CMSC740
Advanced Computer Graphics

Matthias Zwicker

Fall 2025

Rendering avatars

- Goal
  - Construct avatar model (virtual human) from videos (using as little data as possible) of real person
  - Render avatars from any camera viewpoint, in arbitrary body pose
- Applications
  - Games, movies, AR/VR/XR
- Approach: deformable NeRF, custom deformation model to approximate humans

Approach

- Goal: Construct rendering function *f* for avatars

  image = *f*(camera parameters, body pose, body shape parameters, body appearance parameters)

- Training: inverse rendering, optimize input parameters (camera parameters, body pose, body shape parameters, body appearance parameters) to match ground truth image data (videos of moving person, from one or multiple viewpoints)
- Rendering: given trained shape, body appearance parameters, evaluate *f* (render) for arbitrary new camera parameters, body poses

Approximate geometry

- Goal: take advantage of known, rough shape of humans
- Construct approximate 3D geometry using body pose, body shape parameters
- Rendering function using approximate geometry

  image = *f*(camera parameters, ***geometry***, body appearance parameters)

  geometry = ***g***(body pose, body shape parameters)

- Two components
  - SMPL model for geometry ***g***
  - Extended NeRF for appearance *f*

Recent work

- HVTR: Hybrid Volumetric-Textural Rendering for Human Avatars, 3DV 2022
  https://www.cs.umd.edu/~taohu/hvtr/
- Neural actor: neural free-view synthesis of human actors with pose control, ACM TOG 2021
  https://vcai.mpi-inf.mpg.de/projects/NeuralActor/
- And many more…

Approximate geometry: SMPL

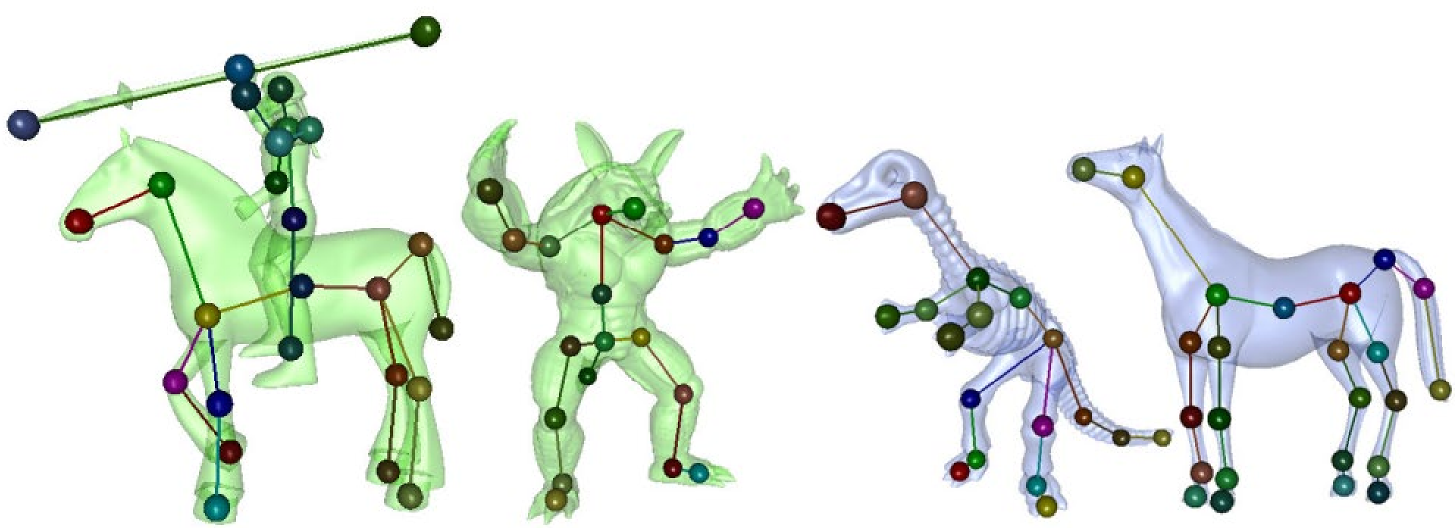
- SMPL: A Skinned Multi-Person Linear Model, ACM TOG 2015 https://smpl.is.tue.mpg.de/
- Parametric model for pose- and body shape-dependent geometry, represented as mesh

geometry = SMPL(body pose, body shape parameters)

SMPL geometries (meshes) for different pose, shape parameters

SMPL

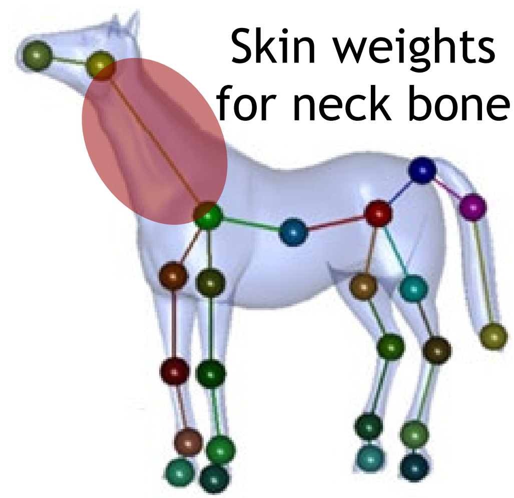
- Based on idea of skeletal animation http://en.wikipedia.org/wiki/Skeletal_animation
- Equip ("rig") manually created template mesh with skeleton
  - Joints connected to each other via rigid parts ("bones"), https://en.wikipedia.org/wiki/Kinematic_chain)
- Drive mesh deformation using skeleton
  - Mesh is attached to skeleton like skin
  - Skeleton pose given by set of rigid transformations, one per bone
  - Simple mathematical formulation
- Main method to produce CG character animation, also used in computer vision for character pose estimation



Meshes rigged with skeletons http://www.geometry.caltech.edu/pubs/SZTDBG07.pdf

Rigged template mesh

- Rigging: given template mesh in reference pose, construct skeleton and association of mesh with skeleton
- Rig consists of two parts
  - Skeleton
  - Skin weights: influence of each bone on each surface point



Skin weights for neck bone

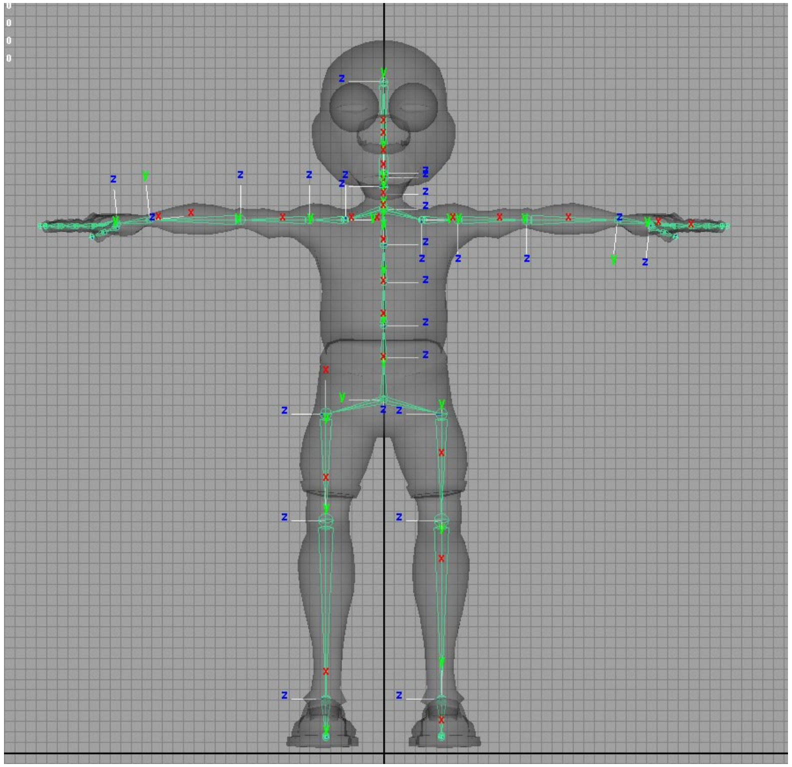
SMPL template mesh with joint locations, skin weights for each bone

Rigging

- Often manually
- Automatic methods exist
  "Automatic rigging and animation of 3D characters", Baran et al., https://dl.acm.org/citation.cfm?id=1276467
- Commercial software often specialized for human characters
  http://www.mixamo.com/

Rigging

- Skeleton in reference pose

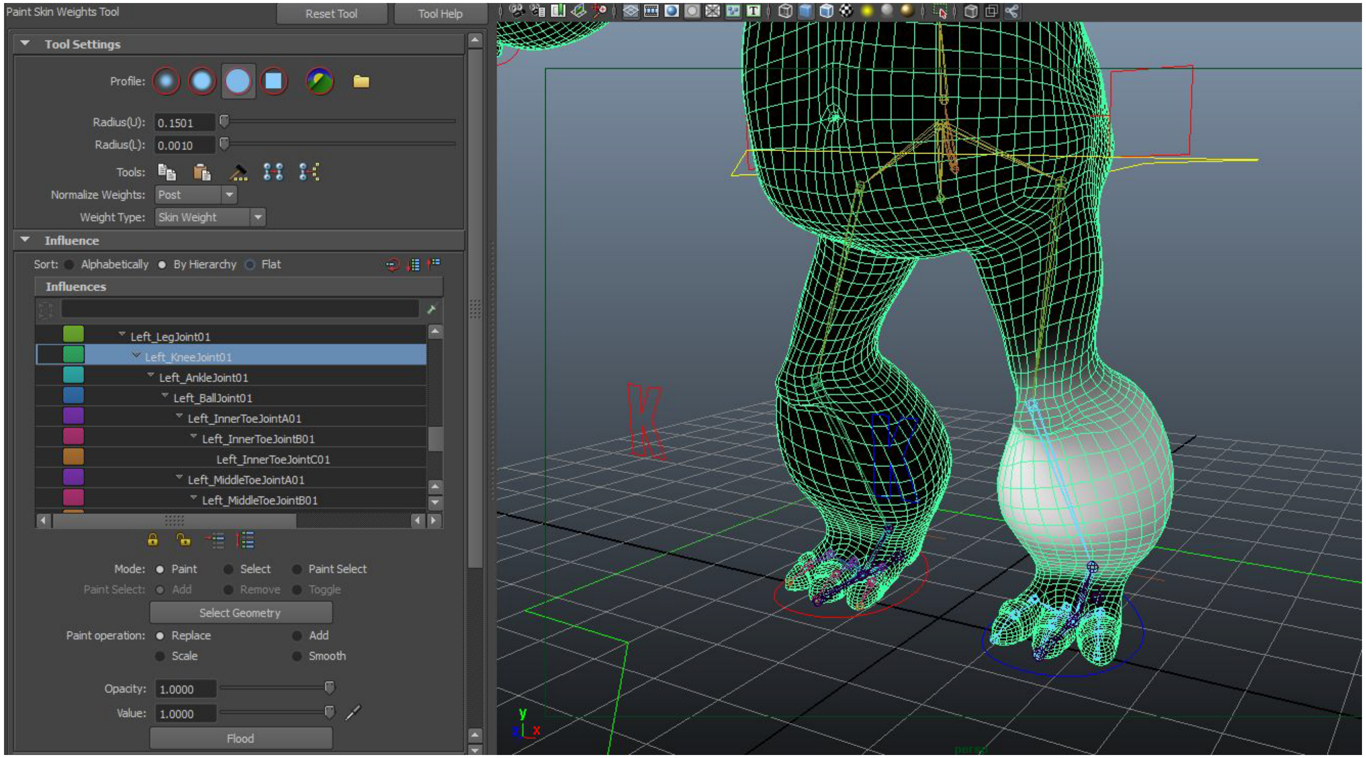


Skeleton constructed in Autodesk Maya

https://www.cs.washington.edu/education/courses/458/07au/projects/project6/rigging_tutorial/skeleton.htm

Rigging

- Skin weights: visualizes for each bone how much it influences each point on mesh



Weight painting in Autodesk Maya

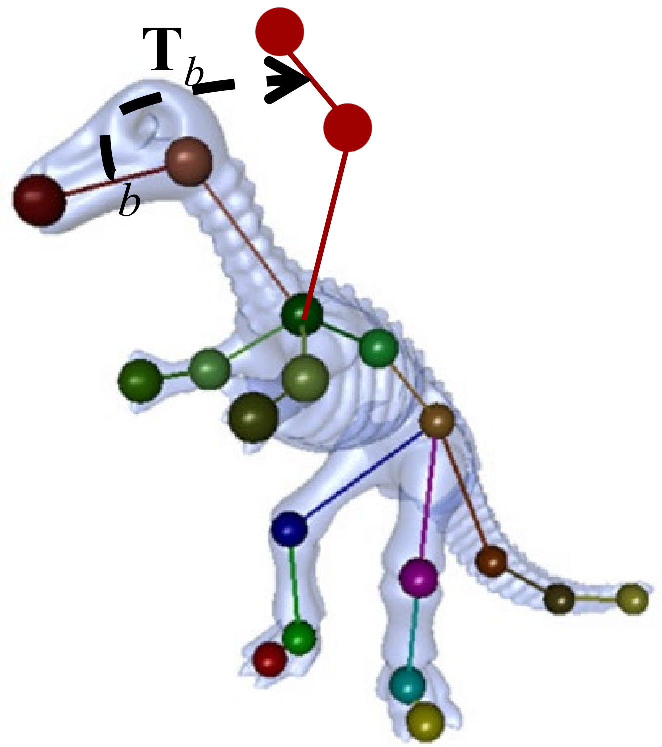
http://jamesrburr.wordpress.com/2012/01/15/rigging-the-goblin-character/

Skin deformation model

- Template mesh (surface): vertices in reference pose $\bar{\mathbf{x}}_i$
- Deformed vertices $\mathbf{x}_i$
- Bone weights w_{bi} of bone b for vertex i (matrix of size #bones x #vertices)
- Transformation matrix $\mathbf{T}_b$ of bone b relative to reference pose

$$\mathbf{x}_i = \Big(\sum_{b \in \text{bones}} w_{bi} \mathbf{T}_b \Big) \bar{\mathbf{x}}_i$$

Locally weighted
rigid transformations
("skeletal animation",
https://en.wikipedia.org/wiki/Skeletal_animation
"linear blend skinning")



Template mesh/reference pose and transformation of a bone

Forward kinematics

- Bone transformation matrices $\mathbf{T}_b$ can be computed based on skeletal joint angles
  https://en.wikipedia.org/wiki/Forward_kinematics
- Fix "root" joint, compute transformations relative to root step-by-step along kinematic chain using joint angles
  - Bone transformation is multiplication of transformation matrices for each steps along kinematic chain from root to bone
- Note: better to use dual quaternions rather than transformation matrices $\mathbf{T}_b$
  - Dual quaternions: representation of rigid motions such that weighted averaging "makes sense", has desirable properties https://users.cs.utah.edu/~ladislav/dq/index.html

$$\mathbf{x}_i = \left(\sum_{b \in \text{bones}} \boxed{w_{bi}\mathbf{T}_b} \right) \bar{\mathbf{x}}_i$$

Replace with dual quaternion blending

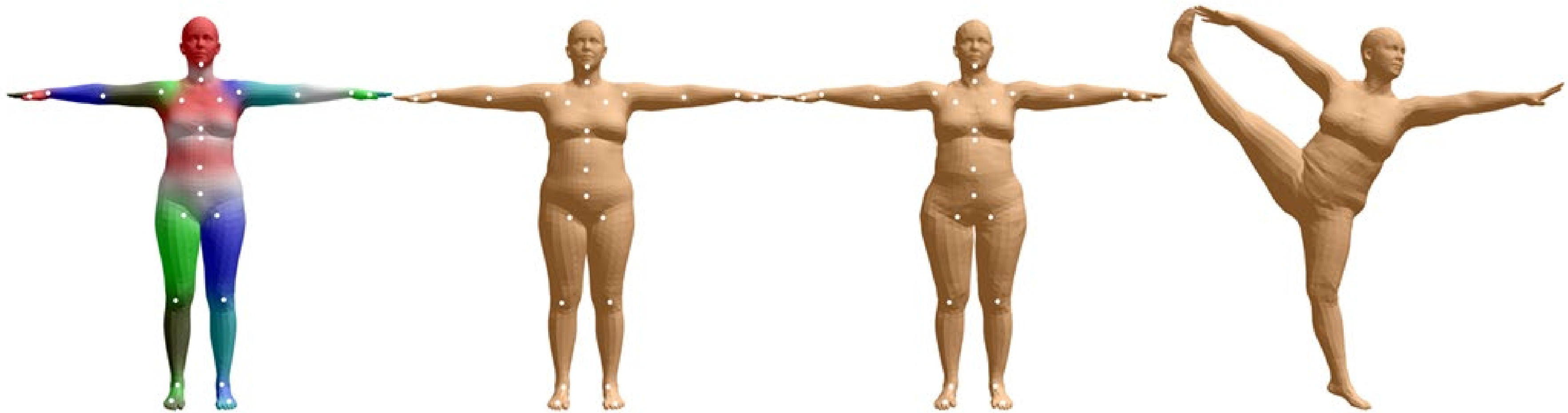
So far

- Skin deformation model (linear blend skinning) for generic template body shape (mesh) provides pose-dependent geometry function

  geometry = LinearBlendSkinning(body pose parameters, template mesh, bone weights)

  where body pose parameters is set of joint angles
- Limitations
  - Skin deformation model cannot match detailed deformed body shapes
  - Single template mesh, no identity-specific shape

- Extensions of linear blend skinning



Template with joints, blend weights | Identity-specific shape details | Pose-specific shape details | Final geometry using linear blend skinning

Shape representation

- Template mesh unrolled into vector of xyz vertex positions $\bar{\mathbf{T}} \in \mathbb{R}^{3N}$
- Number of vertices $N = 6890$
- Skinning blend weights given by matrix $\mathcal{W} \in \mathrm{R}^{N \times K}$
  - Number of joints $K = 23$
- Identify specific and pose-dependent details given as $\mathbf{R}^{3N}$ vectors storing 3D offset for each template vertex, called blend shape vectors
  - Identity specific blend shape vector $B_S \in \mathbf{R}^{3N}$
  - Pose-dependent blend shape vector $B_P \in \mathbf{R}^{3N}$

SMPL

- Extensions of linear blend skinning

Template with K joints,
blend weights W

$$\bar{\mathbf{T}} \in \mathbb{R}^{3N}$$

$$\mathcal{W} \in \mathrm{R}^{N \times K}$$

$N = 6890$, K=23

SMPL

- Extensions of linear blend skinning

Template with K joints, blend weights W

$$\bar{\mathbf{T}} \in \mathbb{R}^{3N}$$

$$\mathcal{W} \in \mathrm{R}^{N \times K}$$

$N = 6890$, K=23

Identity-specific shape details

$$\bar{\mathbf{T}} + B_S$$

SMPL

- Extensions of linear blend skinning

Template with K joints, blend weights W

$$\bar{\mathbf{T}} \in \mathbb{R}^{3N}$$

$$\mathcal{W} \in \mathrm{R}^{N \times K}$$

$N = 6890$, $K=23$

Identity-specific shape details

$$\bar{\mathbf{T}} + B_S$$

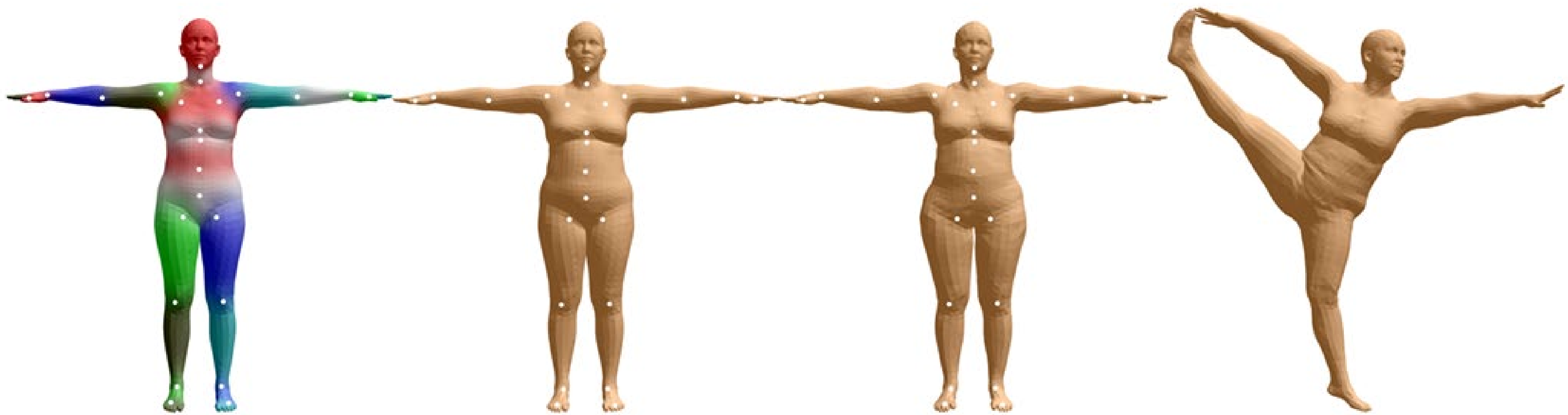
Pose-specific shape details

$$\bar{\mathbf{T}} + B_S + B_P(\vec{\theta})$$

Vector of 3x23 joint angles $\vec{\theta}$

SMPL

- Extensions of linear blend skinning



Template with K joints, blend weights W

$$\bar{\mathbf{T}} \in \mathbb{R}^{3N}$$

$$\mathcal{W} \in \mathrm{R}^{N \times K}$$

$N = 6890$, K=23

Identity-specific shape details

$$\bar{\mathbf{T}} + B_S$$

Pose-specific shape details

$$\bar{\mathbf{T}} + B_S + B_P(\vec{\theta})$$

Vector of 3x23 joint angles $\vec{\theta}$

Final geometry using linear blend skinning

SMPL Training

- Goal: Given multi-view images of person, fit SMPL parameters to match input views (or even just single view)
- Problem: too many parameters, ill defined if only few input views
- Approach: use large data set of 3D scans to pre-train entire model first, then apply to new data
  - Split model parameters into two sets

1. First set to be pre-trained only on 3D data set of many identities and poses, then fixed

   $\mathcal{W} \in \mathbb{R}^{N \times K} \quad \bar{\mathbf{T}}, B_P(\vec{\theta}) \in \mathbb{R}^{3N}$

2. Identity specific shape details B_S to be fit to training and new data

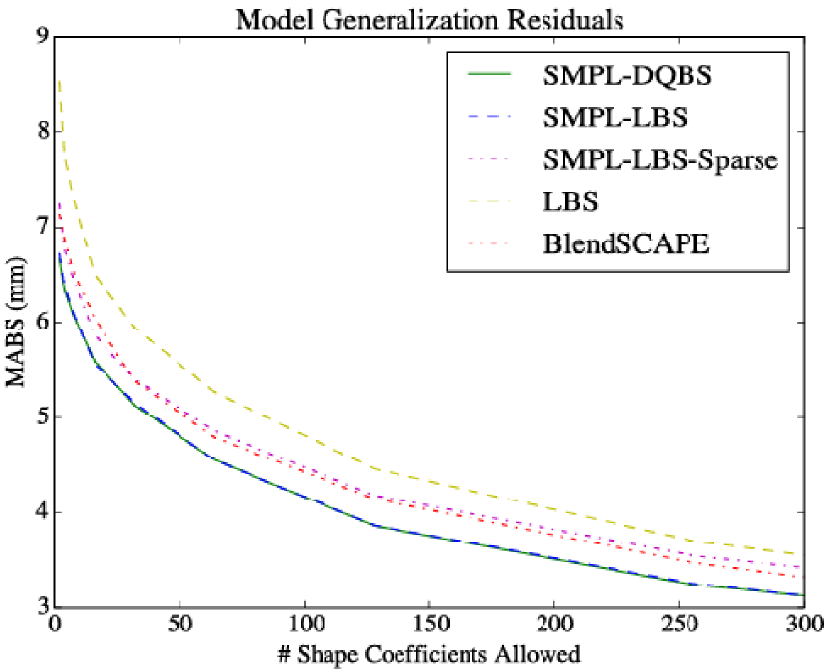
Identity-specific blend shapes B_S

- Split into pre-trained parameters and parameters that will be fit to new data
  - Linear model

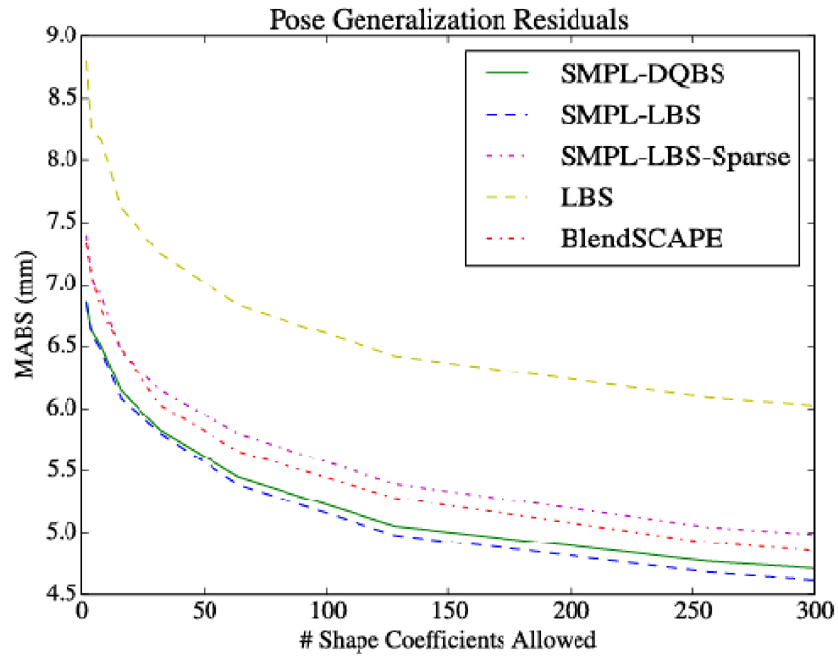
$$B_S(\vec{\beta}; \mathcal{S}) = \sum_{n=1}^{|\vec{\beta}|} \beta_n \mathbf{S}_n \qquad \mathbf{S}_n \in \mathbf{R}^{3N}$$

  - Pre-trained "basis vectors" $\mathbf{S}_n$
  - Shape coefficients (scalar weights) β_n, will be fit to new data
  - Number of shape coefficients $|\vec{\beta}|$

Number of shape coefficients $|\vec{\beta}|$



$|\vec{\beta}|$



$|\vec{\beta}|$

DQBS: dual quaternion blend skinning
LBS: linear blend skinning

Pose-dependent blend shapes B_P

- Linear function $$B_P(\vec{\theta};\mathcal{P}) = \sum_{n=1}^{9K} (R_n(\vec{\theta}) - R_n(\vec{\theta}^*))\mathbf{P}_n$$
- Pre-trained vectors $\mathbf{P}_n \in \mathbf{R}^{3N}$ (#vertices N)
- Function R maps 23 joint angles $\vec{\theta}$ to vector consisting of 207 elements of all joint rotation matrices (K=23 joint matrices x 9 elements)
  – R_n is n-th element in R
  – R computed using Rodrigues' formula
  https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula
  - Joint angles of rest pose $\vec{\theta}^*$

Pre-training

- Using data set of thousands of 3D scans, registered to template mesh topology
  - Same mesh structure as template for all scans, 1-1 vertex correspondence
- Objective: optimize SMPL parameters (template mesh vertices $\mathbf{T}$, blend weights $\mathcal{W}$, identity blend shapes $\mathbf{S}_n$, shape coefficients β_n, pose dependent blend shapes $\mathbf{P}_n$) to minimize vertex registration error
  - Loss for each 3D scan: each vertex of SMPL model needs to match corresponding vertex in 3D scan
- Details of optimization see paper https://smpl.is.tue.mpg.de/

Note

- Description here omitted pose dependent joint locations (details see paper)

Applications

- Character animation for graphics (plugins available for many standard graphics tools, Maya, Blender, Unreal engine)
  https://smpl.is.tue.mpg.de/
- Motion capture by fitting SMPL model to multiple camera views
- Fitting SMPL model to single view video using keypoints
  https://github.com/zju3dv/EasyMocap

Neural Actor

- Neural Actor: Neural Free-view Synthesis of Human Actors with Pose Control, ACM TOG 2021, https://vcai.mpi-inf.mpg.de/projects/NeuralActor/
- Goal: Learn rendering function for virtual human from videos of person
  - Using approximate geometry

    image = *f*(camera parameters, *geometry*, body appearance parameters)
    geometry = *g*(body pose, body shape parameters)

- SMPL model for geometry g
- Extended NeRF for rendering f

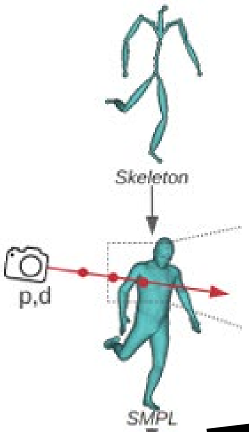
Neural actor

Learn person-specific rendering function to match ground truth views

f(camera parameters, *geometry*, body appearance parameters)
geometry = *g*(body pose, body shape parameters)

Train *f* using inverse rendering
Geometry given by SMPL

Neural actor

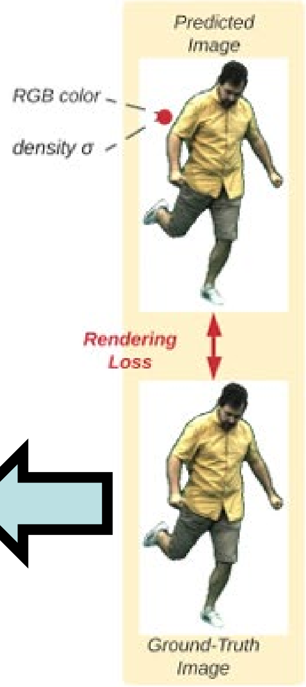


SMPL

geometry = *g*(body pose, body shape parameters)

body pose parameters inferred from ground truth images using separate techniques, e.g.

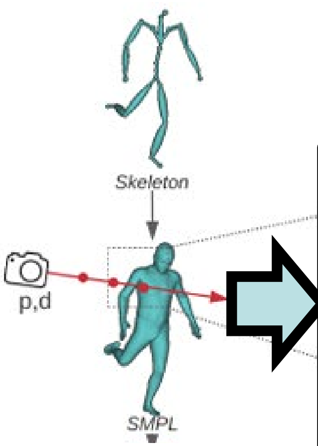
https://arxiv.org/pdf/2204.12484.pdf



Neural actor

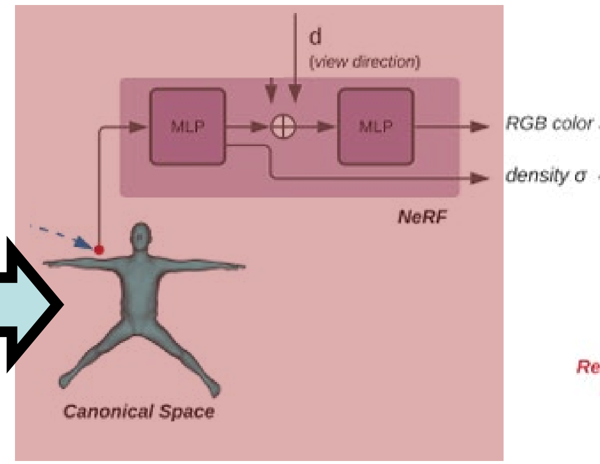
- “Inverse deformation” to canonical space

NeRF in canonical space



Volumetric samples for NeRF rendering

Map sample points to canonical space (fixed pose using reference joint angles) by inverting linear blend skinning transformations

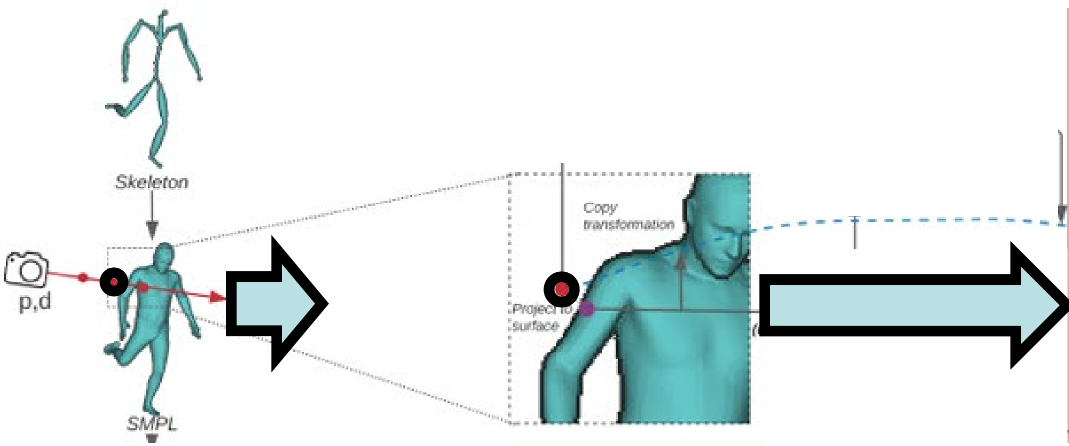


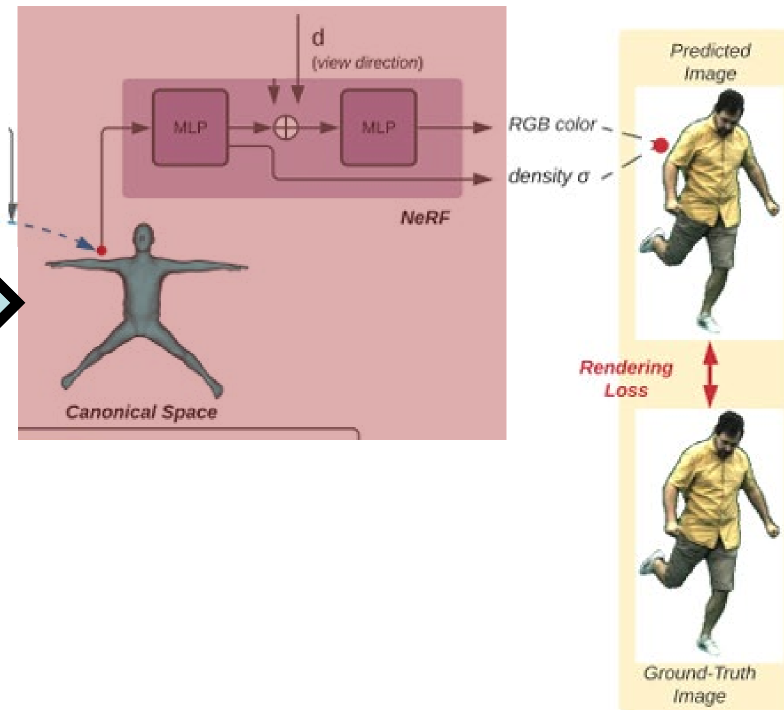
Same idea as “Nerfies” (deformable neural radiance fields), but with specialized deformation model for human bodies

Neural actor

- “Inverse deformation” to canonical space

NeRF in canonical space



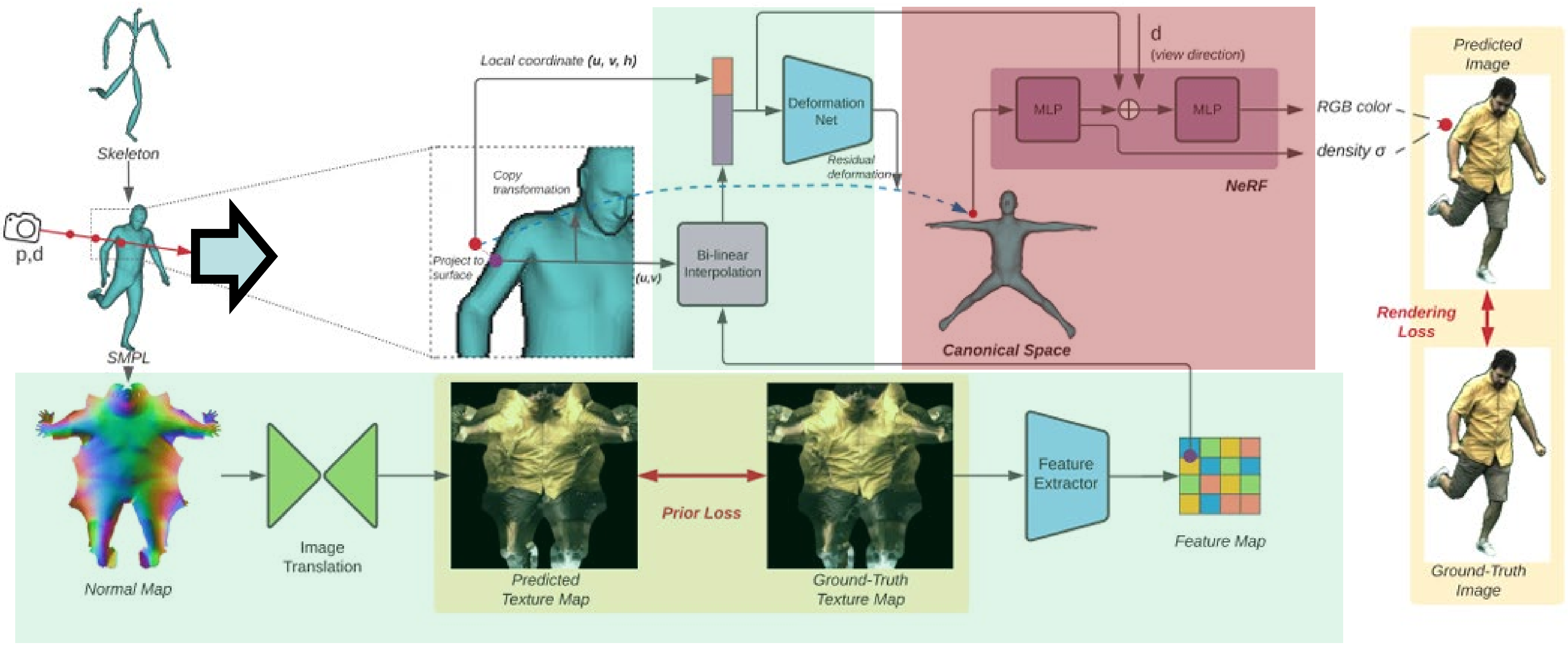


Volumetric samples for NeRF rendering

- Skinning weights only defined on surface
- Use weights given by point on surface closest to sample point (project sample point to surface)

Neural actor

NeRF in canonical space



Texture space features to learn detail deformations, improve NeRF rendering

Training, results, discussion

- Person-specific multi-view video data sets, 11-12 cameras, approx. 30,000 training frames
- Results https://vcai.mpi-inf.mpg.de/projects/NeuralActor/
- Limitations?
- Related work
  - Fast, high quality rendering https://taohuumd.github.io/projects/hvtrpp/
  - Including dynamic motion https://github.com/TaoHuUMD/SurMo?tab=readme-ov-file