Visual AI

CPSC 532R/533R – 2019/2020 Term 2

Lecture 6. Representing and learning shapes

Helge Rhodin

Forward kinematics, linear or not?

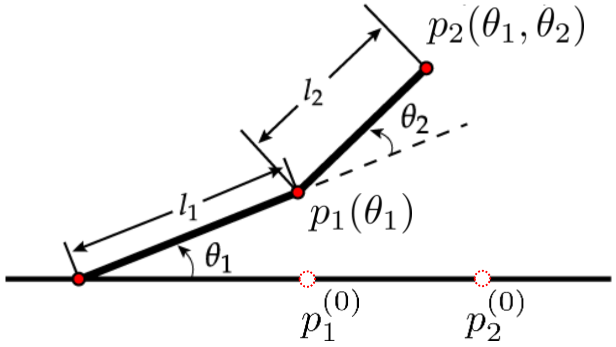
Forward kinematics

- non-linear in the angle (due to cos and sin)

$$R_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 \\ \sin\theta_1 & \cos\theta_1 \end{bmatrix} \qquad R_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 \\ \sin\theta_2 & \cos\theta_2 \end{bmatrix}$$

- linear given a set of rotation matrices

$$p_2(\theta_1, \theta_2) = R_1 p_1^{(0)} + R_2 R_1 \left(p_2^{(0)} - p_1^{(0)} \right)$$

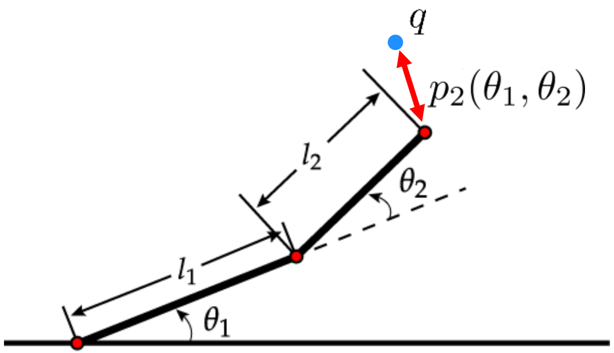


Inverse kinematics

- minimize objective to reach goal location q

$$O(\theta_1, \theta_2) = \| q - p_2(\theta_1, \theta_2) \|$$

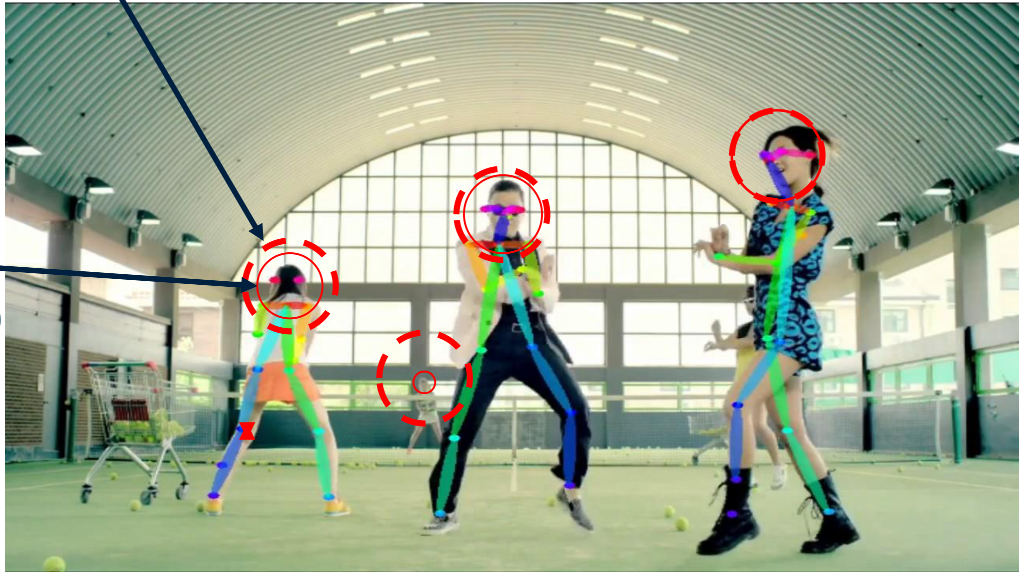
- difficult, due to nonlinear dependency on theta



Recap: Percentage of Correct Keypoints (PCK)

- The number of keypoints below a threshold
  - usually using Euclidean distance
  - less sensitive to outliers
  - scale sensitive

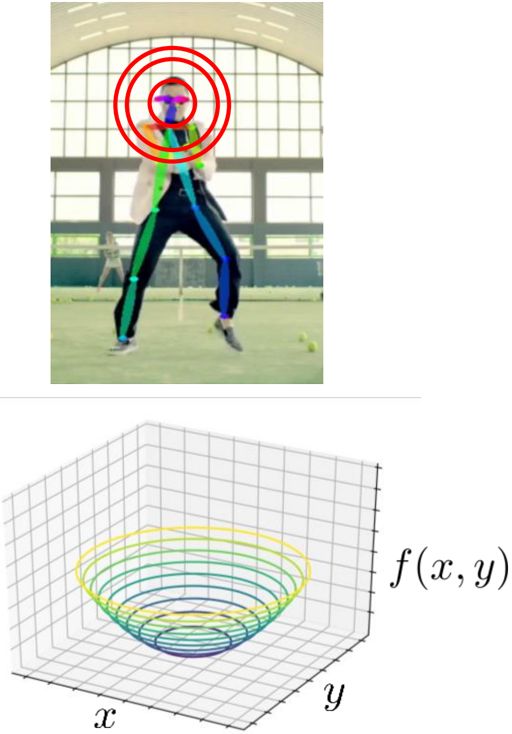
- Scale invariant version: PCKh
  - relative to the scale of the GT annotatio
    - e.g. halt the head-neck distance is common for 2D human pose



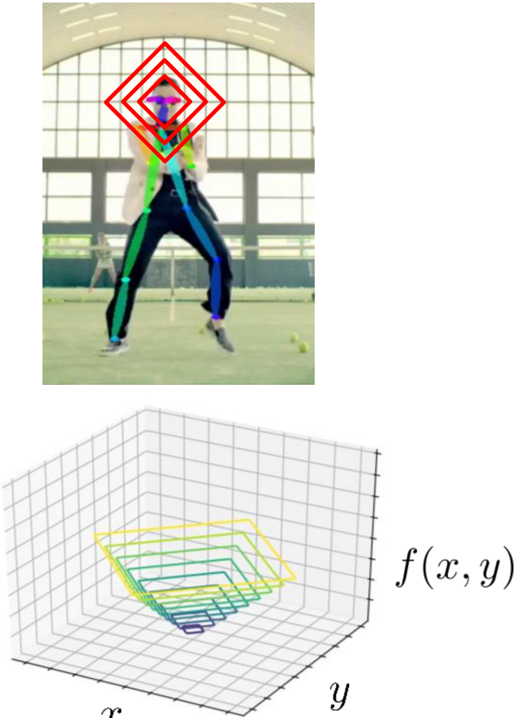
Loss comparison

Contour lines

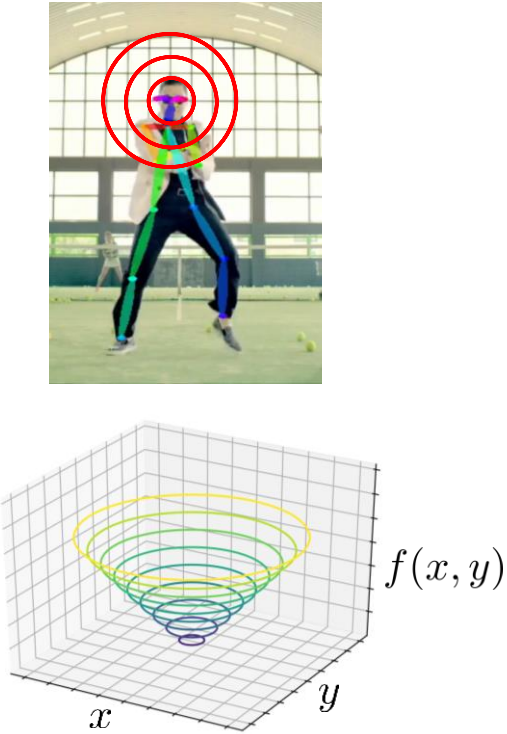
3D slope



Squared error
MSE



Absolute error
MAE



Euclidean distance
MPJPE

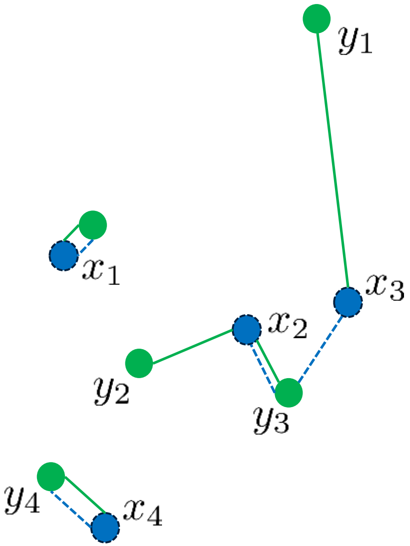
Recap: Chamfer distance

A distance between point clouds without correspondence

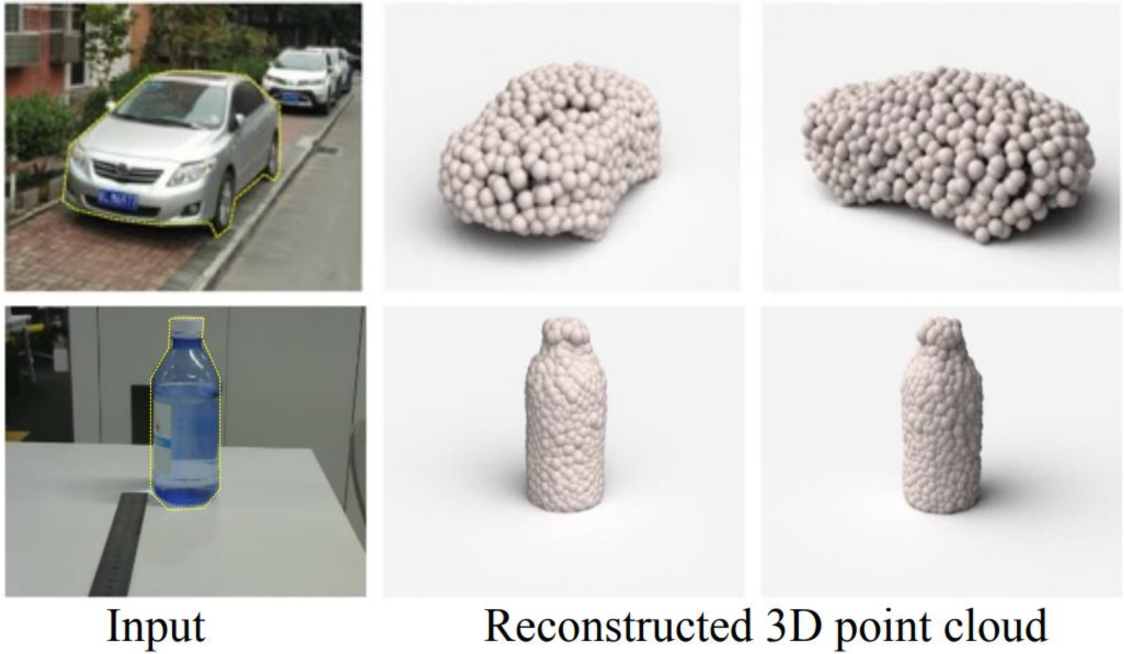
- sum of distances between closest points
- bi-directional
  - closest point of y in Y for all x in X
  - closest point of x in X for all y in Y

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \underbrace{\min_{y \in S_2} \|x - y\|_2^2} + \sum_{y \in S_2} \underbrace{\min_{x \in S_1} \|x - y\|_2^2}$$

  - is not a *distance function* in the mathematical sense, because the triangle inequality does not hold

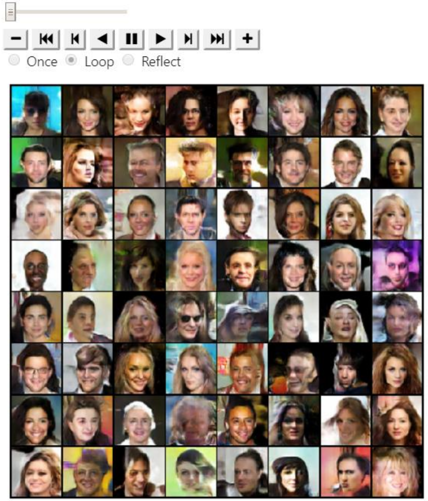


A Point Set Generation Network for 3D Object Reconstruction from a Single Image

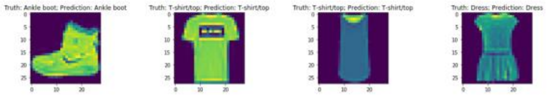


Shape completion

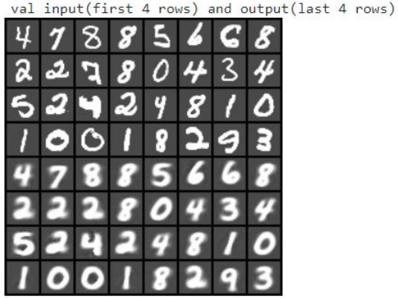
Assignment 1 highlights



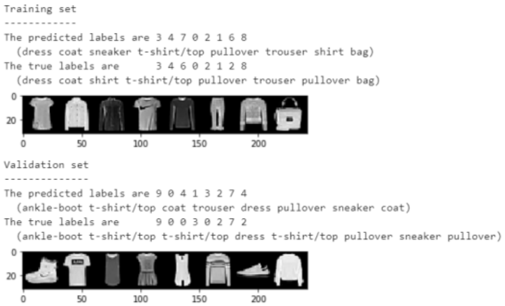
GAN on faces



Classification on fashion MNIST



Variational auto encoder



More fashion

Transfer learningD

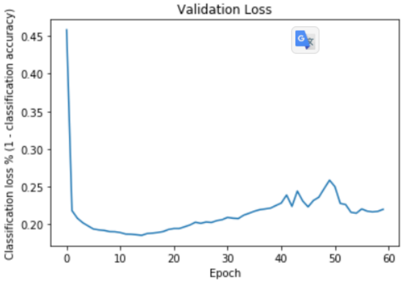
Assignment 1 highlights

Useful sources

- https://pytorch.org/tutorials/beginner/data_loading_tutorial.html
- https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
- https://towardsdatascience.com/build-a-fashion-mnist-cnn-pytorch-style-efb297e22582
- https://pytorch.org/tutorials/
- https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
- https://pytorch.org/docs/stable/torchvision/datasets.html#mnist
- https://www.tensorflow.org/tensorboard/tensorboard_in_notebooks
- https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

Please include your output in the submission and make it readable

```
[9]: Text(0, 0.5, 'Classification loss % (1 - classification accuracy)')
```




```
[12]: epoch = training_loss[:,1]
      loss = training_loss[:,2]
      plt.plot(epoch,loss)
      plt.title("Training loss: cross entropy loss")
      plt.xlabel("Gradient Update")
      plt.ylabel("Cross Entropy Loss)")

[12]: Text(0, 0.5, 'Cross Entropy Loss)')
```


```
[ ]:
```

Good (plot with labels)

```
[ ]: optimizer = optim.Adam(network.parameters(), lr=0.0001)
     loss_interval = 10
     print_interval = 10

     training_losses = []
     validation_losses = []
     for epoch in range(5):
         running_loss = 0.0
         for i, data in enumerate(training_loader, 0):
             optimizer.zero_grad()
             output = network(data)
             loss = criterion(data["output"], output["prediction"]) # Task IV
             loss.backward()
             optimizer.step()

             running_loss += loss.item()
             if i % loss_interval == loss_interval - 1:
                 training_losses.append(running_loss / loss_interval)
                 running_loss = 0.0
                 validation_losses.append(validation_loss())

             if i % print_interval == print_interval - 1:
                 # Task VII
                 # Task VIII
                 clear_output(wait=True)
                 print('epoch %d, iteration %5d' % (epoch + 1, i + 1))
                 plt.plot(training_losses, label="Training Loss");
                 plt.plot(validation_losses, label="Validation Loss");
                 plt.legend()
                 plt.show()

     print("Done training")

[ ]: # Task V
     test_loader = torch.utils.data.DataLoader(
         test_dataset,
         batch_size=num_test_examples,
         shuffle=True,
     )

     test_batch = next(iter(test_loader))
     def test_loss():
         preds = network(test_batch)["prediction"]
         out = test_batch["output"]
         loss = criterion(preds, out).cpu().detach().numpy();
         return np.mean(loss)

     print("The final test loss is ", test_loss())

[ ]: def show_predictions(loader, n):
```

Output missing!

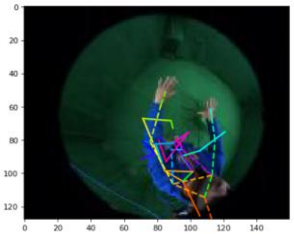
Assignment 2: Clarification

Heatmap-based pose classification

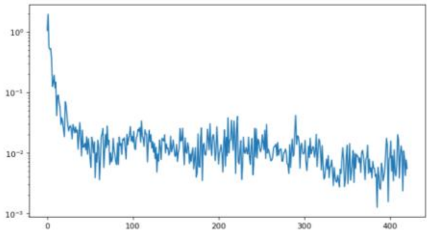
[*]:
```
# Detection network that handles dictionaries as input and output
class HeatNetWrapper(torch.nn.Module):
    def __init__(self, net):
        super().__init__()
        self.net = net

    def forward(self, dictionary):
        return DeviceDict({'heatmap':(self.net(dictionary['img'])['out'])})
num_joints = len(joint_names)
det_network = HeatNetWrapper(torchvision.models.segmentation.deeplabv3_resnet50(num_classes=num_joints)).cuda()
```

[*]:
```
hat takes an NxKx2 pose vector (N: batch dimension, K: number of keypoints) to create stacks of heatmaps that have Gaussian distribution with the mean at the keypoint and standard deviation equal to 3.
 argument specifies the output dimensions of the map. Note that the keypoints are defined in normalized coordinates, ranging from 0..1 irrespectively of the image resolution.
```

Was meant to be 3 pixel. Chose your own std instead!



Epoch 0, iteration 420 of 2823 (14 %), loss=0.005484297405928373



Empty graph in example output has been removed in final version

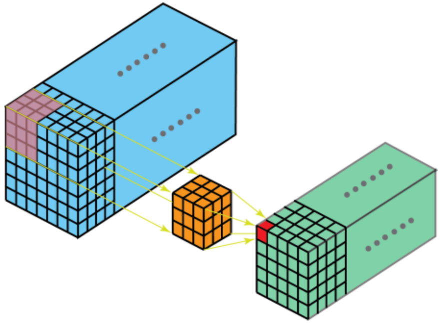
Voxel representations

Idea: A 3D tensor that encodes occupancy

- stores binary values
  - occupied or empty cell

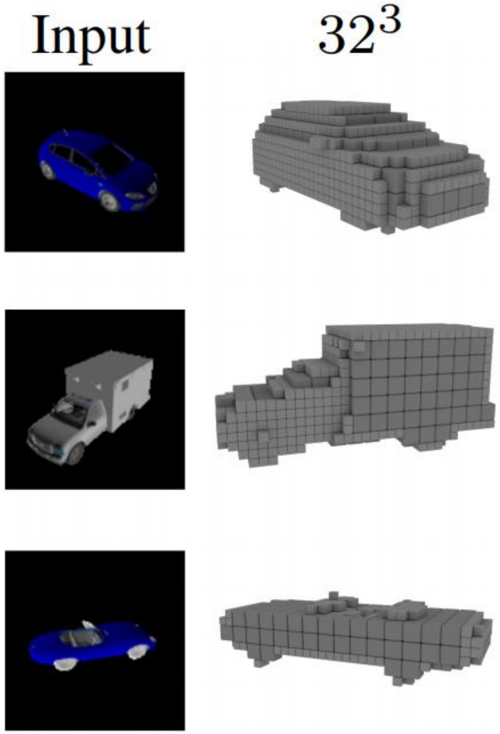
Benefits: We can apply 3D convolutions

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



Drawback:

- cubic in memory footprint and computational complexity

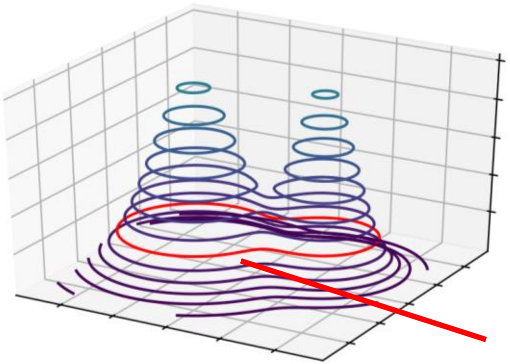


[Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs]

Implicit functions

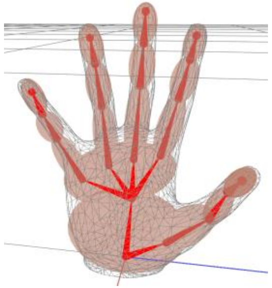
Idea: define complex shapes as the zero-crossing of a function

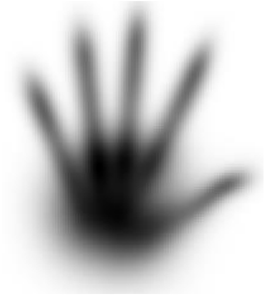
- use parametric function
  - e.g., mixtures of Gaussian distributions
    with position mu and covariance Sigma



$$\mathcal{C}(\mathrm{x}) = \sum_{i=1}^{n} \mathcal{G}_i(\mu_i, \Sigma_i)$$

contour line / zero crossing





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

  - a neural network?!

Implicit functions through NNs

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

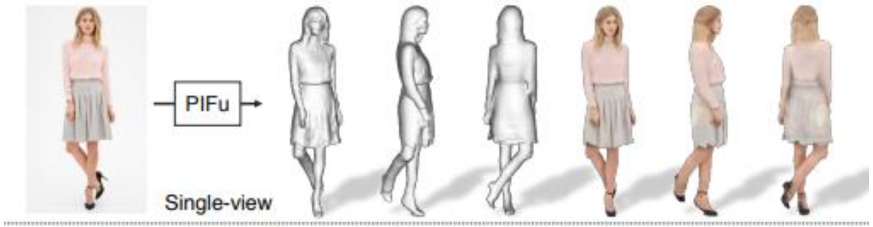
- a positive value for positions **inside** the object
- a negative value for positions **outside** the object
- reconstruct by querying a dense sampling

Advantage:

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

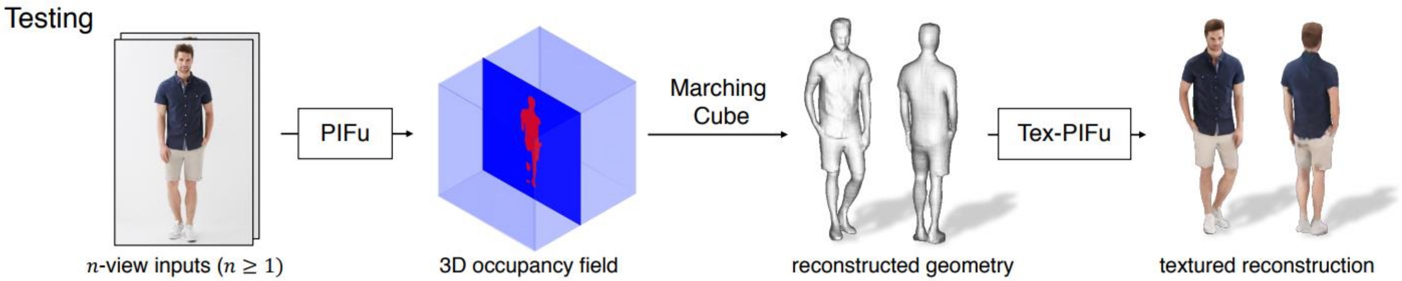
Disadvantage:

- Reconstruction requires many network evaluations, its slow!



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

Not straightforward to train… wait for the paper presentation



Surface mesh

Representation: Vertices connected by edges forming faces

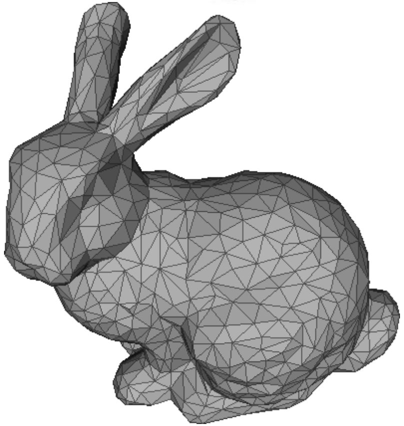
- Size: N x D + E x 2 (# points, space dimension, # edges)
- A 3D surface parametrization (can be higher-dimensional)
    - Piece-wise linear with adaptive detail; triangle faces are usual

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)



General graph convolution

- traditional 2D convolutions is convolution on a regular grid

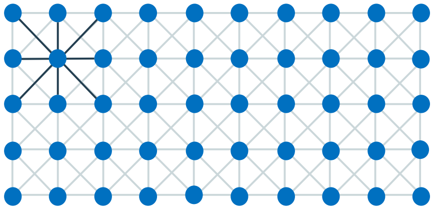
Difficulties for general graph convolution

- no notion of left/right and up/down
- different number of neighbors
- distances between nodes

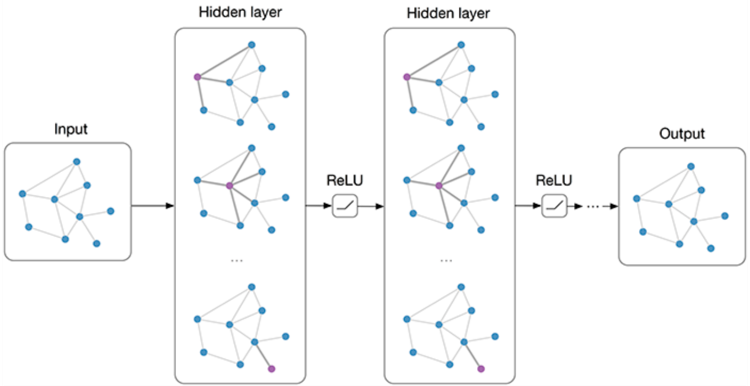
Solution

- per-node weight matrix for all nodes (like 1x1 conv.)
- weighted average over all neighbors (like average pooling)

$$h_i^{(l+1)} = \sigma\left(\sum_j \frac{1}{c_{ij}} h_j^{(l)} W^{(l)}\right)$$



Convolution on a regular grid



Graph convolution network

https://tkipf.github.io/graph-convolutional-networks/

Mesh Laplacian

Goal: A form of derivative on the mesh

Difficulty:

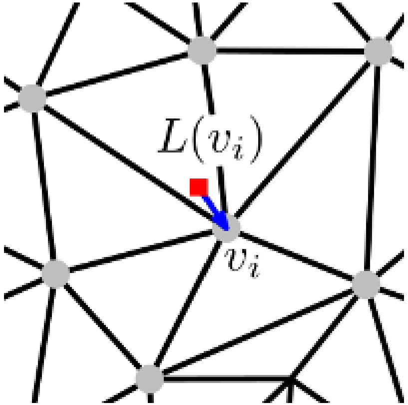
- irregularity, where is left/right/up/down?

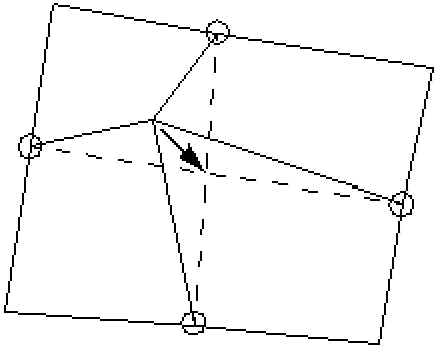
Solution

- (weighted) average over all neighboring nodes

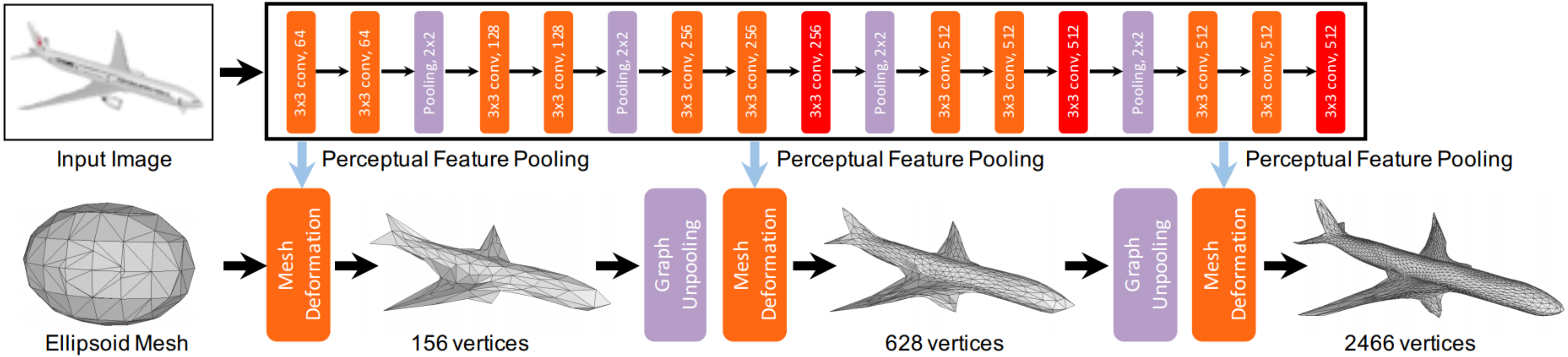
$$\mathscr{L}(\mathbf{v}_i) = \mathbf{v}_i - \frac{1}{d_i} \sum_{j \in \mathscr{N}_i} \mathbf{v}_j.$$

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





Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images

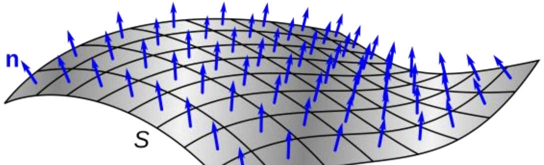


Desired:

- an output mesh that matches in position
  - Chamfer distance
- and has the same surface orientation
  - surface normal

$$l_n = \sum_p \sum_{q=\arg\min_q(\|p-q\|_2^2)} \|\langle p-k, \mathbf{n}_q\rangle\|_2^2$$

- … and follows a coarse-to-fine manner
  - minimize change of Laplacian between layers



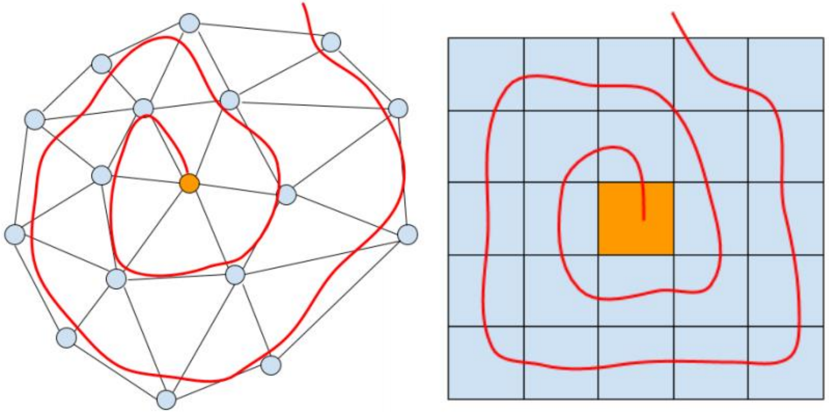
Spiral convolution

Goal: break the permutation invariance of neighbors

Idea: Order neighbors by simple rules

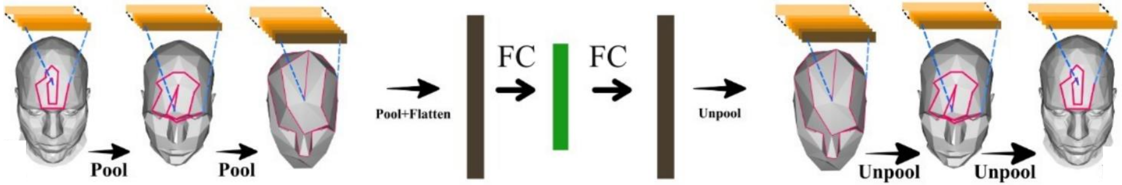
1. collect all neighbors (d hops in the graph)
2. pick the closest one (geodesic distance)
3. continue clockwise until spiral is of length k
4. multiply features h along spiral with weight matric

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



Advantages:

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



Keep it SMPL: Automatic Estimation of 3D Human Pose and Shape from a Single Image

Regression of SMPL parameters from images using deep learning

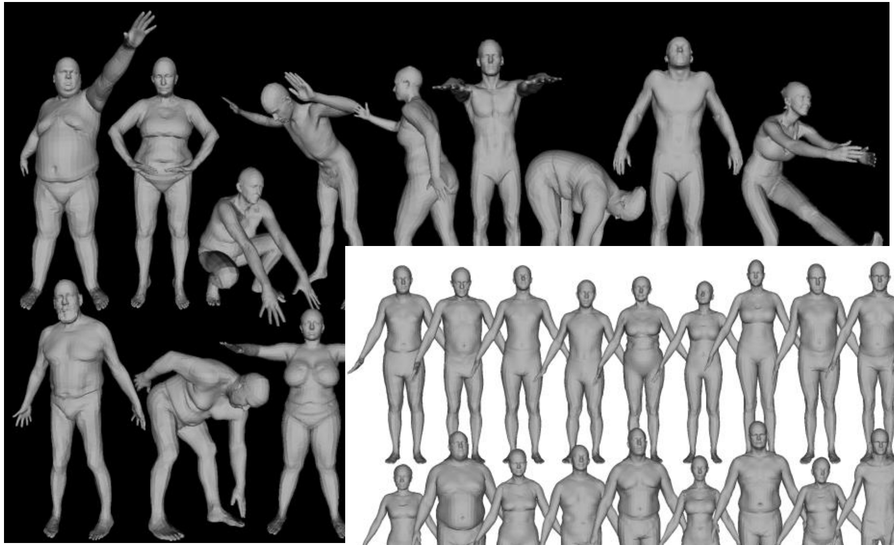
PCA body model

[Movie Reshape]

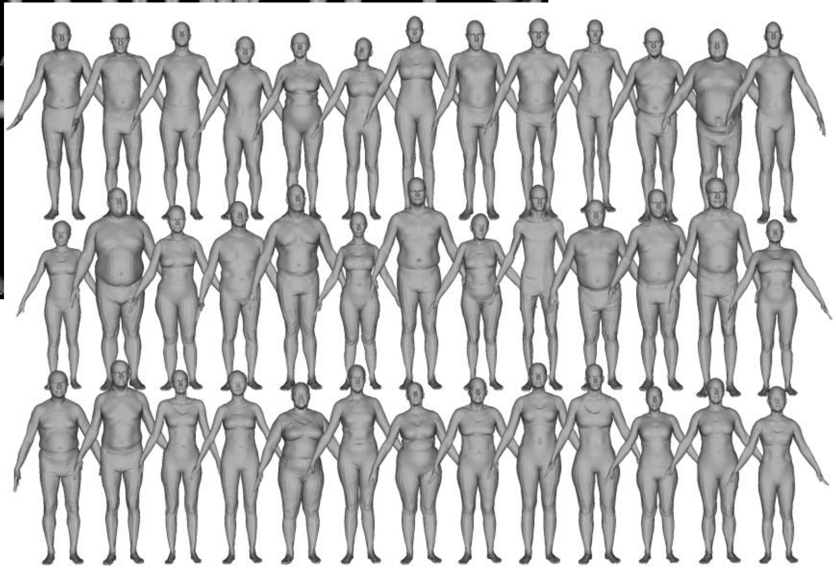
Body shape spaces

Data-driven model

- fitted to laser scans
- linear shape model
  - Principal Component Analysis (PCA)
- non-linear correction for articulation
  - corrective blend shapes (common in the CG community)
- Good for ‘naked’ body shape
- Hard to model clothing
  - too varied
  - topological changes (e.g. opening a jacket)



Pose and shape



Shape in canonical pose (registered)

Principal component analysis

- 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

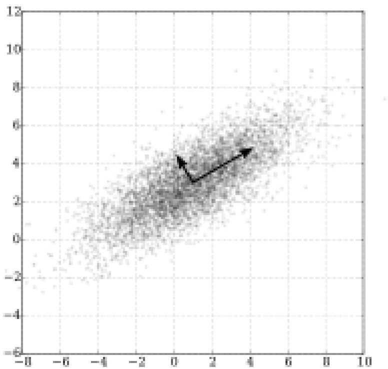
- First weight vector w(1)

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

  computed over all x(i) in the dataset
- … continue iteratively in orthogonal directions

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

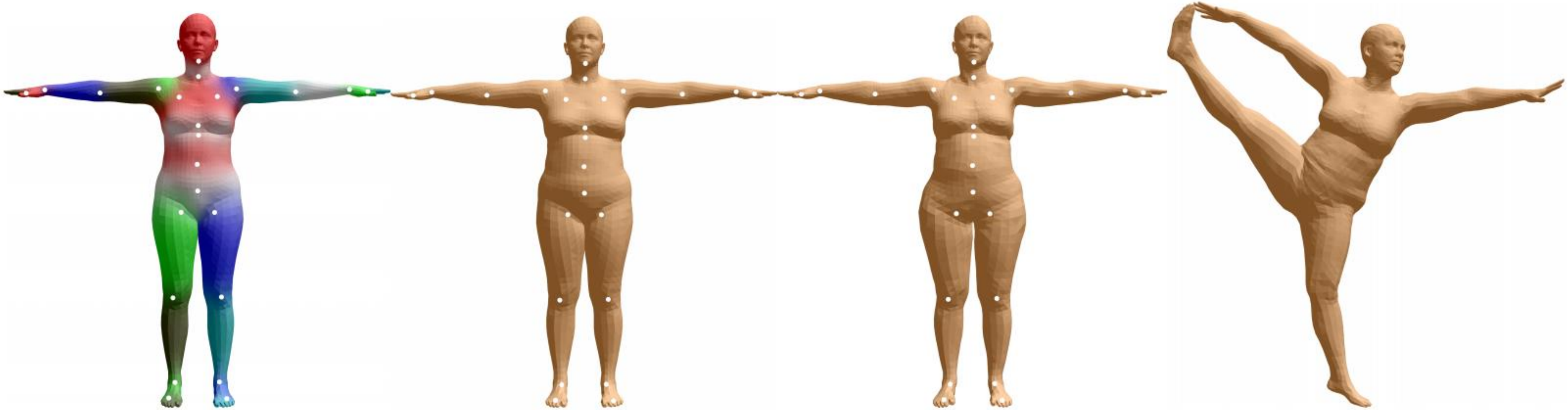
$$\underbrace{\hat{p} = W^\top}_{\text{reconstruction (decoding)}} \underbrace{W p}_{\text{projection (encoding)}}$$



two-dimensional space

thousand-dimensional space

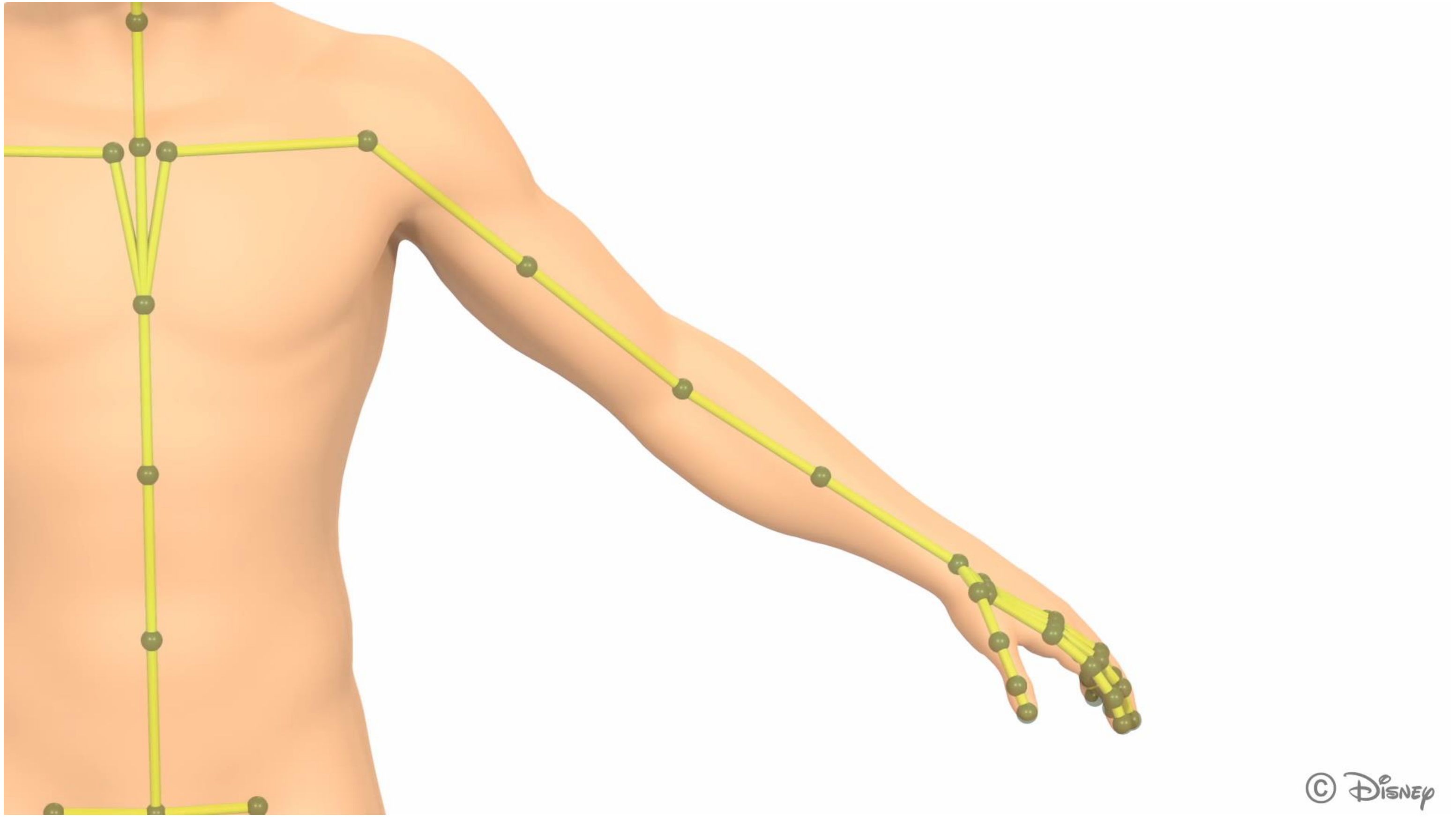
SMPL: A Skinned Multi-Person Linear Model



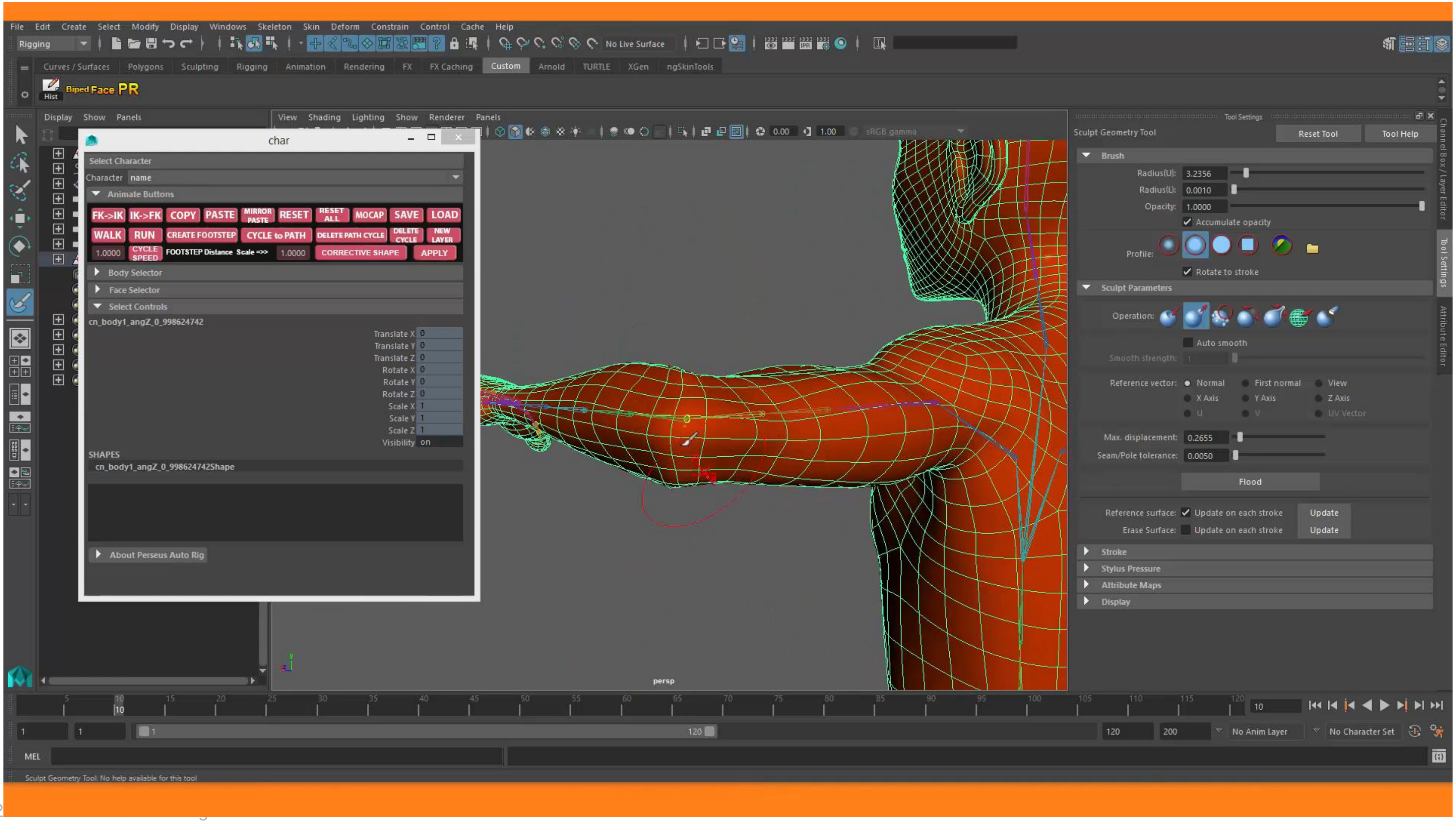
(a) $\bar{\mathbf{T}}$, $\mathcal{W}$ (b) $\bar{\mathbf{T}} + B_S(\vec{\beta})$, $J(\vec{\beta})$ (c) $T_P(\vec{\beta}, \vec{\theta}) = \bar{\mathbf{T}} + B_S(\vec{\beta}) + B_P(\vec{\theta})$ (d) $W(T_P(\vec{\beta}, \vec{\theta}), J(\vec{\beta}), \vec{\theta}, \mathcal{W})$

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.*

Skinning



Corrective blend shapes



Surface texture

Representation: A map that assigns a color to every point of a surface

- Size: W x H + N x 2 (W: width, H: height, N: #points for uv-coordinates)
- Dimensions: 2 D (embedded in 3D space via a mesh)
  - Discrete in space, continuous in color
- UV-coordinates attached to each mesh vertex define the spatial association

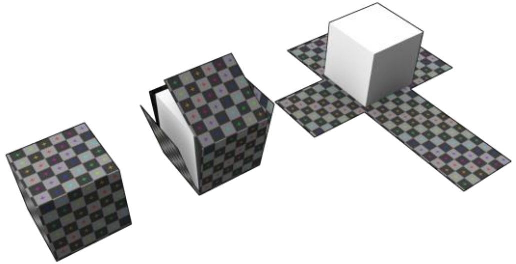
Benefits

- Appearance modelling for graphics and vision
  (e.g., rendering and reconstruction)
- Can carry more than color
  (shadowmaps, normal maps, **feature maps)**

Drawbacks

- Texture mapping (assigning vertices to texture map location) is hard
- Only a surface, not volumetric

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



UV mapping

- describe points on the texture with u,v coordinates
  - the horizontal and vertical position
- equip each vertex with the u,v coordinate
  - a 2D point

Example: teapot.obj

v -3.000000 1.800000 0.000000 *(vertex definition)*

v -2.991600 1.800000 -0.081000

…

vt 0.000100 0.000100 *(uv texture coordinates)*

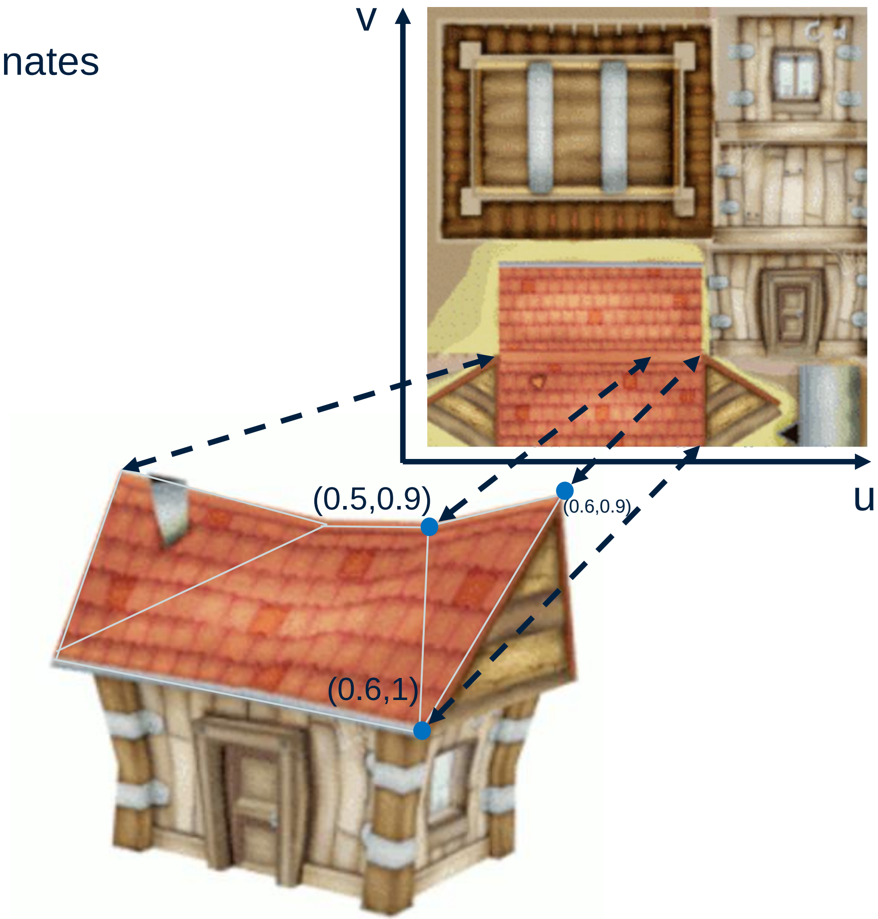
vt 0.999900 0.000100

…

f 1252 1248 1122 *(edges of a triangle/face)*

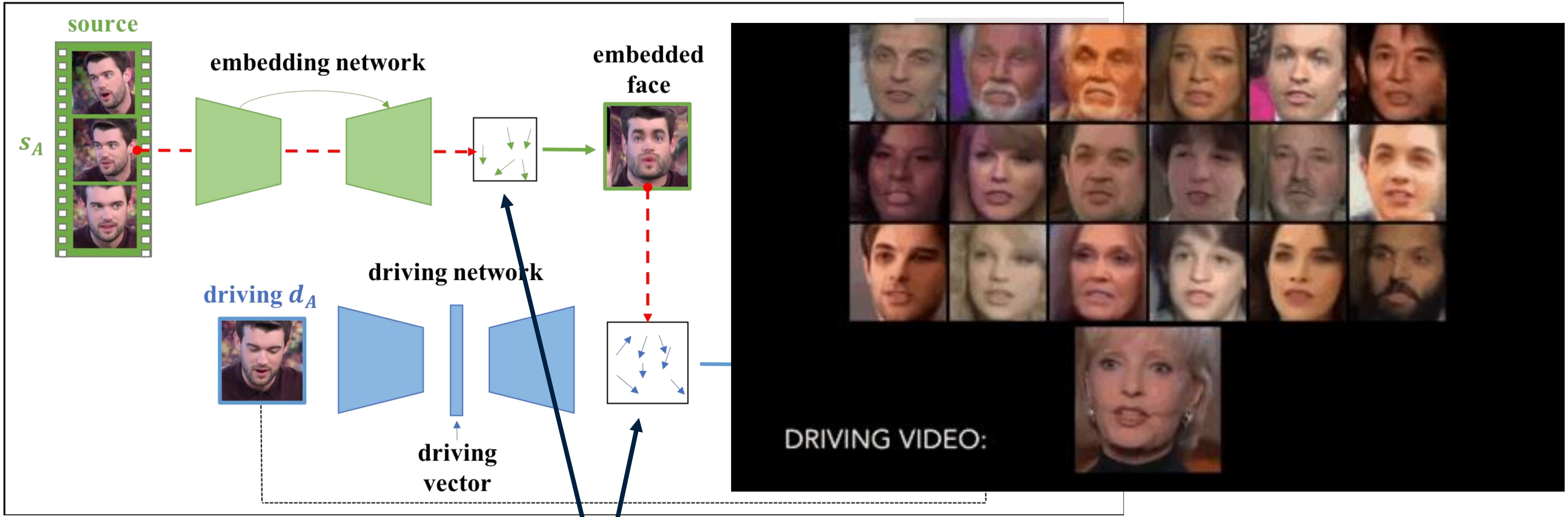
f 1027 1035 1133

…



Example: mapping a face to a texture

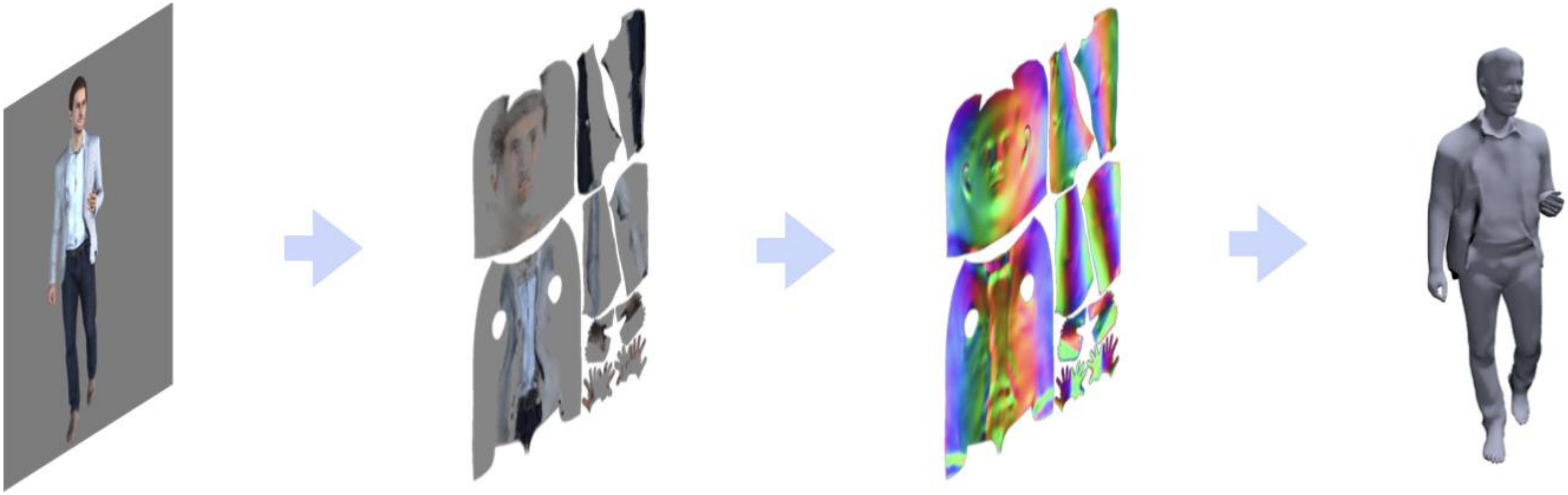
Wiles et al., X2Face: A network for controlling face generation by using images, audio, and pose codes



a form of uv mapping

Tex2Shape: Detailed Full Human Body Geometry From a Single Image

- Convolutional detail estimation via texture and normal maps

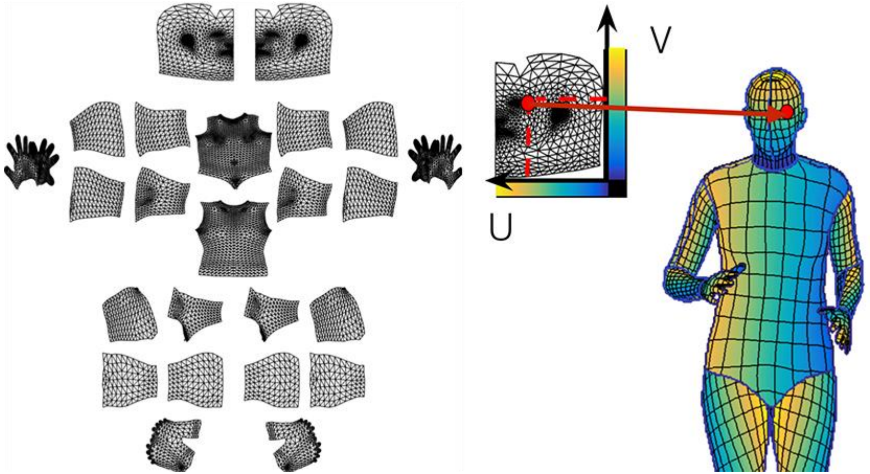


Dense Pose: Dense Human Pose Estimation In The Wild

Issue: Heatmap representations don't generalize well to many points (one map per point)

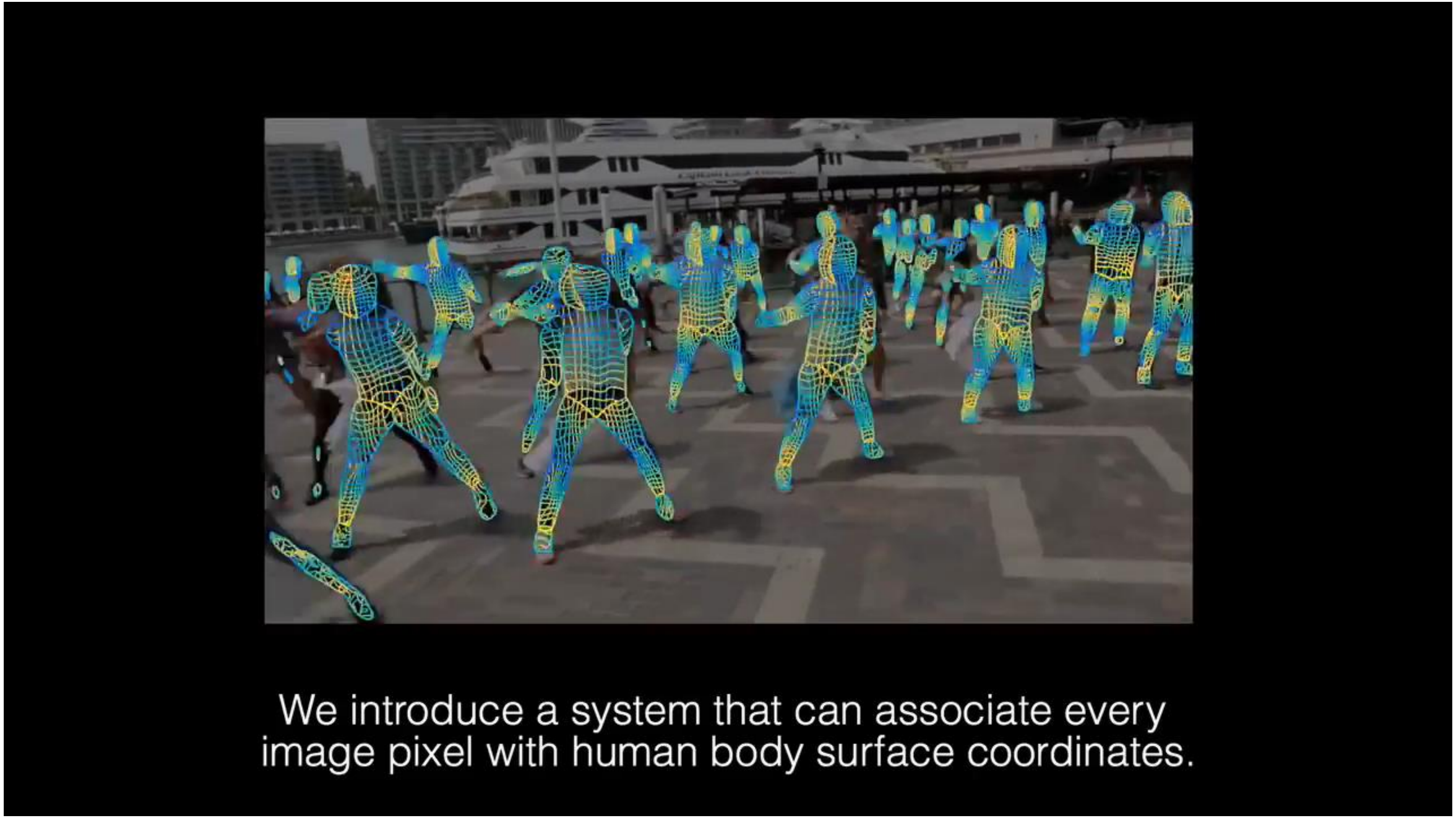
Idea: Encode locations as continuous value

- as u,v coordinates
- generalizes well to multiple people



[Dense Pose: Dense Human Pose Estimation In The Wild]

Dense Pose results



3D ‘uv coordinates’

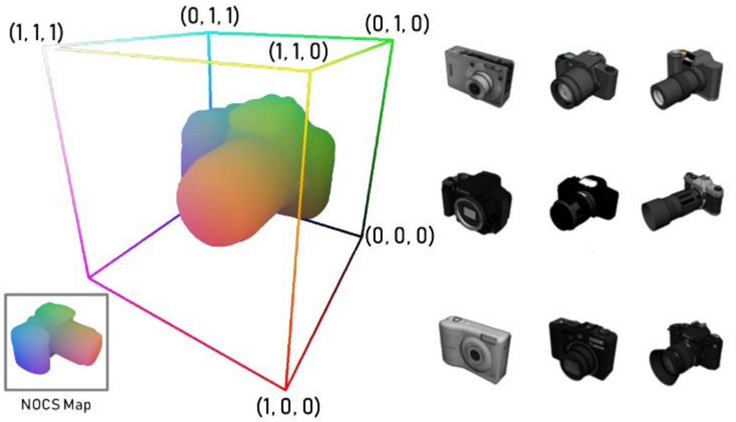
Idea: Learn to map to 3D coordinates

Solution:

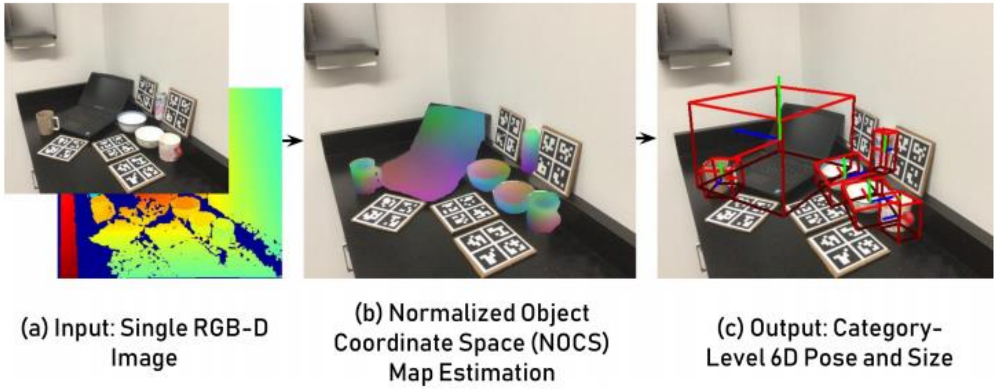
- a generalization of uv-coordinates in 3D

Benefits:

- compact, continuous, accurate



[Normalized Object Coordinate Space for Category-Level 6D Object Pose and Size Estimation]

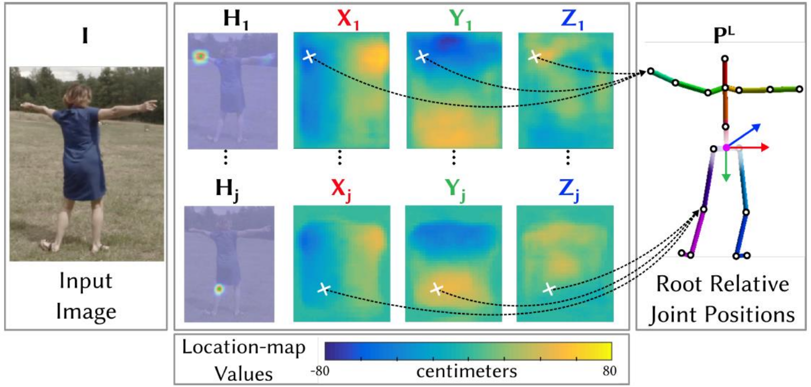


Location maps

Idea: Predict 3D pose in a convolutional manner

Implementation:

1. predict three location maps alongside the heatmap H
    - respectively one for the x,y,z positon
2. retrieve the arg max of the heatmap (2D joint location)
3. Read out the x,y,z maps at the predicted 2D location

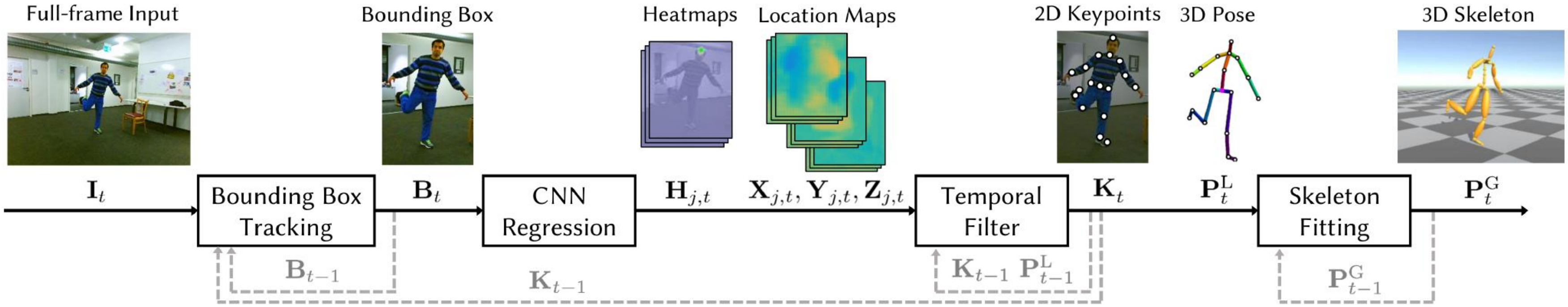


Admantages:

- fully convolutional networks, which apply to varying image resolution
- (convolutional) operations are centered around the area of interest (joints)
- generalized well to multiple persons

VNect: Real-time 3D Human Pose Estimation with a Single RGB Camera

- Using location maps
- A combination of feed forward prediction with NNs and optimization of skeleton parameters



Hidden questions