Visual AI

CPSC 532R/533R – 2019/2020 Term 2

Lecture 7. Representation learning I

Helge Rhodin

Recap: Voxel representations

Idea: A 3D tensor that encodes occupancy

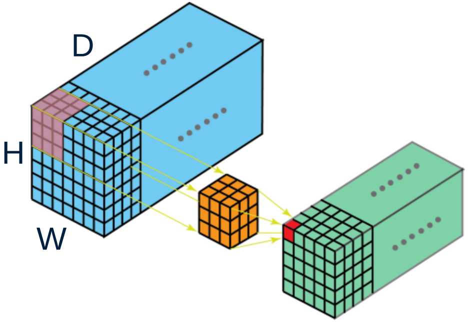
- stores binary values
  - occupied or empty cell

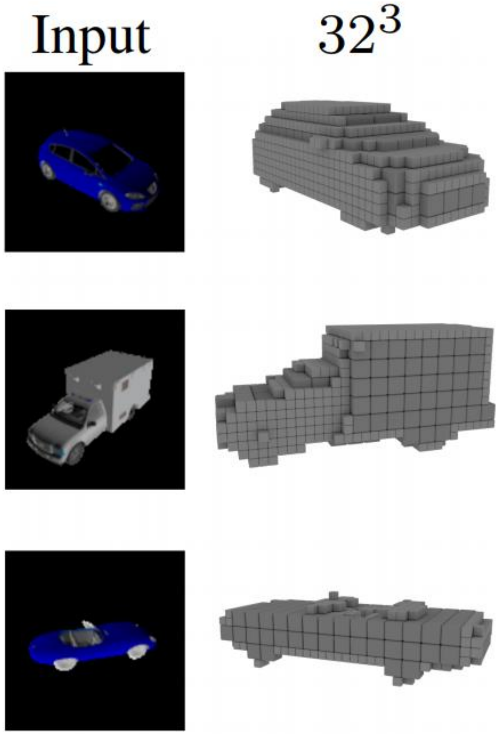
Size: C x D x H x W (C: channels, D: depth, H: height, W: width)

Batched size: N x D x H x W (N: number of elements in mini batch)

Benefits: We can apply 3D convolutions

- A generalization to 2D convolutions with a 3D kernel





[Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs]

Drawback:

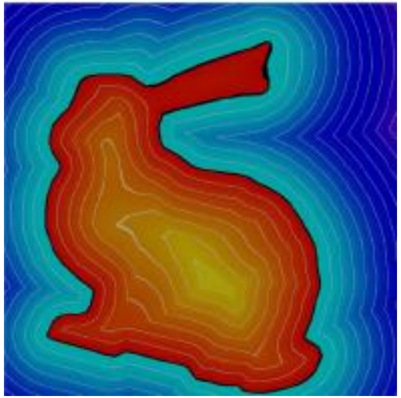
- cubic in memory footprint and computational complexity

Signed Distance Field (SDF)

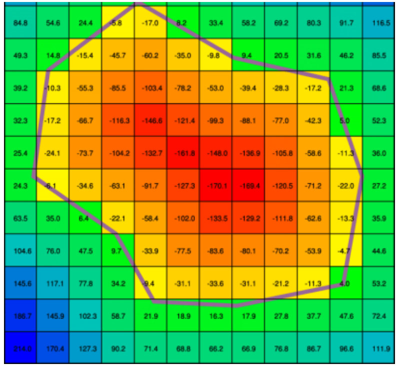
- input domain: dimension equal to the dimension of the space
  - usually two or three-dimensional
- output domain: a scalar
  - negative for inside of the object
  - positive outside

- continuous SDF: defined by a parametric function
  - e.g., sum of Gaussians, neural network
- discrete SDF: defined on a grid
  - e.g. 2D grid or 3D grid

- easy to display SDF in color code
  (red to blue = negative to positive)
- non-trivial to reconstruct the exact shape boundary



Continuous SDF



Discrete SDF

Implicit functions through NNs

[Saito et al., PIFu: Pixel-Aligned Implicit Function for High-Resolution Clothed Human Digitization]

Idea: Train a neural network that takes an image as well as a 3D query point as input and outputs:

- negative for positions **inside** the object
- positive **outside** the object
- reconstruct by querying a dense sampling

Advantage:

- No explicit limit on resolution (only limited by NN capacity)

Disadvantage:

- Reconstruction requires many network evaluations, its slow!

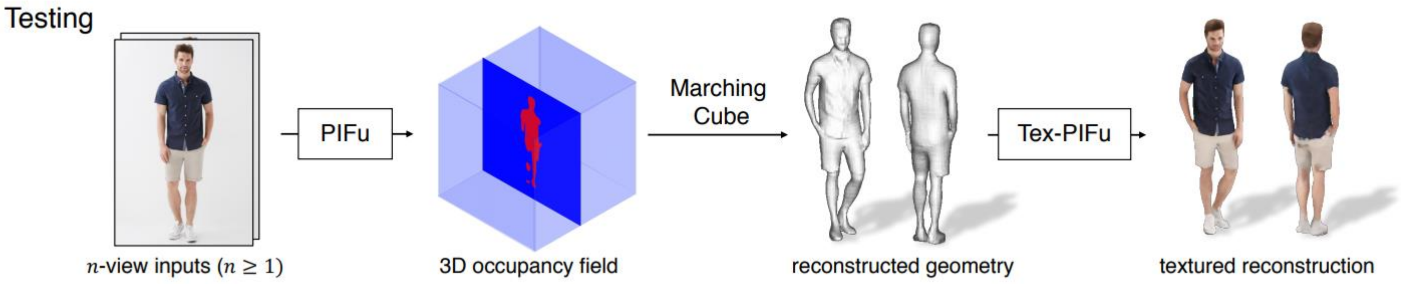
occupancy output
(positive within the object hull)

$$f(x) = \mathrm{CNN}_\theta(x, \mathbf{I})$$

3D query location

Image to reconstruct

Not straightforward to train…
wait for the paper presentation



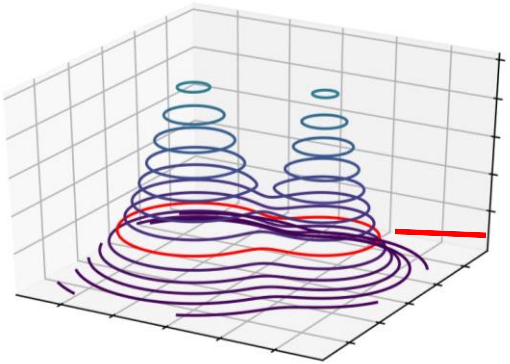
Recap: Implicit functions

Idea: define complex shapes as the zero-crossing of a function

Size: W (the number of parameters of the function)

- independent of output space dimension!

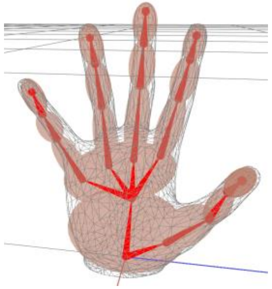
- Any parametric function works
  - e.g., mixtures of n Gaussian distributions
    with position mu and covariance Sigma

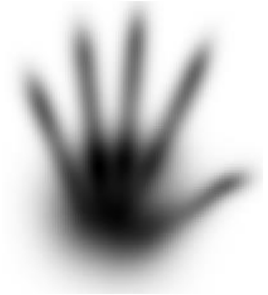


$$f(x) = \sum_{i=1}^{n} G(x, \mu_i, \sigma_i)$$

contour line / zero crossing

- a neural network?!





[Real-time Hand Tracking Using a Sum of Anisotropic Gaussians Model]

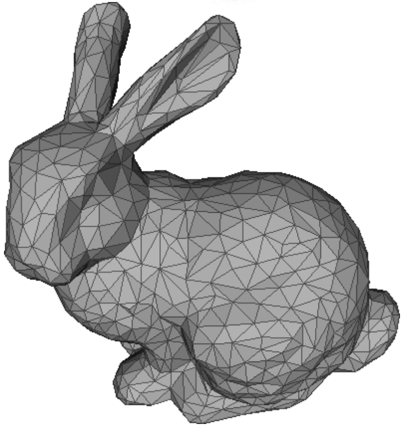
Recap: Surface mesh

Representation: Vertices connected by edges forming faces (usually triangles)

- Size: N x D + F x 3 (N: # points, D: space dimension, F: #triangles)
- A 3D surface parametrization (can be higher-dimensional)
  - Piece-wise linear with adaptive detail; triangle faces are usual

Benefits

- Good for single and multi-view reconstruction
- Provides orientation information (surface normal)
- Graph convolutions possible

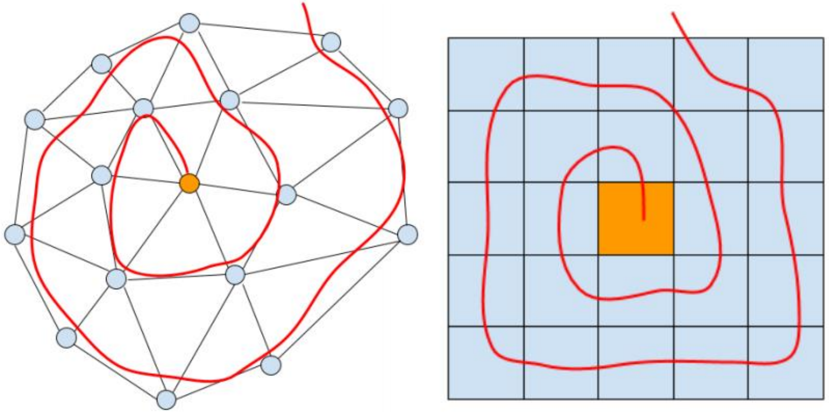


Drawbacks

- Irregular structure (number of neighbors, edge length, face area)
- Difficult to change topology
  (shape changes require to create new vertices and edges)

Spiral convolution

Goal: break the permutation invariance of neighbors



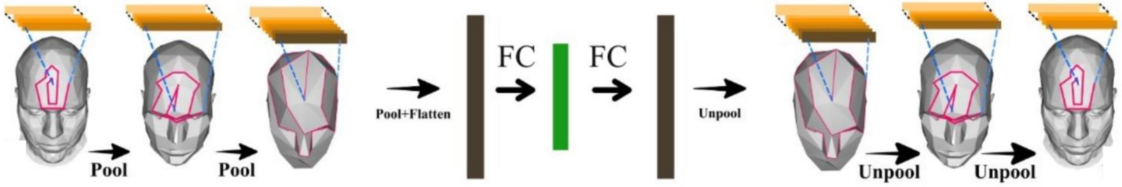
Idea: Order neighbors by simple rules

1. collect all neighbors (d hops in the graph)
2. pick the closest one (geodesic distance)
3. continue counterclockwise until spiral is of length
4. multiply features h along spiral with weight matric

$$\mathbf{h}_i^{(l+1)} = \sigma\left(h_{\text{spiral(neighbors}(i))} W^{(l)}\right)$$

Advantages:

- fixed number of points in each spiral
- efficient to compute
- anisotropic and topology-aware
- easy to optimize



Details: Mesh Laplacian

Goal: A form of 2nd order derivative on the mesh

Laplacian for a function in 3D space:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}$$

Difficulty:

- irregularity, where is left / right / up / down?

Solution:

- (weighted) average over all neighboring nodes Ni

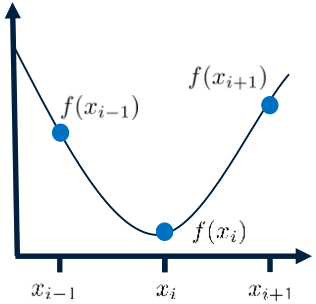
$$\mathscr{L}(\mathbf{v}_i) = \mathbf{v}_i - \frac{1}{d_i} \sum_{j \in \mathscr{N}_i} \mathbf{v}_j.$$

- Widely used to encode surface detail and to compare meshes
  - as a loss to compare surfaces

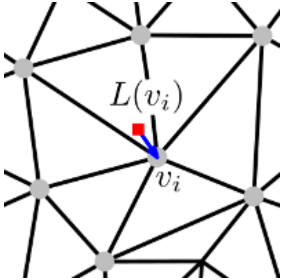
Finite differences approximation in 1D

$$f'(x_i, x_{i+1}) \approx \frac{f(x_{i+1}) - f(x_i)}{h}$$

$$f''(x_{i-1}, x_i, x_{i+1}) \approx \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2}$$



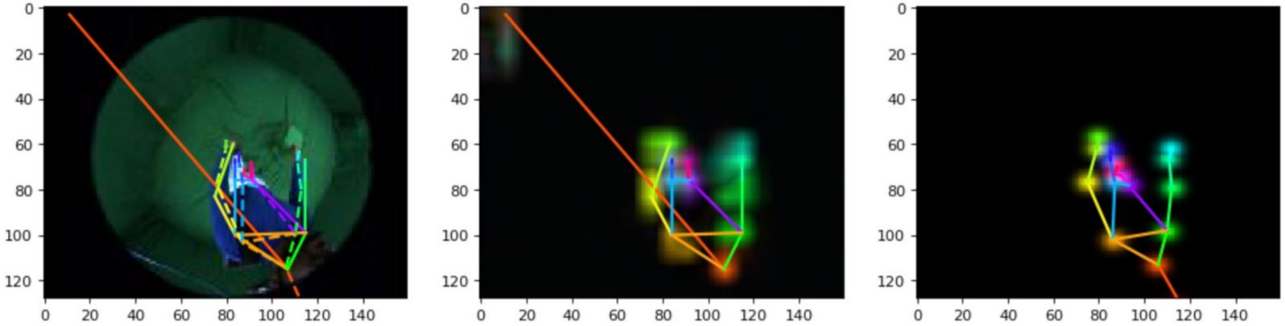
1D Laplacian



Graph Laplacian

Assignment 2 discussion

- Issues of heatmap prediction
  - outliers at inference time



- Issues of integral pose regression
  - bias towards the center

For example, if we have the following 1D heatmap

0, 1, 1, 0, 0, 0, 0,

after applying softmax we get the probability map

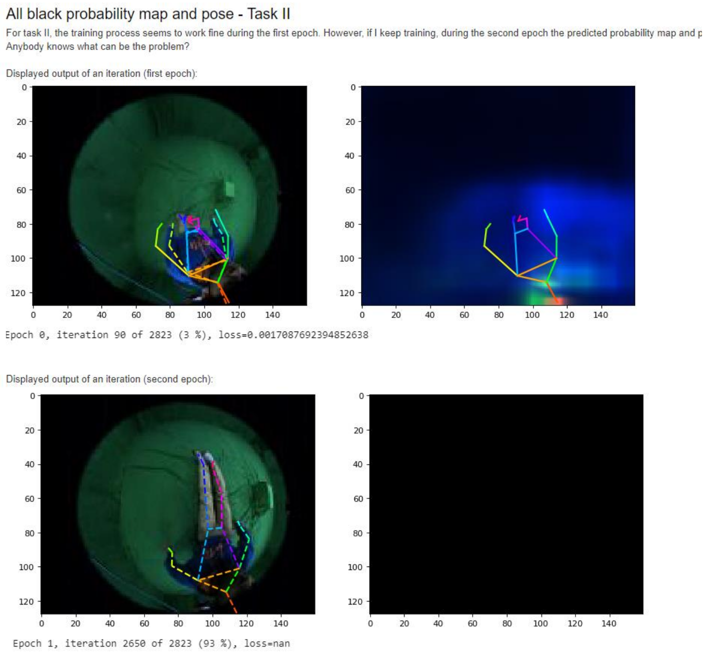
0.0958, 0.2605, 0.2605, 0.0958, 0.0958, 0.0958, 0.0958,

which leads to predicted position $0 \times 0.0958 + 1 \times 0.2605 + 2 \times 0.2605 + 3 \times 0.0958 + 4 \times 0.0958 + 5 \times 0.0958 + 6 \times 0.0958 \approx 2.5$

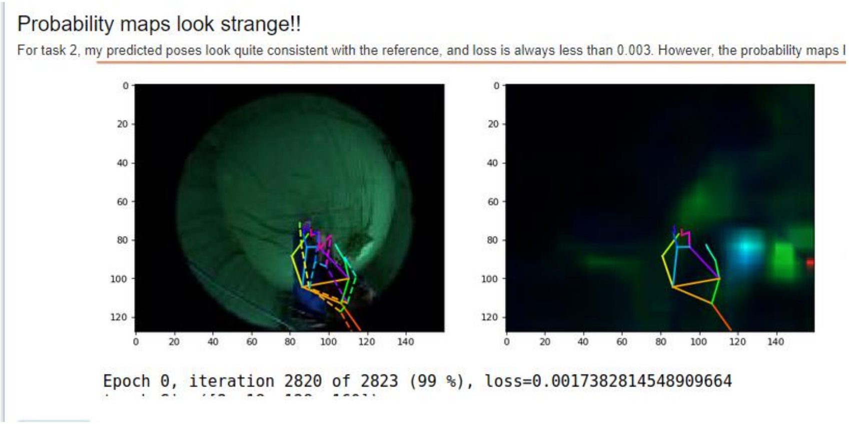
Assignment 2 discussion II

Mind the numerical stability of soft-max

- a stable implementation was introduced in an earlier lecture



Dimensions, width and height…



Any other issues?

Debugging best practice!

1. Basic principle: garbage in, garbage out
    - make sure your input has the correct type
        - correct tensor dimension, correct order of dimension, correct values, …
        - if it is an image or matrix, plot it
        - if you deal with points, plot them
2. How do I determine whether my input/output values are correct?
    - read the specification (e.g., assignment)
    - if there is no specification, write one
    - toy examples where you know the correct behavior
        - e.g., a single object, single color, primitive shape
        - try to separate influence factors, such as scale and shape
            - e.g., two images with the same shape but different scale
3. My input is correct, but the output is wrong, how do I find the bug?

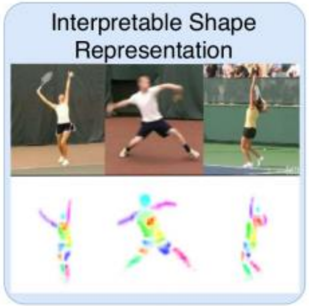
Wolf fence algorithm by Edward Gauss:

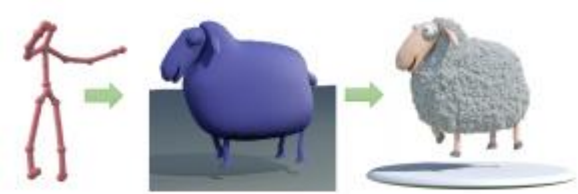
There's one wolf in Alaska; how do you find it?

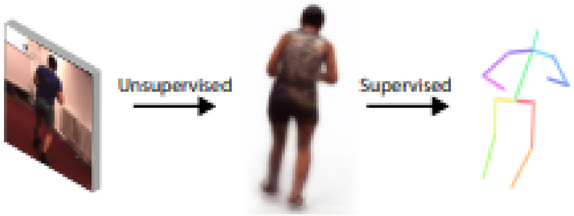
- build a fence down the middle of the state, wait for the wolf to howl, determine which side of the fence it is on (point 1&2).
- Repeat process on that side only, until you can see the wolf.

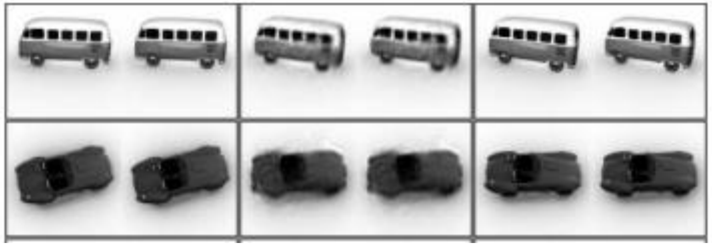
PCA and AEs will be important for the paper reading!

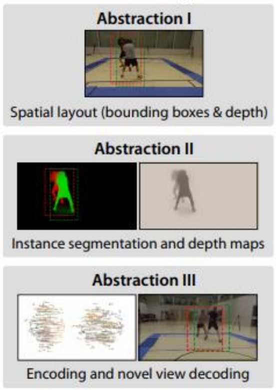
Principal Component Analysis (PCA) and Auto Encoder (AE)







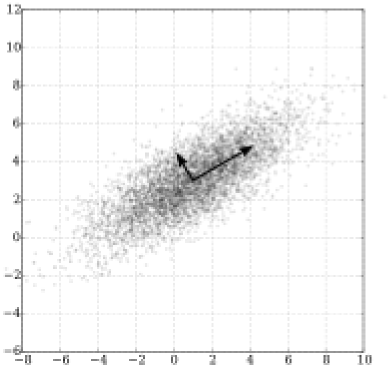




Principal Component Analysis (PCA)

Recap: Principal component analysis overview

- The orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate



two-dimensional space

- First weight vector w(1)

$$\mathbf{w}_{(1)} = \underset{\|\mathbf{w}\|=1}{\arg\max} \left\{ \sum_i \left(\mathbf{x}_{(i)} \cdot \mathbf{w}\right)^2 \right\}$$

  computed over all x(i) in the dataset

- … continue iteratively in orthogonal directions

- Stacking all weight vectors as rows into a matrix W yields a *'linear auto encoder'*

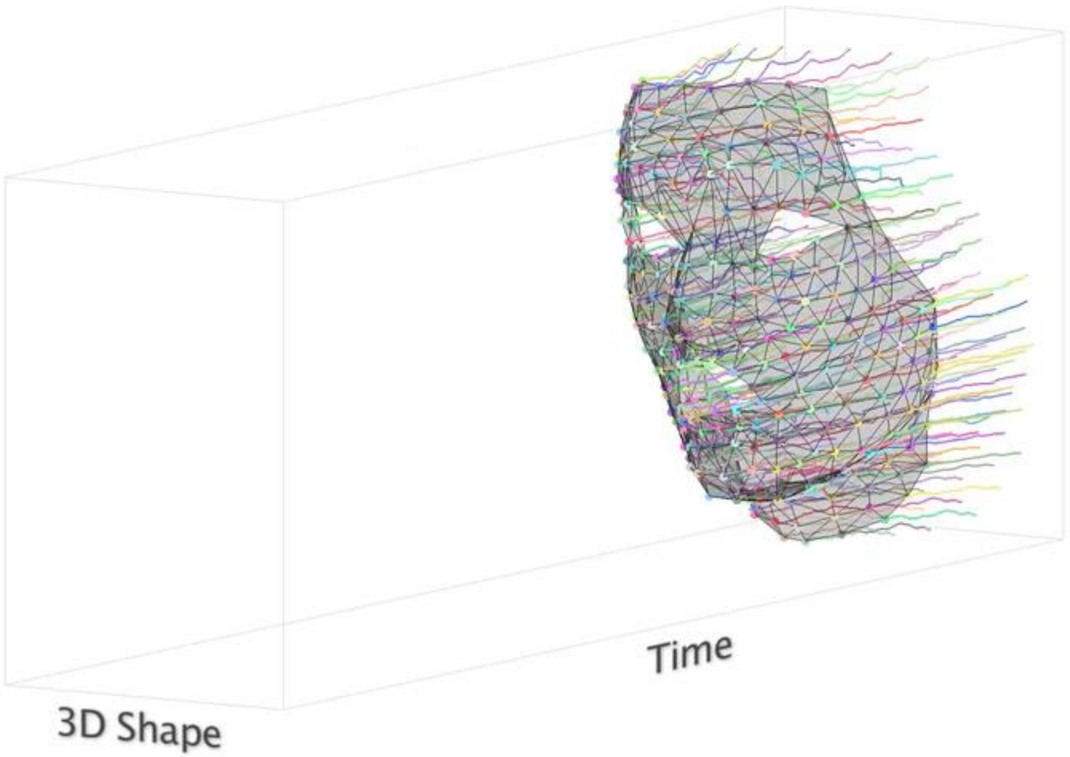
$$\hat{p} = \underbrace{W}_{\substack{\text{reconstruction} \\ \text{(decoding)}}} \underbrace{W^\top p}_{\substack{\text{projection} \\ \text{(encoding)}}}$$

thousand-dimensional space

PCA-like body model

[Movie Reshape]

PCA space: time or space?



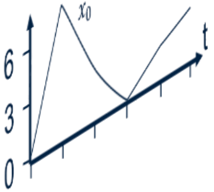
Data matrix

The data matrix X encodes

- each row represents a new measurement
- each column represents the

Measurement at time t_0

$$
X = \begin{array}{c|cccccc}
 & x_0 & y_0 & z_0 & x_1 & y_1 & z_1 \\
t_0 & 1 & 7 & 3 & 1 & 2 & 3 \\
t_1 & 8 & 2 & 4 & 4 & 5 & 1 \\
t_2 & 1 & 9 & 7 & 4 & 5 & 9 \\
t_3 & 2 & 9 & 4 & 4 & 5 & 2 \\
t_4 & 1 & 4 & 4 & 2 & 6 & 2 \\
t_5 & 1 & 9 & 2 & 1 & 5 & 5
\end{array}
$$



Motion of coordinate x_0

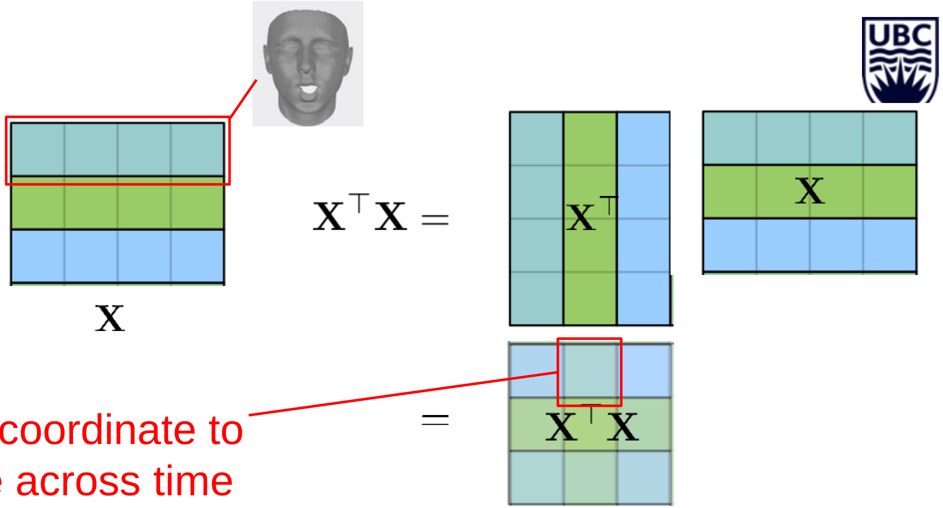
The covariance matrix

The empirical sample covariance matrix of $\mathbf{X}^T$

$$\frac{1}{1-n}\mathbf{X}^\top\mathbf{X}$$

We consider its unscaled form

$$\mathbf{X}^\top\mathbf{X}$$



Relation of first coordinate to the second one across time

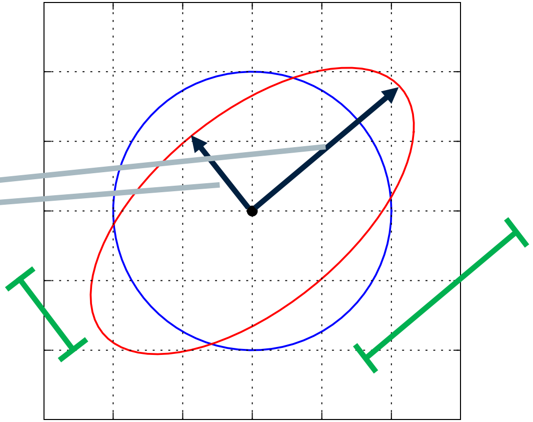
This symmetric matrix can be decomposed (Eigendecomposition)

$$\mathbf{X}^T\mathbf{X} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^T$$

Eigenvalues

Eigenvectors

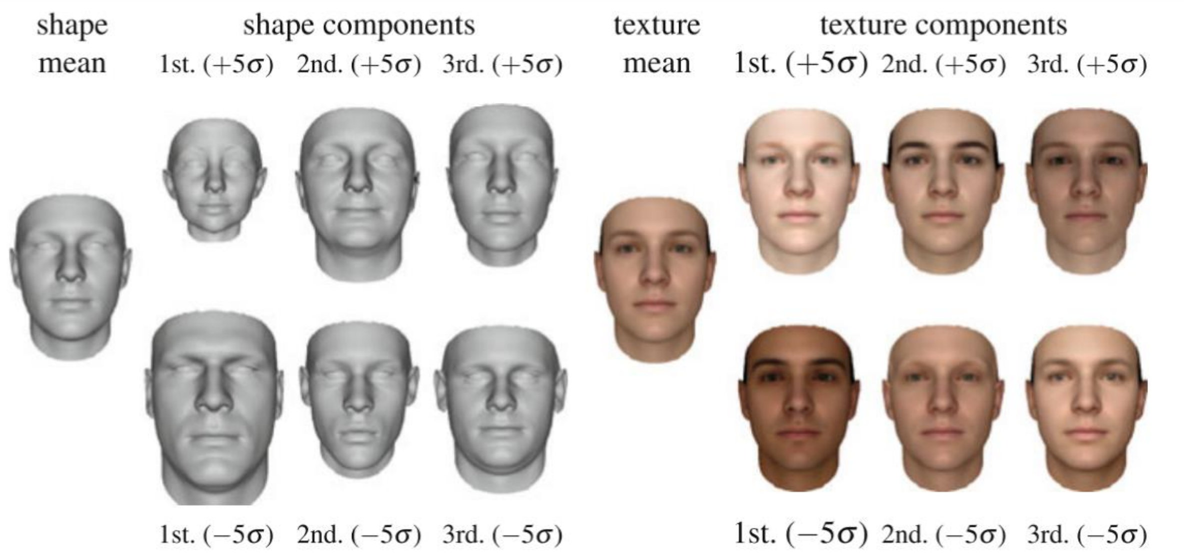
Sphere-ellipse representation
(how are points on a sphere deformed by $\mathbf{X}^T\mathbf{X}$)



Spatial components

- each component captures
- points that ‘move’ together
  - together = correlated
  - move = change across different measurements
  - e.g. left and right side
- if the input motion is smooth,
- it will lead to a smooth shape basis
  - global: scale ,male-female
  - forehead wrinkles in one basis

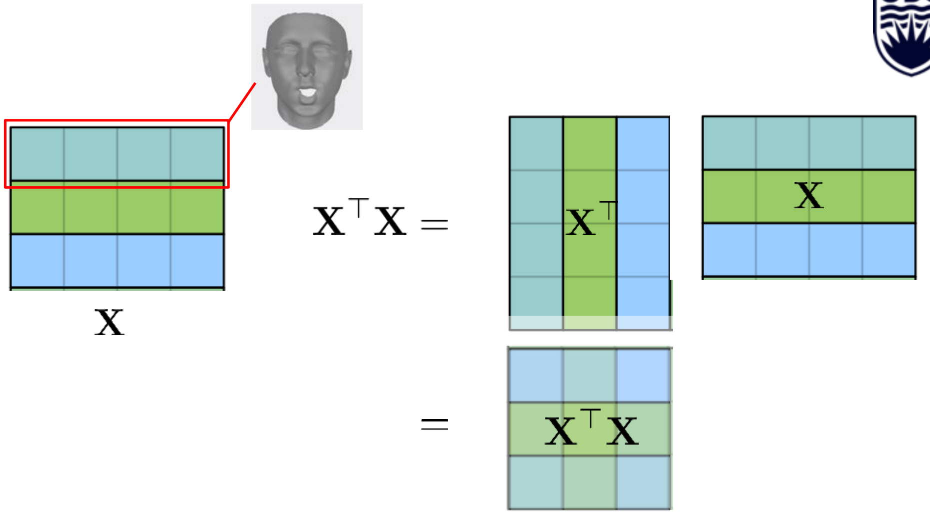
- works also on textures

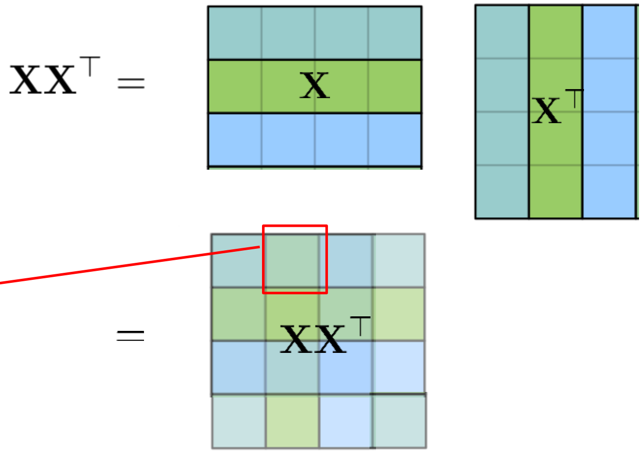


[Blanz and Fetter, A morphable model for the synthesis of 3D faces. **1999**]

The covariance matrix II

Exchanging the role of rows and columns



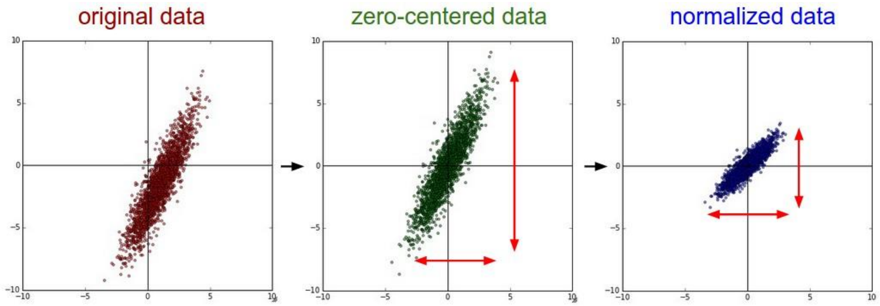


Relation of first **frame** to
the second one across vertices

Recall from lecture 3: Input and output normalization

Goal: Normalize input and output variables to have μ=0 and σ=1

$$\tilde{\mathbf{x}} = \frac{\mathbf{x} - \mu}{\sigma}$$



http://cs231n.github.io/neural-networks-2/

- For an image, normalize each pixel by the std and mean color (averaged over the **training** set)

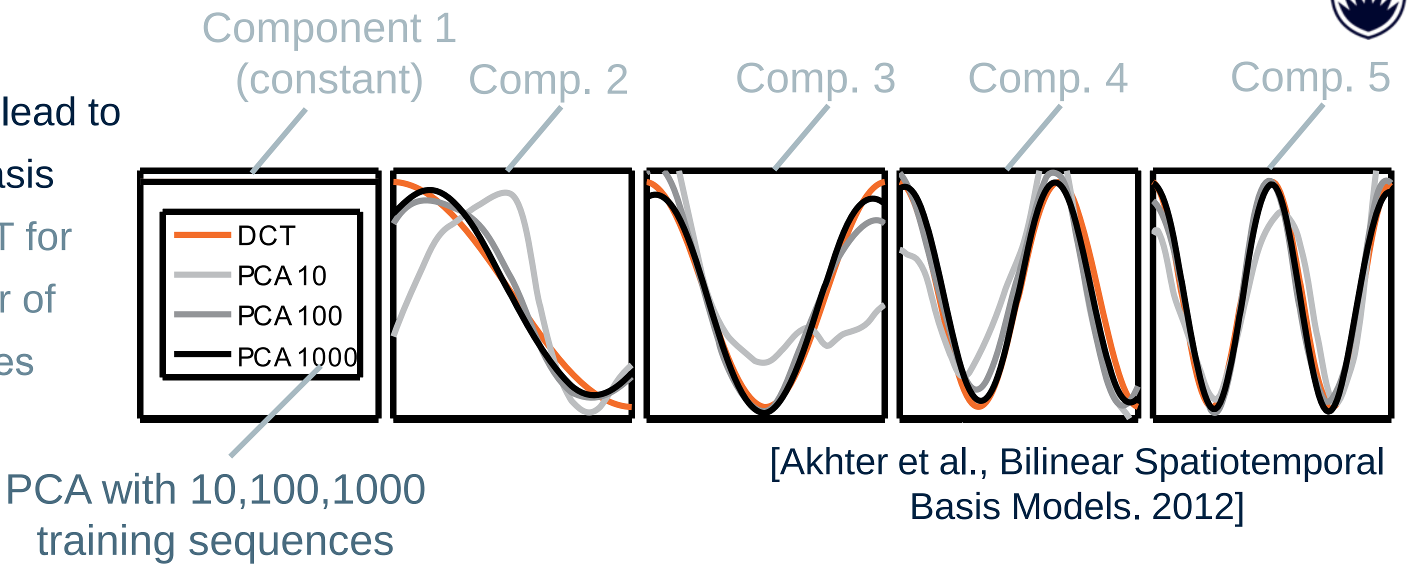
Related to data whitening

- whitening transforms a random vector to have zero mean and unit diagonal covariance
- by contrast, the default normalization for deep learning is element wise, neglecting dependency
  - the resulting covariance is not diagonal!

This is what we can do with PCA, it's a rotation and scaling of the data

Trajectory basis

- smooth input motions lead to a smooth trajectory basis
  - approximates DCT for increasing number of 'training' sequences



[Akhter et al., Bilinear Spatiotemporal Basis Models. 2012]

Fourier transform / Discrete cosine transform (DCT)

- a change of basis
  - orthogonal basis
  - turn a function of time into a function of frequency

Singular value decomposition (SVD)

Decomposition of the data matrix X with SVD

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{W}^T$$

orthonormal (unit length and linearly independent columns) matrix of eigenvectors

diagonal matrix of singular values

- singular values are arranged in descending order (makes SVD unique)

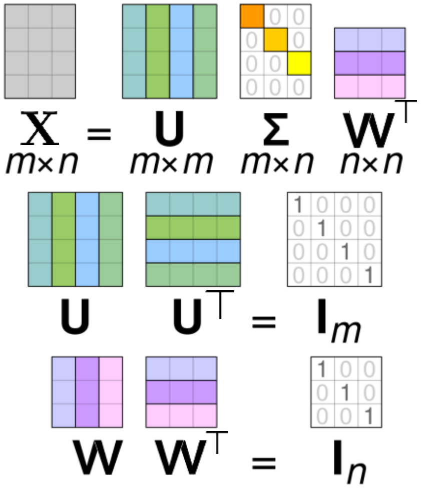
- closely related to PCA:

$$\mathbf{X}^T\mathbf{X} = \mathbf{W}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{W}^T$$
$$= \mathbf{W}\mathbf{\Sigma}^T\mathbf{\Sigma}\mathbf{W}^T$$

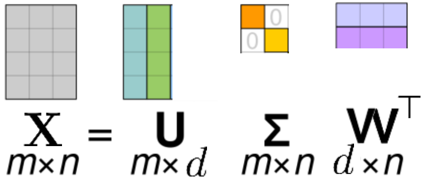
trajectory basis

shape basis

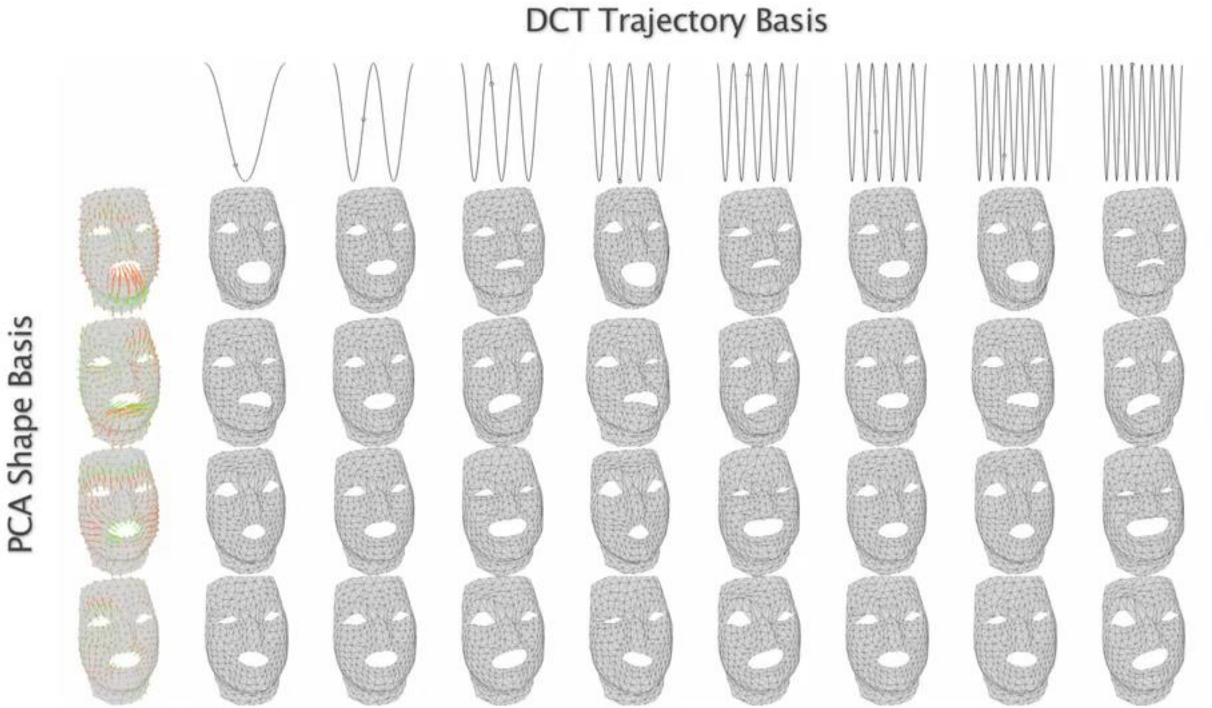
SVD in matrix form



Dimensionality reduction



Extension: Bilinear model

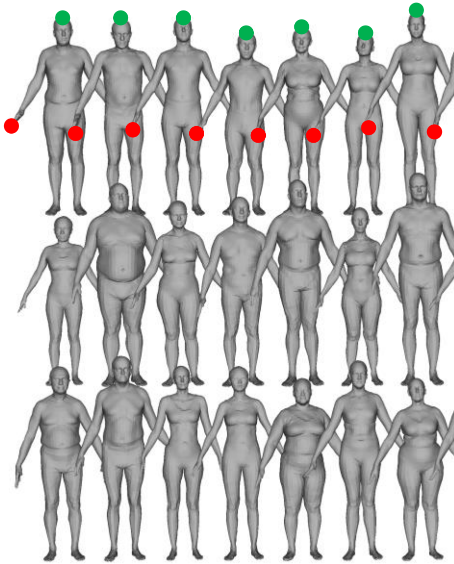


[Akhter et al., Bilinear Spatiotemporal Basis Models. 2012]

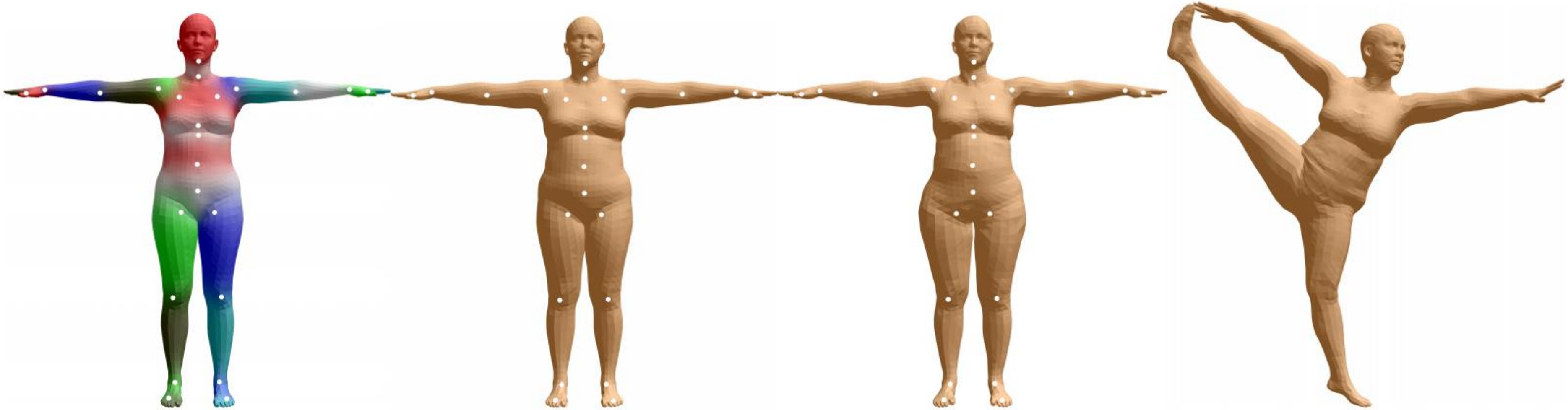
PCA - correspondence

PCA requires multiple ‘measurements’ of the same quantity

- e.g., for a human mesh model:
    - same number of vertices in mesh
    - the same vertex must correspond to the same semantic position. E.g., vertex 612 is the nose
- holes (missing data) is not supported
    - inappropriate for monocular reconstructions, e.g., where the back of the person is missing
    - generalizations exist to address this case
- scale sensitive
    - estimates those components that maximize variance
        - facial details are outweighed by belly shape
            - for human perception the face is important!



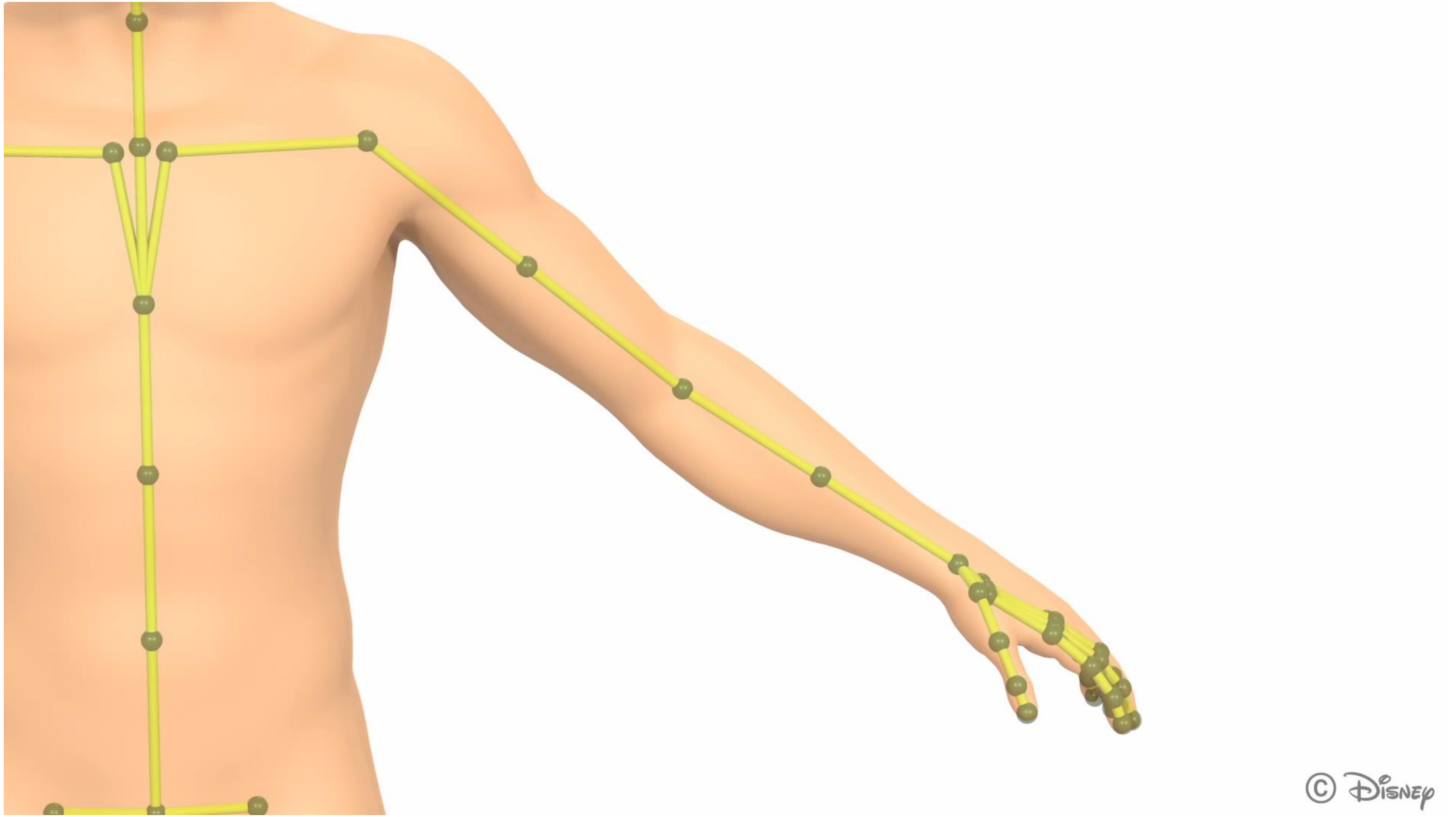
Recap: SMPL: A Skinned Multi-Person Linear Model



(a) $\bar{\mathbf{T}}$, $\mathcal{W}$ **(b)** $\bar{\mathbf{T}} + B_S(\vec{\beta})$, $J(\vec{\beta})$ **(c)** $T_P(\vec{\beta}, \vec{\theta}) = \bar{\mathbf{T}} + B_S(\vec{\beta}) + B_P(\vec{\theta})$ **(d)** $W(T_P(\vec{\beta}, \vec{\theta}), J(\vec{\beta}), \vec{\theta}, \mathcal{W})$

Figure 3: SMPL model. *(a) Template mesh with blend weights indicated by color and joints shown in white. (b) With identity-driven blendshape contribution only; vertex and joint locations are linear in shape vector $\vec{\beta}$. (c) With the addition of of pose blend shapes in preparation for the split pose; note the expansion of the hips. (d) Deformed vertices reposed by dual quaternion skinning for the split pose.*

Recap: Skinning



Auto Encoder (AE)

General case

$$\mathbf{h} = \text{encoder}_\theta(\mathbf{x})$$

$$\mathbf{x}' = \text{decoder}_\theta(\mathbf{h})$$

Simple non-linear case

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{x}' = \sigma(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

Linear case

$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{x}' = \mathbf{W}'\mathbf{h} + \mathbf{b}'$$

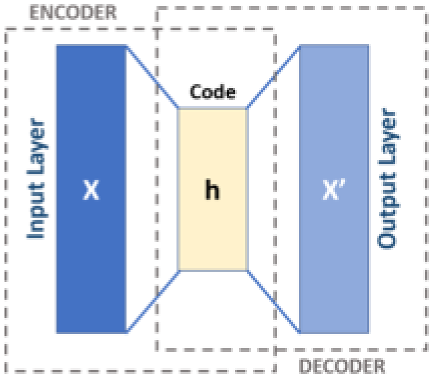
General reconstruction objective

$$\text{loss}(\mathbf{x}, \mathbf{x}')$$

- e.g., MSE loss

A two-layer fully-connected neural network

Similar to PCA when using squared loss (W spans the same space, but neither forms an ordered nor orthogonal basis)



https://en.wikipedia.org/wiki/Autoencoder

Linear autoencoder objective

$$\arg\min_{\mathbf{W}} \sum_i \|\mathbf{x} - \mathbf{x}'\|^2$$

$$= \arg\min_{\mathbf{W}} \sum_i \|\mathbf{x}_{(i)} - \mathbf{W}'\mathbf{W}\mathbf{x}_{(i)}\|^2$$

PCA objective

$$\mathbf{w}_{(1)} = \arg\max_{\|\mathbf{w}\|=1} \left\{ \sum_i \left(\mathbf{x}_{(i)} \cdot \mathbf{w}\right)^2 \right\}$$

$$= \arg\max_{\|\mathbf{w}\|=1} \left\{ \mathbf{w}^T \mathbf{X}^\mathbf{T} \mathbf{X} \mathbf{w} \right\}$$

[From Principal Subspaces to Principal Components with Linear Autoencoders]

Autoencoder variants

Bottleneck autoencoder:

- hidden dimension smaller than input dimension
  - leads to compressed representations
    - like dimensionality reduction with PCA

Sparse autoencoder:

- hidden dimension larger than input dimension
- hidden activation enforced to be sparse
  (=few activations

Denoising autoencoder:

- corrupt the input values, e.g. by additive noise

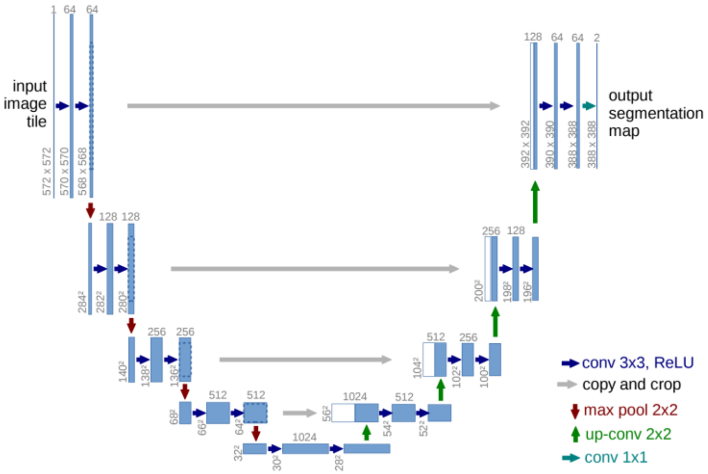
$$\mathbf{h} = \mathrm{encoder}_\theta(\underline{\mathrm{noise}}(\mathbf{x}))$$
$$\mathbf{x}' = \mathrm{decoder}_\theta(\mathbf{h})$$

Variational Auto Encoder (VAE)

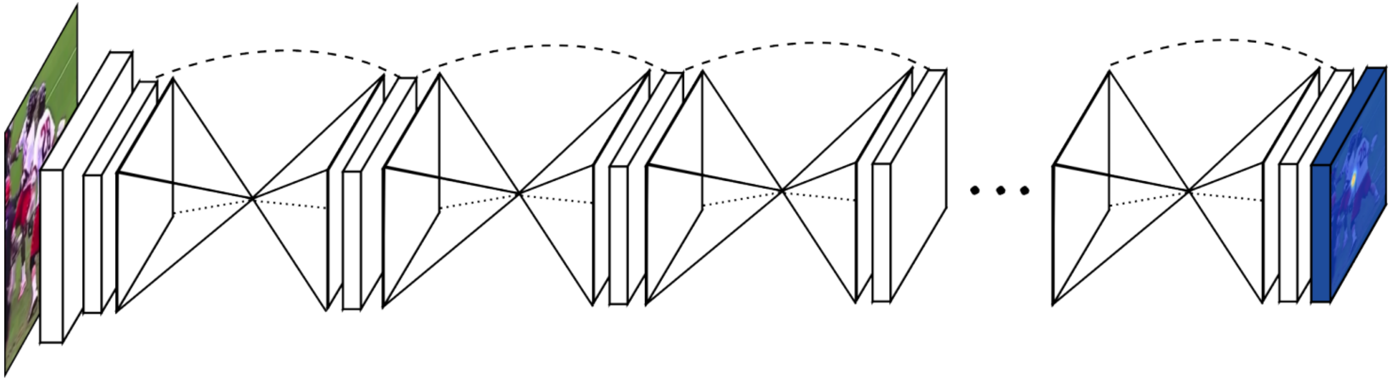
- a probabilistic model
  - *‘adding noise on the hidden variables’*
  - more in lecture 8!

Relation to previous lectures

- The UNet has an encoder-decoder structure



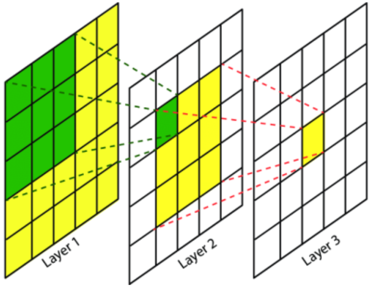
- The stacked hourglass network applies multiple encoders and decoders



Preparation for Assignment 3

- Will be posted tonight or tomorrow
  - PyTorch issues encountered
    - circular convolutions are broken…
  - the current version is a bit boring

Feature map size after convolutional kernels



Transformation of input and output by convolutions

- output size = (input size + 2*padding – kernel size + stride)/stride
  - e.g., a 3x3 kernel that preserves size: W + 2*1 – 3 + 1 = W
  - e.g., a 4x4 kernel that reduces size by factor two: (W + 2*1 – 4 + 2)/2 = W/2
- holds per dimension, i.e., 1D, 2D and 3D convolutions

Transformation of input and output by **transposed** convolutions (aka. deconvolution)

- output size = input size * stride - stride + kernel size - 2*padding
  - it has exactly the opposite effect of convolution
  - e.g., a 3x3 kernel that preserves size: W - 1 + 3 - 2*1 = W
  - e.g., a 4x4 kernel that **increases** size by **factor** two: W*2 + 2*1 – 4 + 2 = W*2
  - e.g., a 3x3 kernel that **increases** size by two elements: W - 1 + 3 - 2*0 = W + 2

Presentation topic assignment ongoing

Anyone missing who send their choice?

Summary: 19 votes for 22 papers

- Gives 3 late votes or remaining slots?
- Remaining papers can be presented by auditing students
  - volunteers?

Dingqing
Ege Unlu
Dave Pagurek van Mossel
Shuxian Fan
Shelly C
Peyman Bateni
Tim Straubinger
Shenyi Pan
Michela Minerva - michela
Jerry Yin
Willis Peng
Michelle Appel
stolet
Zicong Fan (Alex)
fjavadi
ssims
Daniele Reda
Shih-Han Chou
Weidong Yin

Project

Project proposal

- 3-minute pitch per group
- written plan
  - one page, 11pt font, may include figures
    - not more than one, not less than half a page of text
- the proposal plan must cover
  - the research idea
  - the possible algorithmic contributions
  - and an outline of the evaluation
- get feedback from during office hours
  - Yuchi on Tuesdays
  - me on Wednesdays
  - Only three weeks left!

Hidden questions