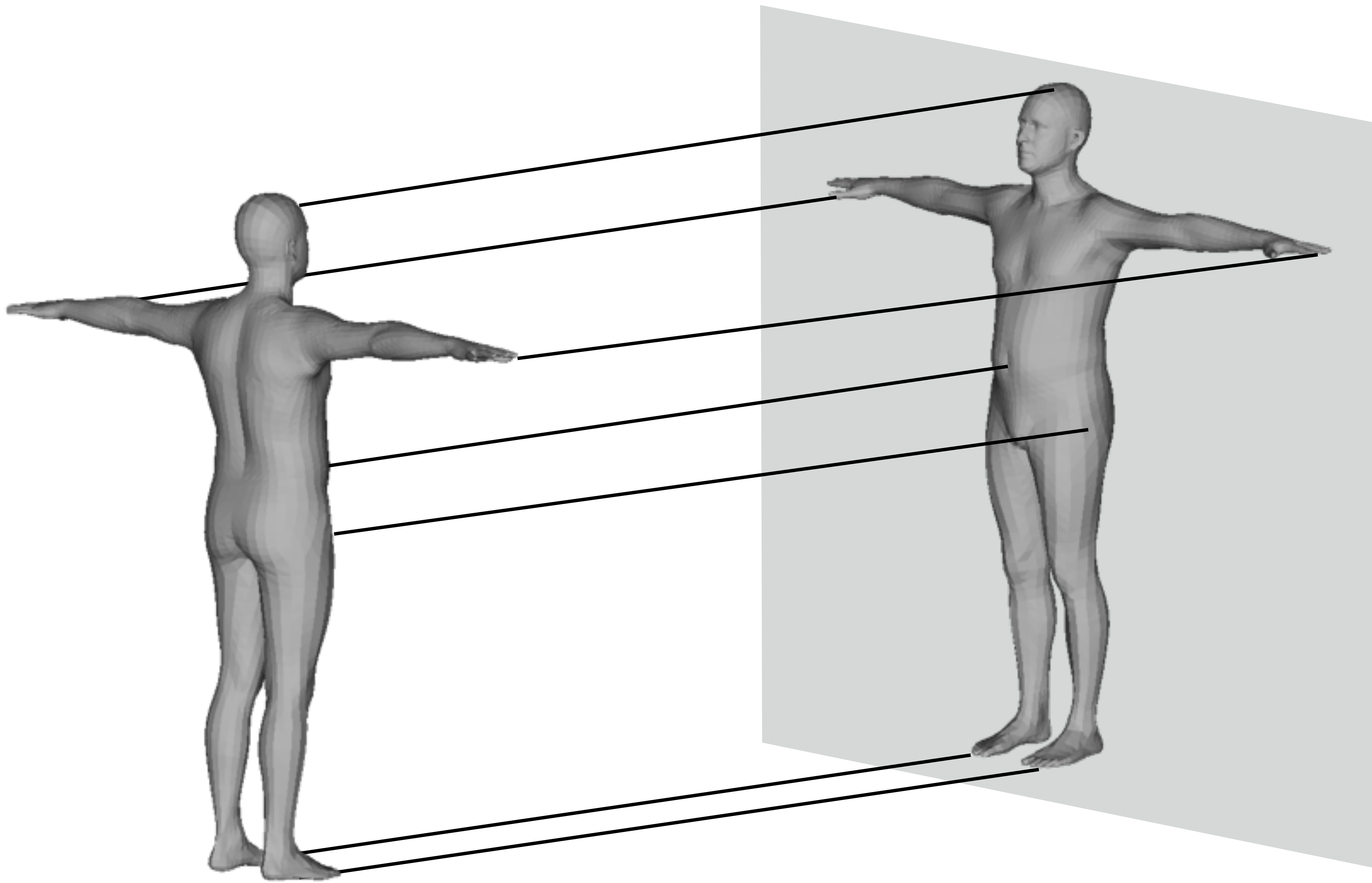
2D-3D Procrustes with weak perspective



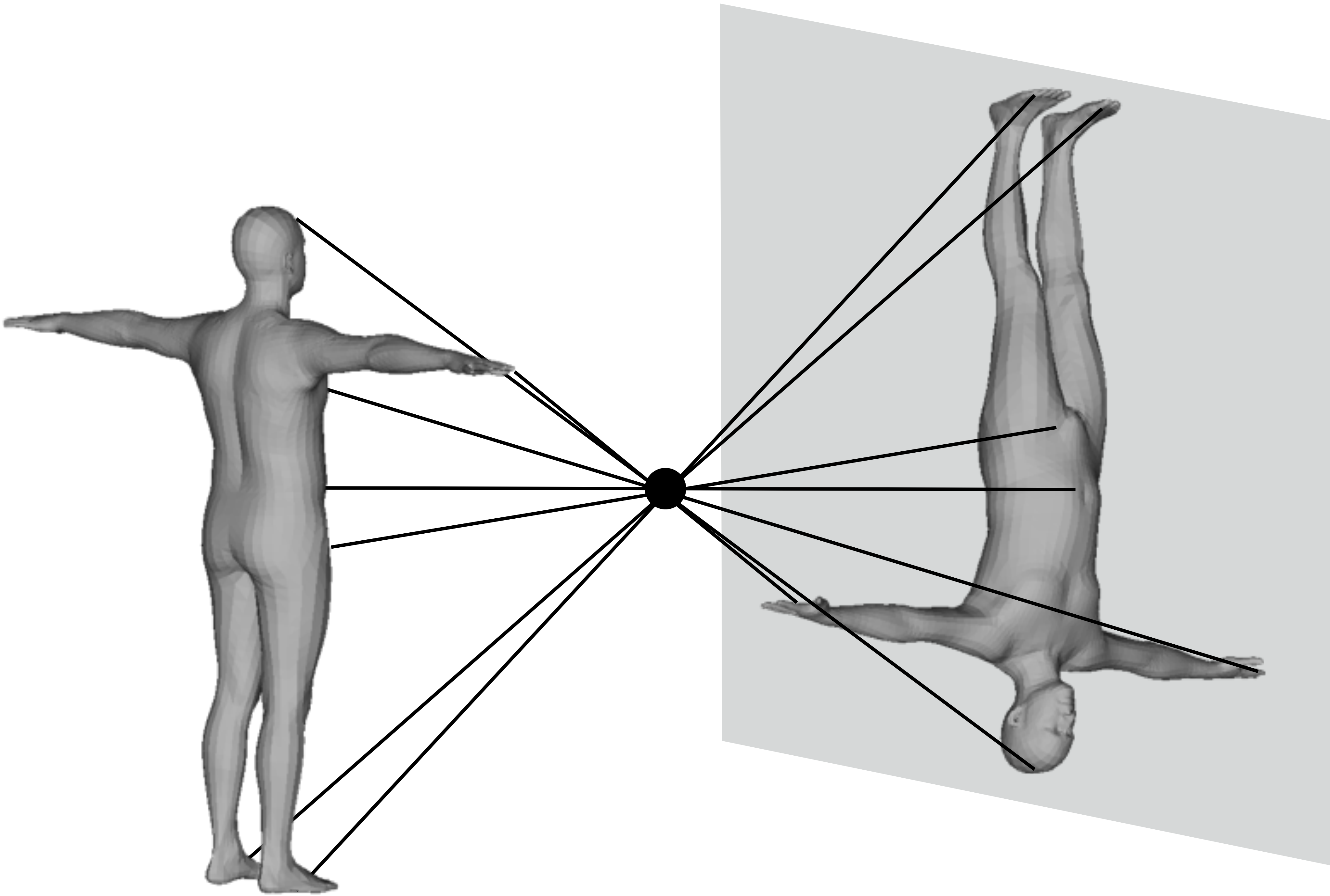
Application: computing camera parameters for inferred 3D poses from 2D

Reconstructing 3D Human Pose from 2D Image Landmarks, Ramakrishna et al.

Is that what happens in reality?



Projective geometry



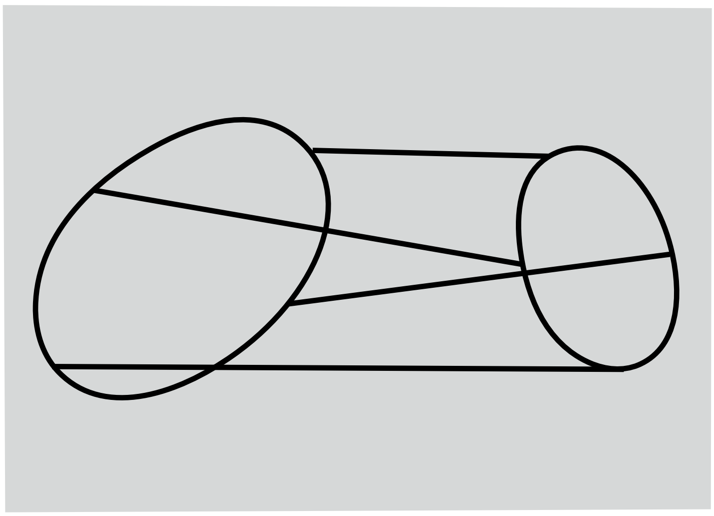
Procrustes with projective camera?

- no closed form solution
  - good initialisations (e.g. procrustes, DLT) + non-linear optimisation

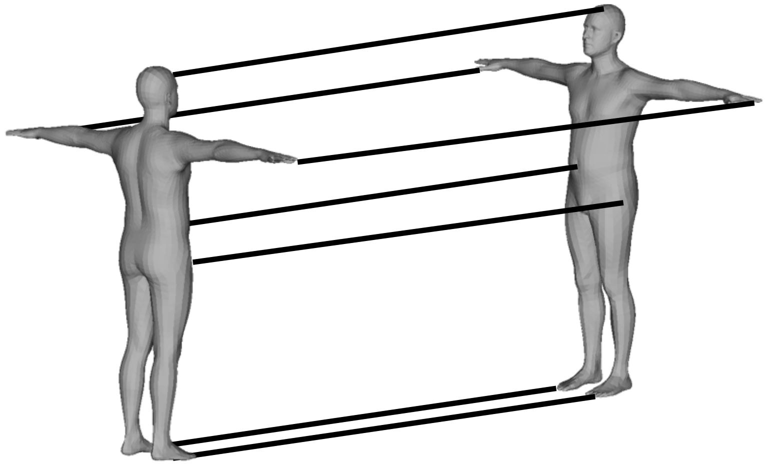
Procrustes with projective camera?

- no closed form solution
  - good initialisations (e.g. procrustes, DLT) + non-linear optimisation
  - No need to worry, this is not the worst

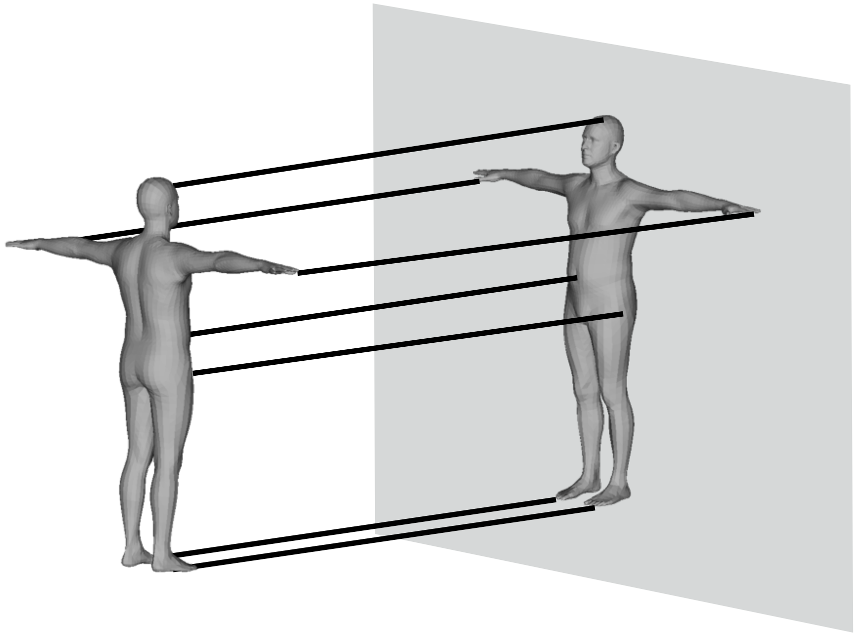
Recap



2D-2D



3D-3D



3D-2D

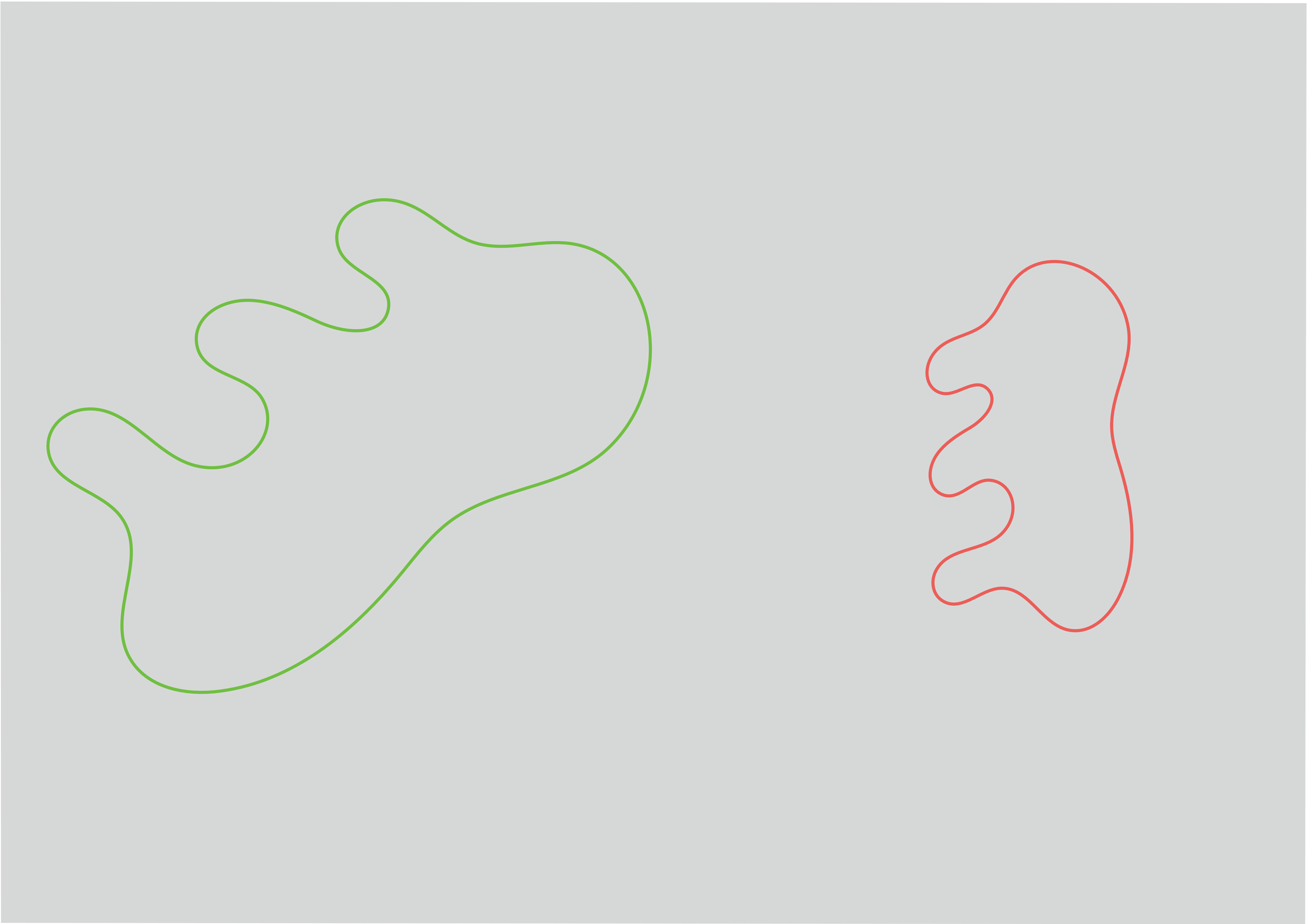
Recap: Correspondences?

- Given correspondences, we know how to find the optimal “similarity” transformation

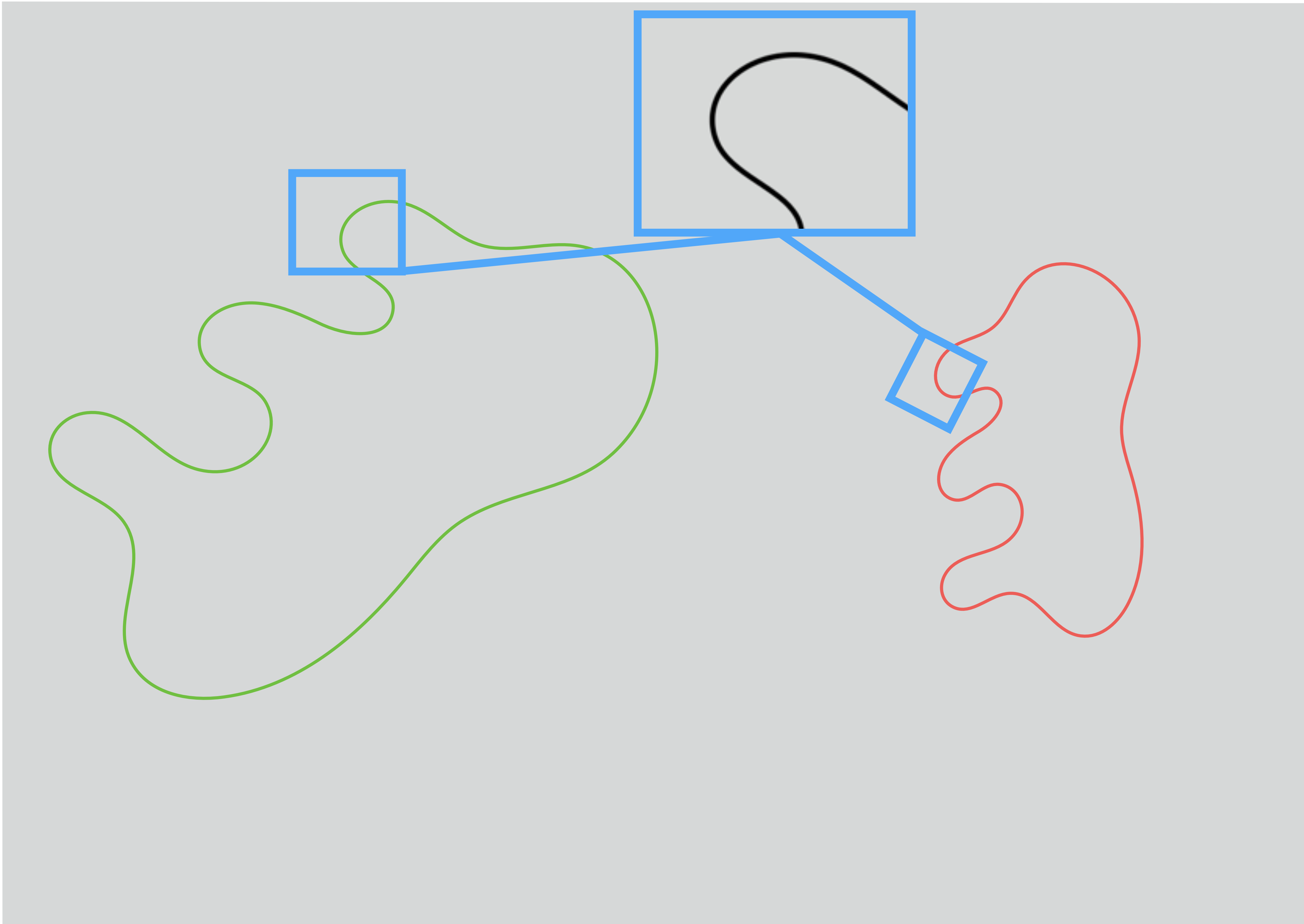
- But who is giving us the correspondences?

- If the correspondences are not optimal, is there anything better than the procrustes “step”?

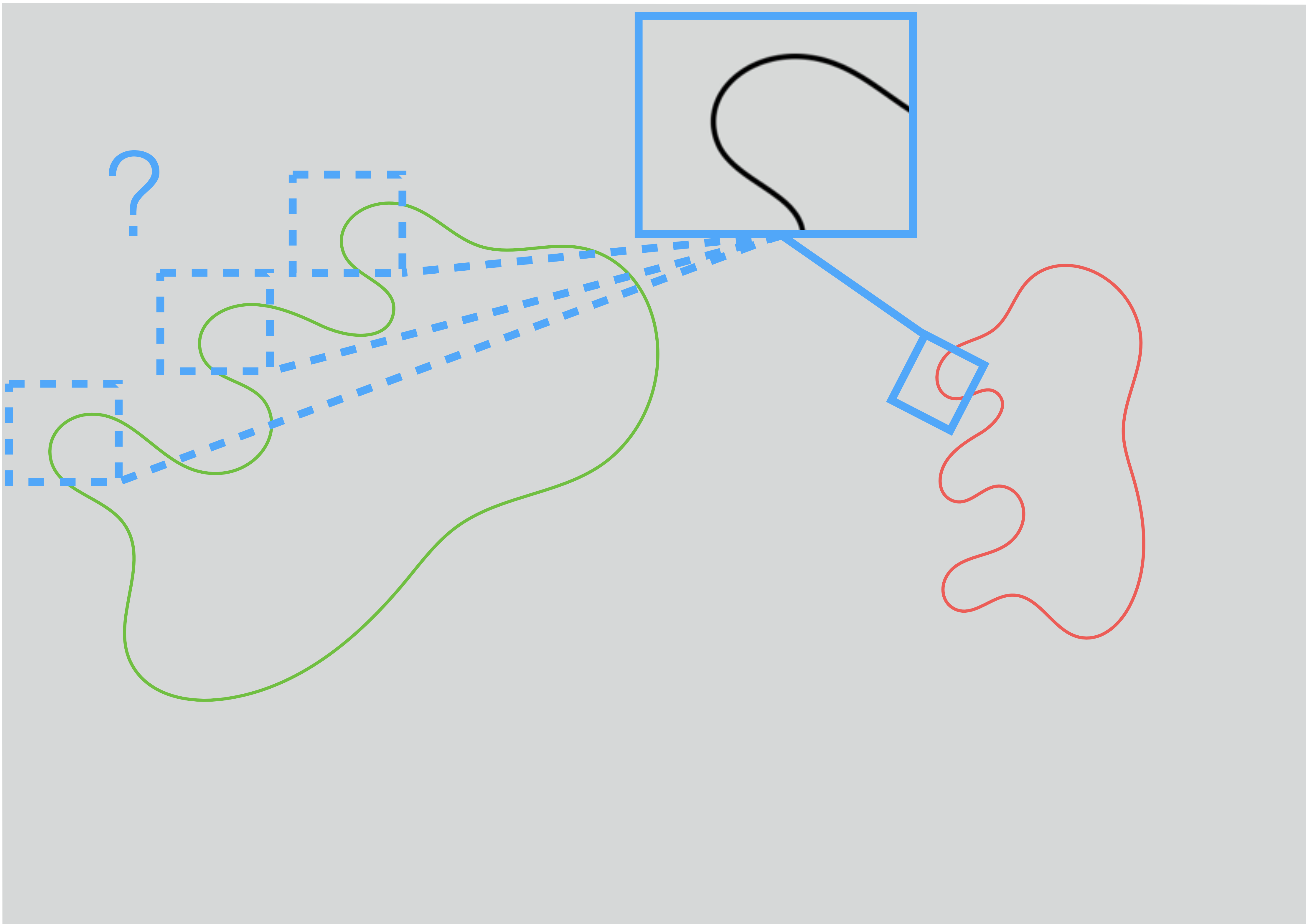
Ideas?



Ideas?



Ideas?



Ideas

- The idea was to minimise the sum of distances between the one set of points and the other set, transformed

$$E \equiv \sum_i \|s\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i\|^2 \equiv \sum_i \|f(\mathbf{x}_i) \overset{?}{-} \mathbf{y}_i\|^2$$

compact notation: f contains translation, rotation and isotropic scale

Ideas

- The idea was to minimise the sum of distances between the one set of points and the other set, transformed

$$E \equiv \sum_i \|s\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i\|^2 \equiv \sum_i \|f(\mathbf{x}_{\boxed{i}}) - \mathbf{y}_{\boxed{i}}\|^2$$

?

compact notation: f contains translation, rotation and isotropic scale

- What if we make up some reasonable correspondences?

Ideas

- The idea was to minimise the sum of distances between the one set of points and the other set, transformed

$$E \equiv \sum_i \|s\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i\|^2 \equiv \sum_i \|f(\mathbf{x}_i) - \mathbf{y}_i\|^2$$

?

compact notation: f contains translation, rotation and isotropic scale

- What if we make up some reasonable correspondences?

iteration

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

original unsorted points

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

Ideas

- The idea was to minimise the sum of distances between the one set of points and the other set, transformed

$$E \equiv \sum_i \|s\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i\|^2 \equiv \sum_i \|f(\mathbf{x}_i) \overset{?}{-} \mathbf{y}_i\|^2$$

compact notation: f contains translation, rotation and isotropic scale

- What if we make up some reasonable correspondences?

iteration

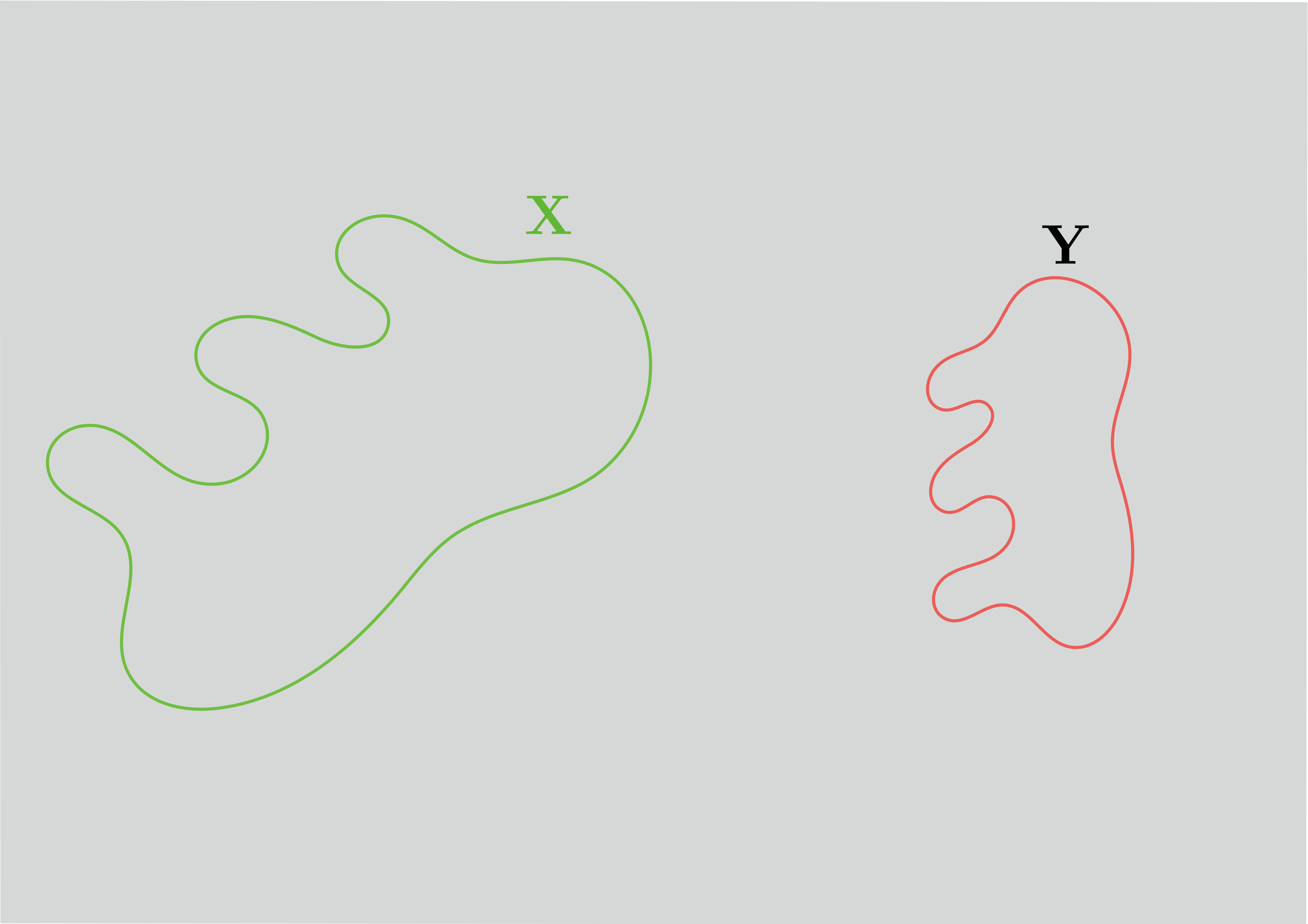
$$\mathbf{x}_i^{j+1} = \arg\min_{\mathbf{x}\in\mathbf{X}} \|f^j(\mathbf{x}) - \mathbf{y}_i\|^2$$

Given current best transformation, which are the closest correspondences?

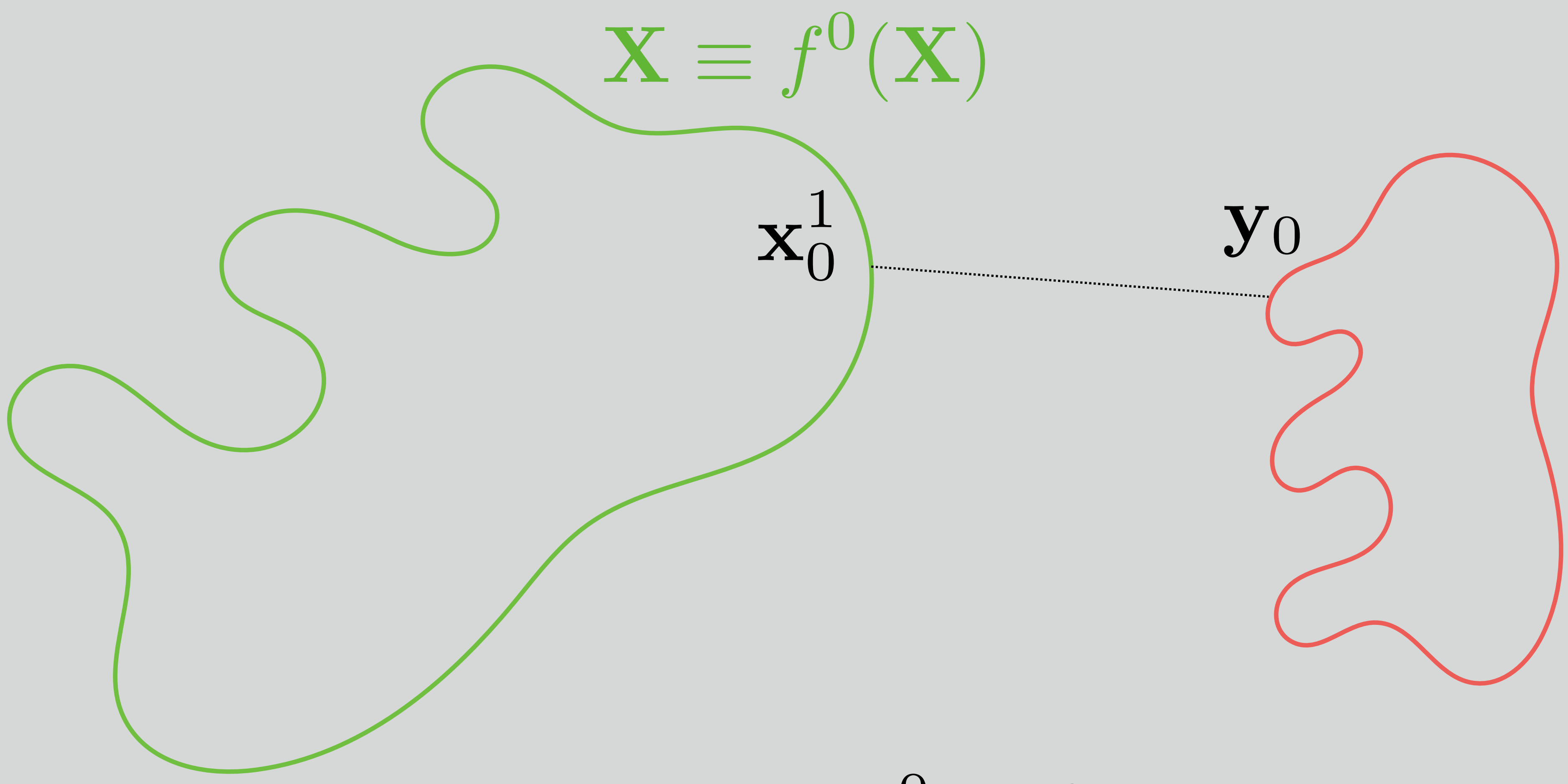
$$f^{j+1} = \arg\min_f \sum_i \|f(\mathbf{x}_i^{j+1}) - \mathbf{y}_i\|^2$$

Given current best correspondences, which is the best transformation?

Make up reasonable correspondences



Make up reasonable correspondences

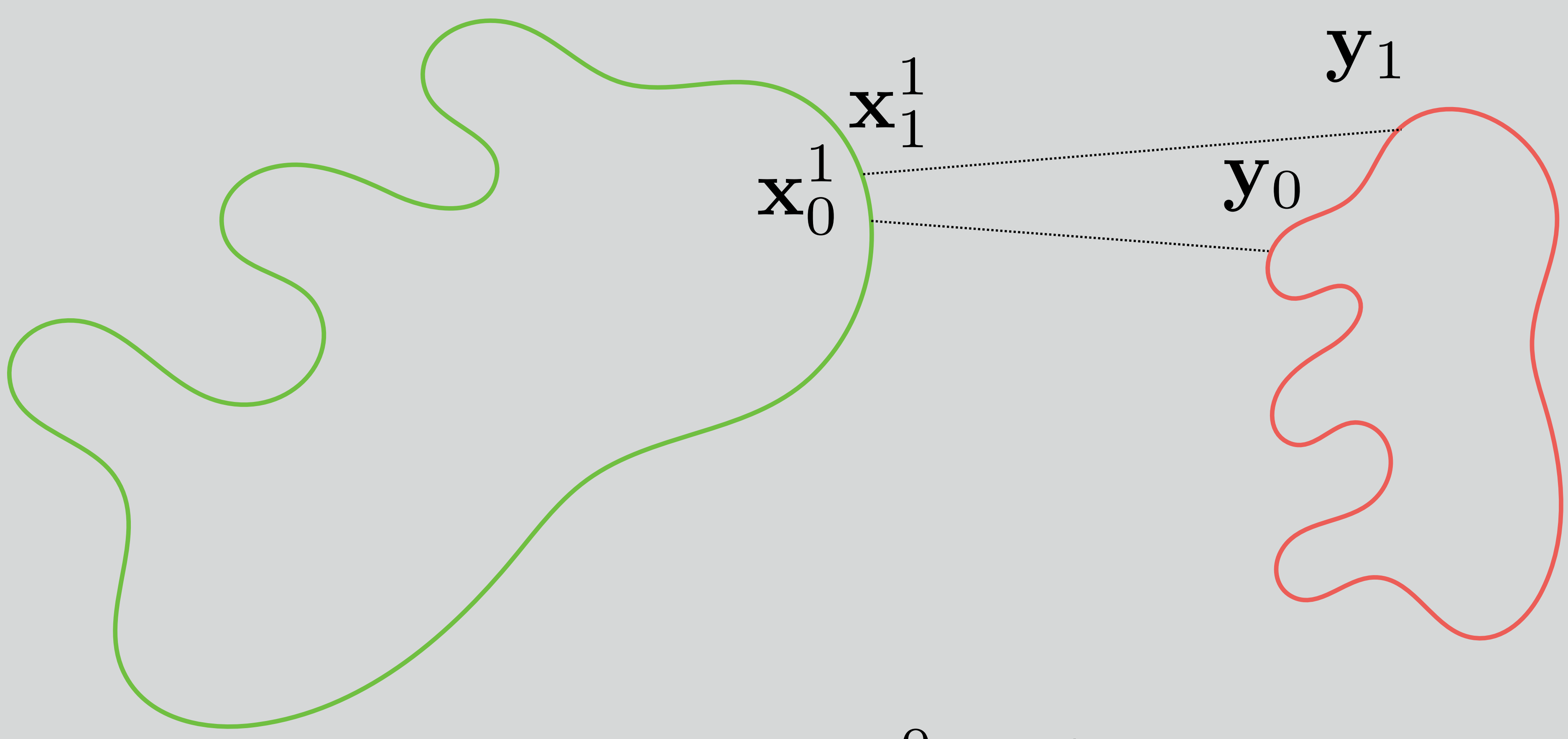


Neutral initialisation.
Initialising t to align centroids should work better!

$$f^0 = \{\mathbf{R} = \mathbf{I}, \mathbf{t} = \mathbf{0}, s = 1\}$$

$$\mathbf{x}_0^1 = \arg\min_{\mathbf{x} \in \mathbf{X}} \| f^0(\mathbf{x}) - \mathbf{y}_0 \|^2$$

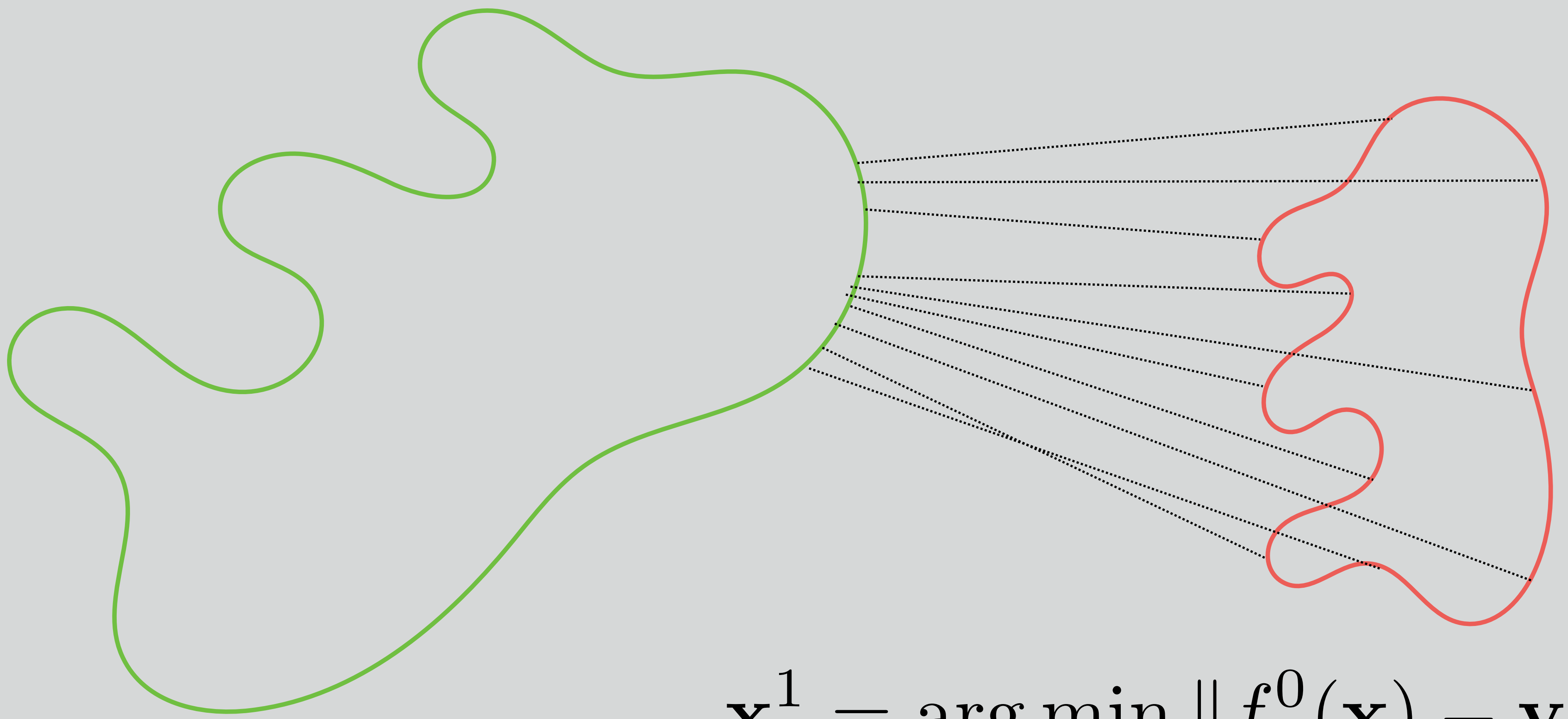
Make up reasonable correspondences



$$f^0 = \{\mathbf{R} = \mathbf{I}, \mathbf{t} = \mathbf{0}, s = 1\}$$

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

Solve for the best transformation

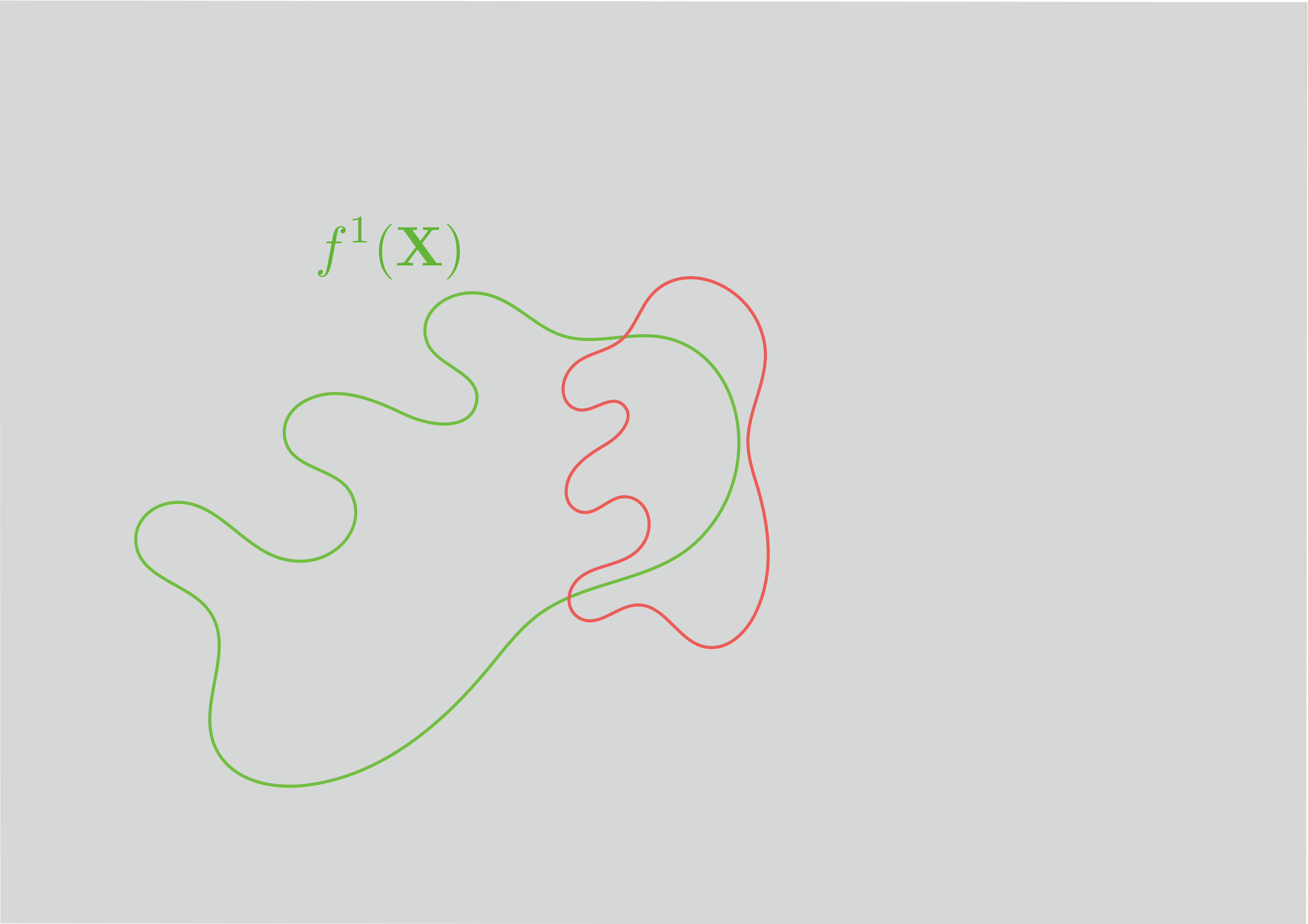


$$\mathbf{x}_i^1 = \arg\min_{\mathbf{x} \in \mathbf{X}} \| f^0(\mathbf{x}) - \mathbf{y}_i \|^2$$

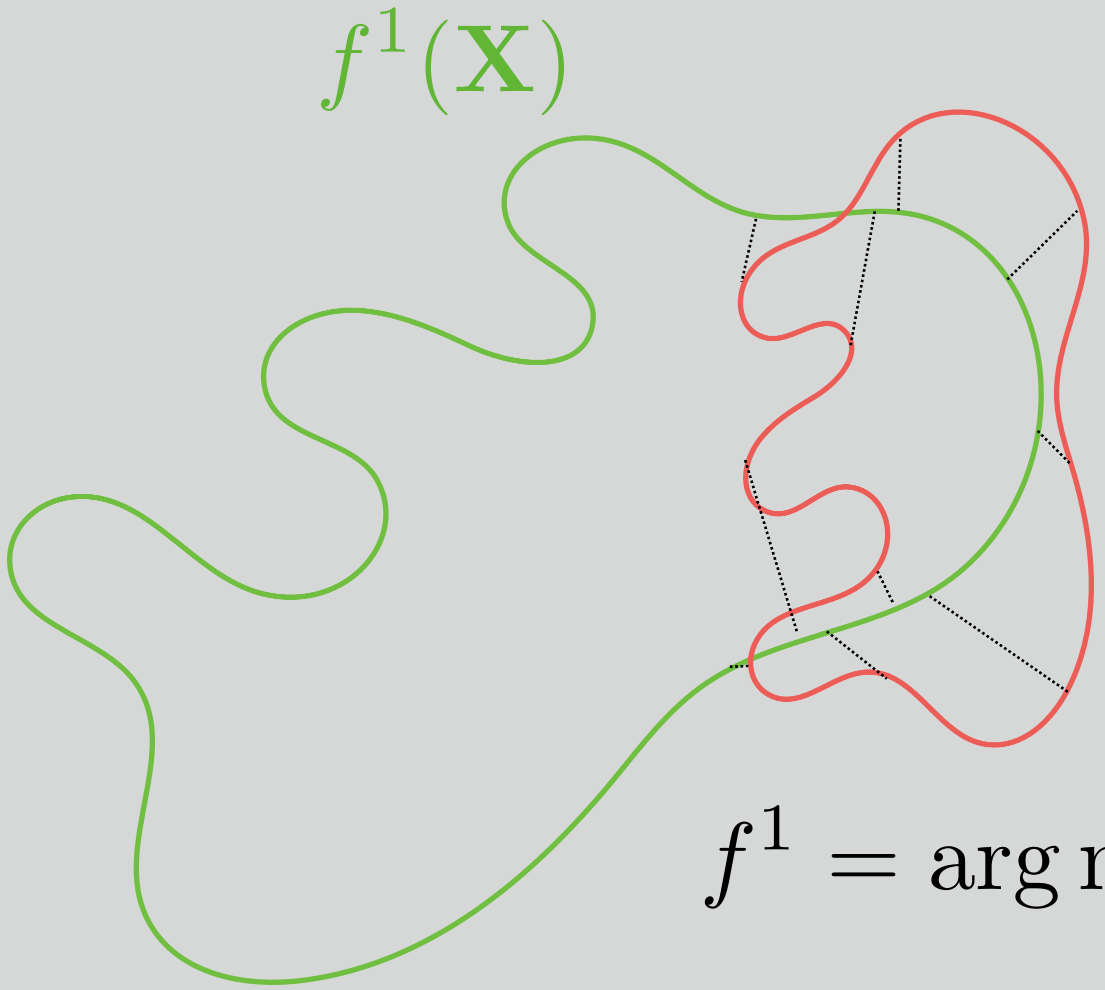
solve with procrustes

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

Apply it …



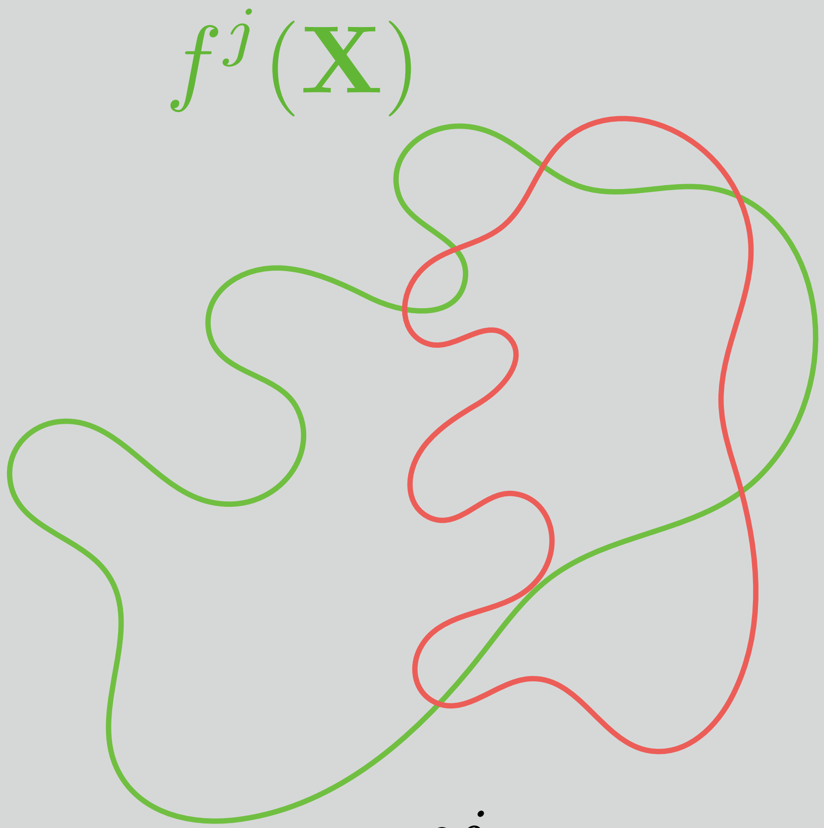
and iterate!



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

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

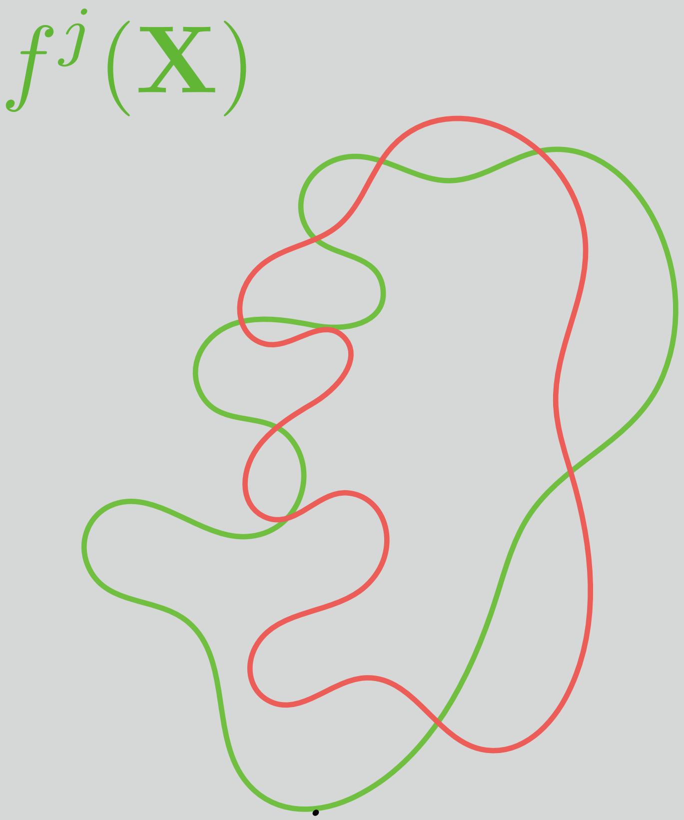
and iterate!



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

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

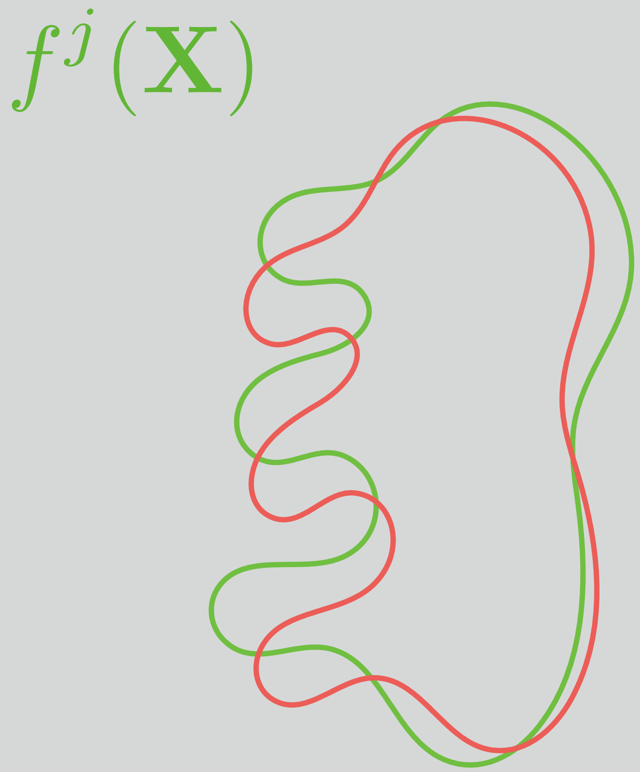
and iterate!



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

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

and iterate!



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

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

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!

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

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

Iterative Closest Point (ICP)

typically better than 0

1. initialise

$$f^0 = \{\mathbf{R} = \mathbf{I}, \mathbf{t} = \boxed{\frac{\sum \mathbf{y}_i}{N} - \frac{\sum \mathbf{x}_i}{N}}, s = 1\}$$

Iterative Closest Point (ICP)

1. initialise

$$f^0 = \{\mathbf{R} = \mathbf{I}, \mathbf{t} = \frac{\sum \mathbf{y}_i}{N} - \frac{\sum \mathbf{x}_i}{N}, s = 1\}$$

2. compute correspondences according to current best transform

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

Iterative Closest Point (ICP)

1. initialise

$$f^0 = \{\mathbf{R} = \mathbf{I}, \mathbf{t} = \frac{\sum \mathbf{y}_i}{N} - \frac{\sum \mathbf{x}_i}{N}, s = 1\}$$

2. compute correspondences according to current best transform

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

3. compute optimal transformation ($\mathbf{s}, \mathbf{R}, \mathbf{t}$)with Procrustes

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

Iterative Closest Point (ICP)

1. initialise $$f^0 = \{\mathbf{R} = \mathbf{I}, \mathbf{t} = \frac{\sum \mathbf{y}_i}{N} - \frac{\sum \mathbf{x}_i}{N}, s = 1\}$$
2. compute correspondences according to current best transform $$\mathbf{x}_i^{j+1} = \arg\min_{\mathbf{x} \in \mathbf{X}} \|f^j(\mathbf{x}) - \mathbf{y}_i\|^2$$
3. compute optimal transformation ($\mathbf{s}, \mathbf{R}, \mathbf{t}$)with Procrustes $$f^{j+1} = \arg\min_f \sum_i \|f(\mathbf{x}_i^{j+1}) - \mathbf{y}_i\|^2$$
4. terminate if converged (error below a threshold), otherwise iterate

Iterative Closest Point (ICP)

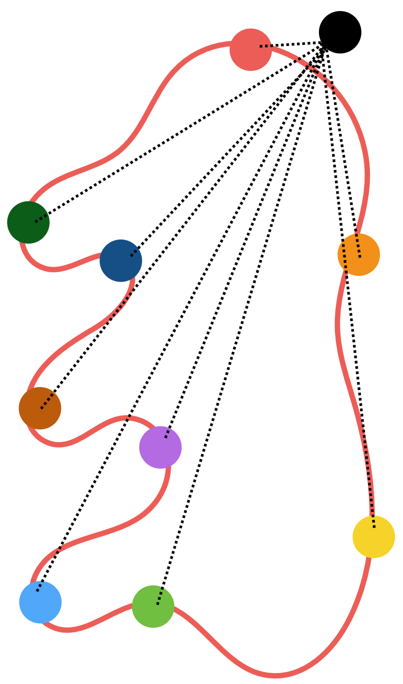
1. initialise $$f^0 = \{\mathbf{R} = \mathbf{I}, \mathbf{t} = \frac{\sum \mathbf{y}_i}{N} - \frac{\sum \mathbf{x}_i}{N}, s = 1\}$$
2. compute correspondences according to current best transform $$\mathbf{x}_i^{j+1} = \arg\min_{\mathbf{x} \in \mathbf{X}} \|f^j(\mathbf{x}) - \mathbf{y}_i\|^2$$
3. compute optimal transformation ($\mathbf{s}, \mathbf{R}, \mathbf{t}$)with Procrustes $$f^{j+1} = \arg\min_f \sum_i \|f(\mathbf{x}_i^{j+1}) - \mathbf{y}_i\|^2$$
4. terminate if converged (error below a threshold), otherwise iterate (go to step 2)
5. converges to local minima

Is ICP the best we can do?

- iteration j
- compute closest points
- compute optimal transformation with Procrustes
- apply transformation
- terminate if converged, otherwise iterate

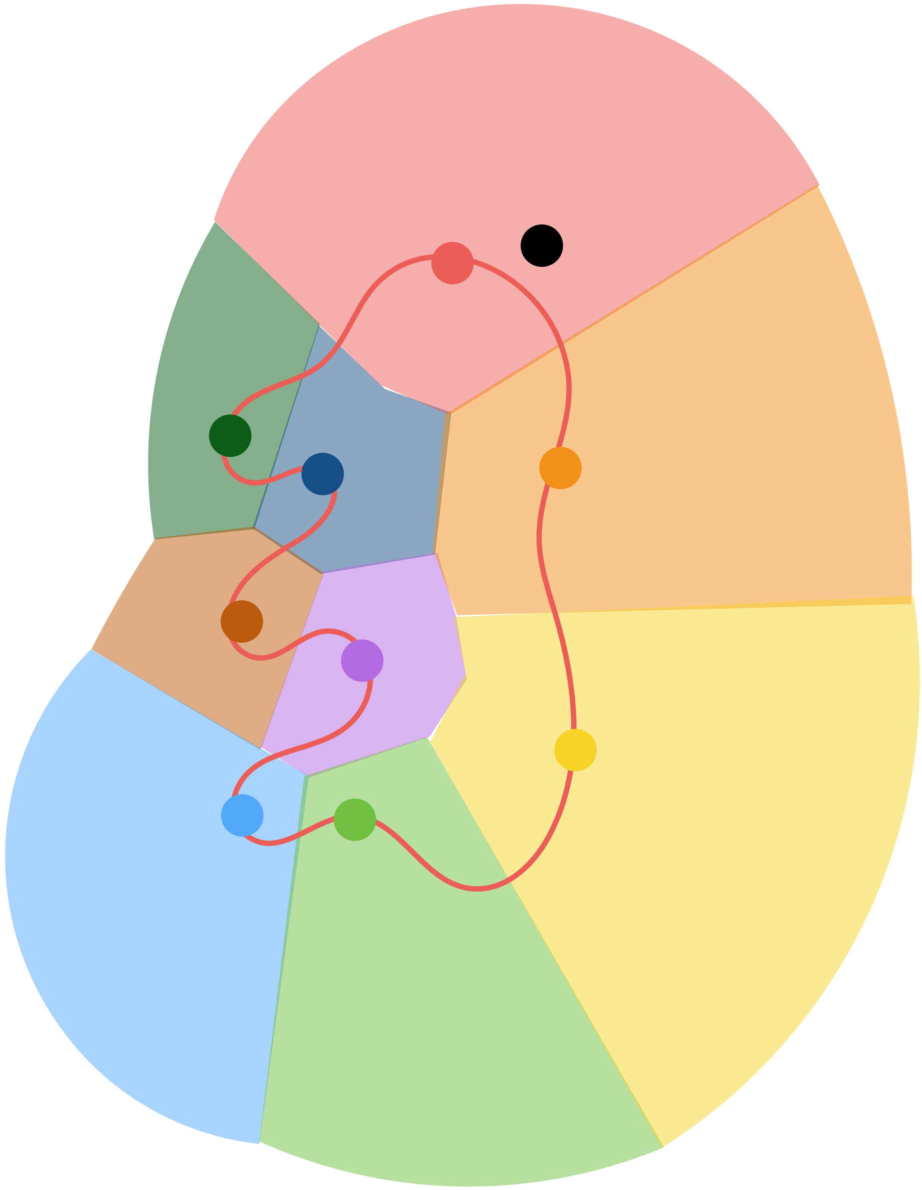
Closest points

- Brute force is n^2



Closest points

- Tree based methods (e.g. kdtree) have avg. complexity log(n)



- Random point sampling also reduces the running time

Closest points: avoid local minima

- Outlier removal, weighting according to inverse distance
- Use additional information (e.g. normals)
- Compute transformation based on greedy subsets of points: RANSAC

Is ICP the best we can do?

- iteration j
- compute closest points
- compute optimal transformation with Procrustes
- apply transformation
- terminate if converged, otherwise iterate

Best transformation?

- Procrustes gives us the optimal transformation given correspondences
- However, nothing guarantees that they are the best when correspondences are wrong!
- Can we do better?

Iterative Closest Point (ICP)

- iteration j
- compute closest points → In which direction should I move?
- compute optimal transformation with Procrustes
- apply transformation
- terminate if converged, otherwise iterate

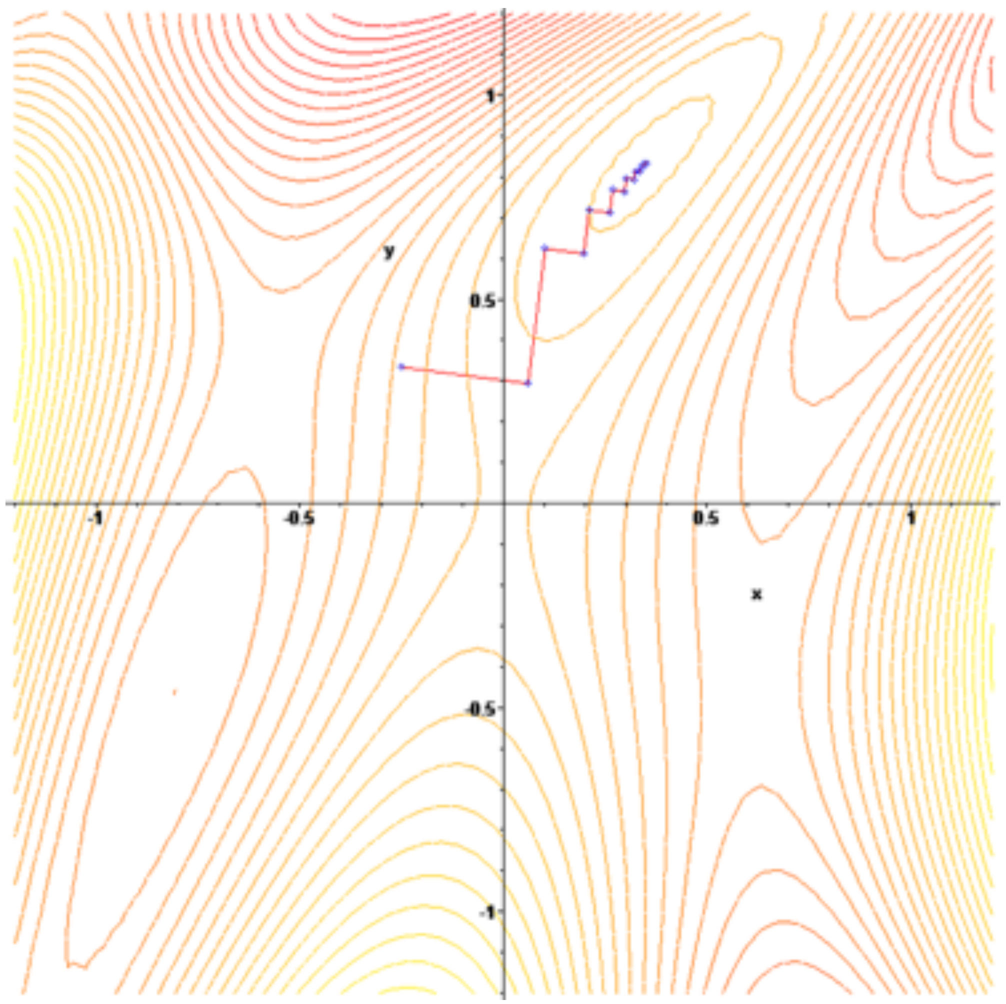
Iterative Closest Point (ICP)

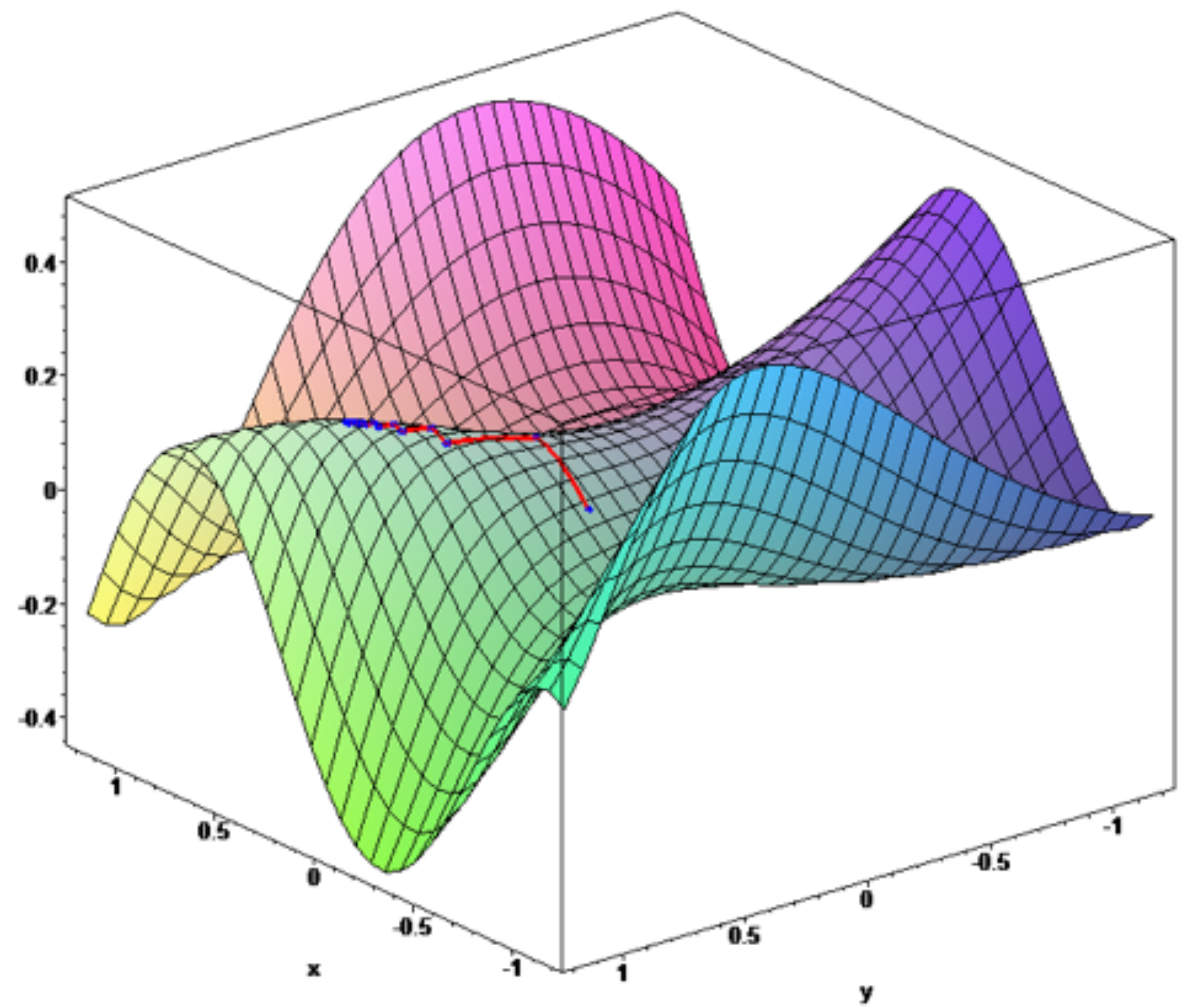
- iteration j
- compute closest points ➡ In which direction should I move?
- compute optimal transformation with Procrustes
  ➡ compute a transform that brings me there
- apply transformation
- terminate if converged, otherwise iterate

Gradient-based ICP

- iteration j
- compute closest points → Jacobian of distance-based energy
- ~~compute optimal transformation with Procrustes~~ → Step in the Jacobian (or Newton, or…) direction
- apply transformation
- terminate if converged, otherwise iterate

Gradient-based optimisation





Gradient-based ICP

1. Energy:

$$E \equiv \sum_i \| \min_{\mathbf{x}} f(\mathbf{x}) - \mathbf{y}_i \|^2$$

2. Consider the correspondences fixed in each iteration j+1

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

3. Compute gradient of the energy around current estimation

$$g^{j+1} = \nabla E(f^j)$$

4. Apply step (gradient descent, dogleg, LM, BFGS…)

$$f^{j+1} = k_{step}(g^{0 \dots j+1}, f^{0 \dots j})$$

(for example $f^{j+1} = f^j - \alpha g^{j+1}$)

5. terminate if converged, otherwise iterate (go to step 2)

Gradient-based ICP

- Energy:
- Consider the correspondences fixed in each iteration j+1
- Compute gradient of the energy around current estimation
- Apply step (gradient descent, dogleg, LM, BFGS…)
- terminate if converged, otherwise iterate

Gradient-based ICP

$$E \equiv \sum_i \| \min_{\mathbf{x}} f(\mathbf{x}) - \mathbf{y}_i \|^2$$

$$g^{j+1} = \nabla E(f^j)$$

- gradient: derivative of the sum of squared distances between target points and scale, rotated and translated source points, with respect to the the scale, rotation and translation

Gradient-based ICP

$$E \equiv \sum_i \| \min_{\mathbf{x}} f(\mathbf{x}) - \mathbf{y}_i \|^2$$

$$g^{j+1} = \nabla E(f^j)$$

- gradient: derivative of the sum of squared distances between target points and scale, rotated and translated source points, with respect to the the scale, rotation and translation

- Each derivative is easy

    - Who takes the chalk and writes it down?

Gradient-based ICP

$$E \equiv \sum_i \| \min_{\mathbf{x}} f(\mathbf{x}) - \mathbf{y}_i \|^2$$

$$g^{j+1} = \nabla E(f^j)$$

- gradient: derivative of the sum of squared distances between target points and scale, rotated and translated source points, with respect to the the scale, rotation and translation

- Each derivative is easy

  - Who takes the chalk and writes it down?

- Chain rule and automatic differentiation!

Chumpy

- https://pypi.python.org/pypi/chumpy
- Automatic differentiation compatible with numpy
- Jacobian: matrix encoding partial derivative of outputs (rows) with respect to inputs (columns)

$$\mathbf{J} = \frac{d\mathbf{b}}{d\mathbf{c}} = \begin{bmatrix} \frac{\delta b_1}{\delta c_1} & \cdots & \frac{\delta b_1}{\delta c_n} \\ \vdots & \ddots & \vdots \\ \frac{\delta b_m}{\delta c_1} & \cdots & \frac{\delta b_m}{\delta c_n} \end{bmatrix}$$

- The Jacobians of each operation are encoded for you
- The final gradient is computed with the chain rule

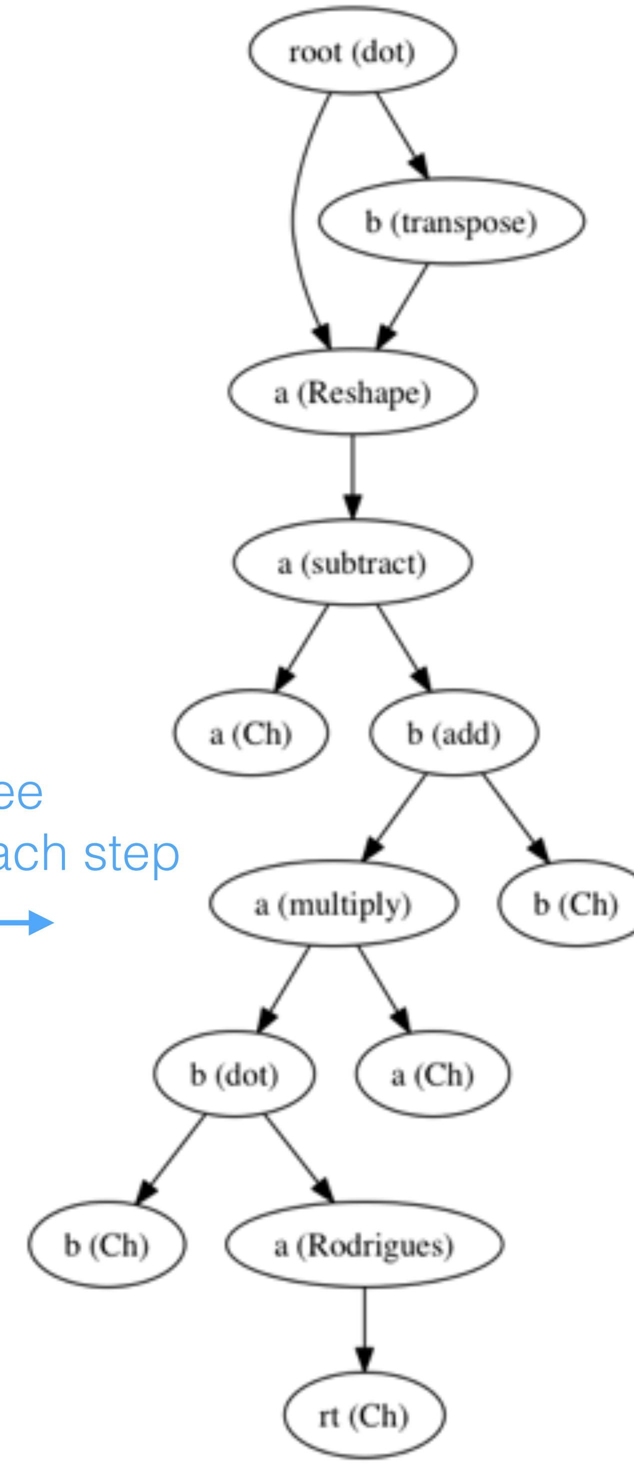
$$\mathbf{J}_{\mathbf{a}\circ\mathbf{b}}(\mathbf{c}) = \mathbf{J}_{\mathbf{a}}(\mathbf{b}(\mathbf{c}))\mathbf{J}_{\mathbf{b}}(\mathbf{c})$$

Chumpy

$$E = \sum_i \|s\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i\|^2$$

write as if it was numpy code

results in expression tree
with jacobians available at each step



Gradient-based ICP

- Energy:
- Consider the correspondences fixed in each iteration j+1
- Compute gradient of the energy around current estimation
- Apply step (gradient descent, dogleg, LM, BFGS…)

$$f^{j+1} = k_{step}(g^{0 \dots j+1}, f^{0 \dots j})$$

- terminate if converged, otherwise iterate

Gradient-based ICP

- The science of computing a good optimisation step is a whole field by itself
    - Ask Maren 😀
- However, lots of standard ways are available in scientific libraries like scipy
    - And chumpy integrates well with it
    - Minimisation in a single line:

or ‘BFGS’, ‘CG’, etc

```
ch.minimize(fun=energy, x0=[scale, rot, trans], method=‘dogleg’)
```

Why Gradient-based ICP?

- Formulation is much more generic: the energy can incorporate other terms, more parameters, etc
- Incorporates insights from the vast research community of gradient-based optimisation
- A lot of available software for solving this problem (cvx, ceres, …)
- **However**, when correspondences are not fixed, it's not guaranteed that a gradient-based step will work better than procrustes!

Take-home message

- Procrustes can also be applied to estimate camera parameters
    - … but only if we have correspondences!
- We can compute correspondences and solve for the best transformation iteratively with Iterative Closest Point (ICP)
- Procrustes is optimal given optimal correspondences: we *might* get better updates exploiting other optimisation strategies