SMPL 코드 분석

Review

Model generation functions

W	$\triangleq$	Skinning function
M	$\triangleq$	SMPL function
B_P	$\triangleq$	Pose blendshapes function
B_S	$\triangleq$	Shape blendshapes function
B_D	$\triangleq$	Dynamic blendshapes function
J	$\triangleq$	Joint regressor: Predicts joints from surface

Model input parameters (controls)

$\vec{\beta}$	$\triangleq$	Shape parameters
$\vec{\theta}$	$\triangleq$	Pose parameters
$\vec{\omega}$	$\triangleq$	Scaled axis of rotation; the 3 pose parameters corresponding to a particular joint
$\vec{\phi}$	$\triangleq$	Dynamic control vector
$\vec{\delta}$	$\triangleq$	Dynamic shape coefficients
$\vec{\theta}^*$	$\triangleq$	Zero pose or rest pose; the effect of the pose blendshapes is zero for that pose

Model parameters (parameters learned)

$\mathcal{S}$	$\triangleq$	Shape blendshapes
$\mathcal{P}$	$\triangleq$	Pose blendshapes
$\mathcal{W}$	$\triangleq$	Blendweights
$\mathcal{J}$	$\triangleq$	Joint regressor matrix
$\bar{\mathbf{T}}$	$\triangleq$	Mean shape of the template

Review

skinning function

model term

joint term

blend weight

$$\boxed{W}\left(\boxed{T_P(\vec{\beta},\vec{\theta};\bar{\mathbf{T}},\mathcal{S},\mathcal{P})},\boxed{J(\vec{\beta};\mathcal{J},\bar{\mathbf{T}},\mathcal{S})},\vec{\theta},\mathcal{W}\right)$$

transform matrix

$$\bar{\mathbf{t}}'_i = \sum_{k=1}^{K} w_{k,i}\boxed{G'_k(\vec{\theta},\mathbf{J})}\bar{\mathbf{t}}_i$$

rest pose template

shape blend shapes

Pose blend shapes

$$T_P(\vec{\beta},\vec{\theta}) = \bar{\mathbf{T}} + \boxed{B_S(\vec{\beta})} + \boxed{B_P(\vec{\theta})}$$

joint regressor matrix

$$J(\vec{\beta};\mathcal{J},\bar{\mathbf{T}},\mathcal{S}) = \mathcal{J}(\bar{\mathbf{T}} + B_S(\vec{\beta};\mathcal{S}))$$

$$G'_k(\vec{\theta},\mathbf{J}) = \boxed{G_k(\vec{\theta},\mathbf{J})}G_k(\vec{\theta}^*,\mathbf{J})^{-1}$$

$$G_k(\vec{\theta},\mathbf{J}) = \prod_{j\in A(k)}\left[\begin{array}{c|c}\exp(\vec{\omega}_j) & \mathbf{j}_j \\ \hline \vec{0} & 1\end{array}\right]$$

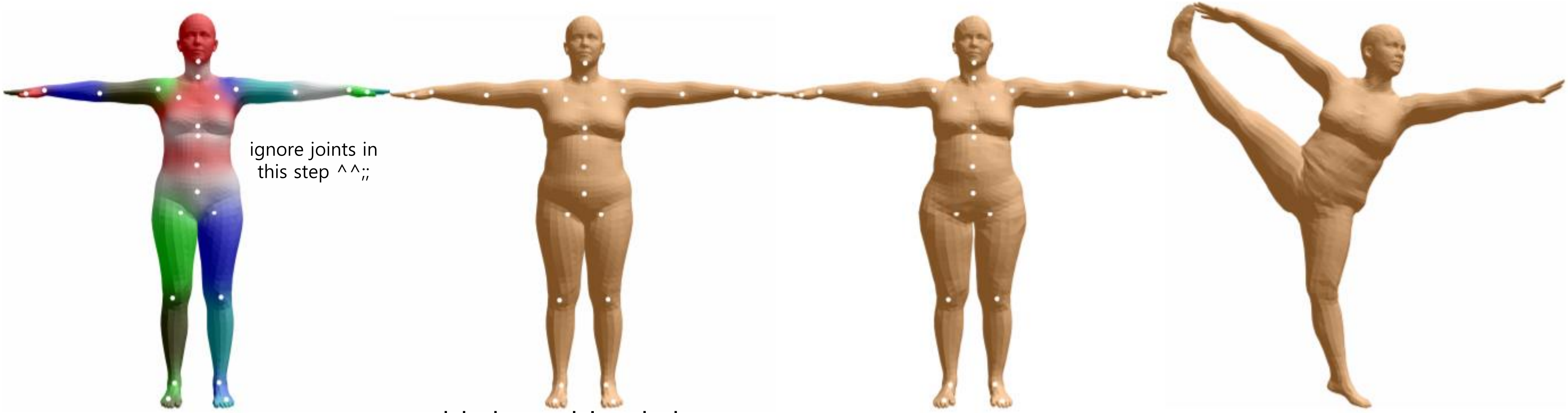
$$B_S(\vec{\beta};\mathcal{S}) = \sum_{n=1}^{|\vec{\beta}|}\beta_n\mathbf{S}_n$$

principal components of shape displacements

rodrigues rotation matrix

$$B_P(\vec{\theta};\mathcal{P}) = \sum_{n=1}^{9K}(R_n(\vec{\theta}) - R_n(\vec{\theta}^*))\mathbf{P}_n$$

Pipeline



rest pose template and blend weight

$$\bar{\mathbf{T}}$$

$$\mathcal{W}$$

add shape blend shapes and calculate joint location

$$\bar{\mathbf{T}} + B_S(\vec{\beta};\mathcal{S})$$

$$\mathcal{J}(\bar{\mathbf{T}} + B_S(\vec{\beta};\mathcal{S}))$$

$$\mathcal{W}$$

add pose blend shapes

$$\bar{\mathbf{T}} + B_S(\vec{\beta};\mathcal{S}) + B_P(\vec{\theta};\mathcal{P})$$

$$\mathcal{J}(\bar{\mathbf{T}} + B_S(\vec{\beta};\mathcal{S}))$$

$$\mathcal{W}$$

$$W\left(T_P(\vec{\beta},\vec{\theta};\bar{\mathbf{T}},\mathcal{S},\mathcal{P}),\ J(\vec{\beta};\mathcal{J},\bar{\mathbf{T}},\mathcal{S}),\vec{\theta},\mathcal{W}\right)$$

$$\bar{\mathbf{t}}'_i = \sum_{k=1}^{K} w_{k,i} G'_k(\vec{\theta},\mathbf{J})\bar{\mathbf{t}}_i$$

Code: body_models.py

- class:
  - SMPL (, SMPLLayer)
  - SMPLH (, SMPLLayer)
  - SMPLX (, SMPLXLayer)
  - MANO (, MANOLayer)
  - FLAME (, FLAMELayer)

MODEL : nn.Module

__init__(model, pretrained parameters, betas, pose, …)

…

…

forward(betas=None, pose=None, …)

…

vertices, joints = lbs(…)

…

return: vertices, pose, joints, betas

```
vertices, joints = lbs(betas, full_pose, self.v_template,
                       self.shapedirs, self.posedirs,
                       self.J_regressor, self.parents,
                       self.lbs_weights, pose2rot=pose2rot)
```

Code: body_models.py>SMPL>forward()

```python
vertices, joints = lbs(betas, full_pose, self.v_template,
                       self.shapedirs, self.posedirs,
                       self.J_regressor, self.parents,
                       self.lbs_weights, pose2rot=pose2rot)
```

```python
def lbs(
    betas: Tensor,
    pose: Tensor,
    v_template: Tensor,
    shapedirs: Tensor,
    posedirs: Tensor,
    J_regressor: Tensor,
    parents: Tensor,
    lbs_weights: Tensor,
    pose2rot: bool = True,
) -> Tuple[Tensor, Tensor]:
```

- betas: $\vec{\beta}$
- pose: $\vec{\theta}$
- v_template: $\bar{\mathbf{T}}$
- shapedirs: $\mathcal{S}$
- posedirs: $\mathcal{P}$
- J_regressor: $\mathcal{J}$
- parents: kinematic tree for the model
- lbs_weight: $\mathcal{W}$
- pose2rot whether to convert the pose $\vec{\theta}$ tensor to rotation mat

Code: lbs.py>lbs()

```
# Add shape contribution
v_shaped = v_template + blend_shapes(betas, shapedirs)
```

$$\bar{\mathbf{T}} + B_S(\vec{\beta}; \mathcal{S})$$

```
# Get the joints
# NxJx3 array
J = vertices2joints(J_regressor, v_shaped)
```

$$\mathcal{J}(\bar{\mathbf{T}} + B_S(\vec{\beta}; \mathcal{S}))$$

```
def vertices2joints(J_regressor: Tensor, vertices: Tensor) -> Tensor:
```

Code: lbs.py>lbs()

```python
# 3. Add pose blend shapes
# N x J x 3 x 3
ident = torch.eye(3, dtype=dtype, device=device)
if pose2rot:
    rot_mats = batch_rodrigues(pose.view(-1, 3)).view(
        [batch_size, -1, 3, 3])

    pose_feature = (rot_mats[:, 1:, :, :] - ident).view([batch_size, -1])
    # (N x P) x (P, V * 3) -> N x V x 3
    pose_offsets = torch.matmul(
        pose_feature, posedirs).view(batch_size, -1, 3)
else:
    pose_feature = pose[:, 1:].view(batch_size, -1, 3, 3) - ident
    rot_mats = pose.view(batch_size, -1, 3, 3)

    pose_offsets = torch.matmul(pose_feature.view(batch_size, -1),
                                posedirs).view(batch_size, -1, 3)
```

$R_n(\vec{\theta}^*)$

$R_n(\vec{\theta})$ $\exp(\vec{\omega})$

$\vec{\theta} = [\vec{\omega}_0^T, \ldots, \vec{\omega}_K^T]^T$

$R_n(\vec{\theta}) - R_n(\vec{\theta}^*)$

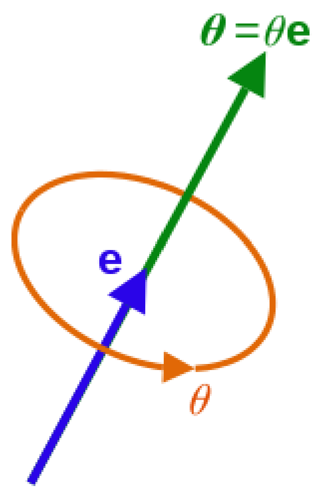
$$(R_n(\vec{\theta}) - R_n(\vec{\theta}^*))\mathbf{P}_n \Rightarrow \sum_{n=1}^{9K} (R_n(\vec{\theta}) - R_n(\vec{\theta}^*))\mathbf{P}_n$$

```python
def batch_rodrigues(
    rot_vecs: Tensor,
    epsilon: float = 1e-8,
) -> Tensor:
```

Code: lbs.py>batch_rodrigues()

```
angle = torch.norm(rot_vecs + 1e-8, dim=1, keepdim=True)
rot_dir = rot_vecs / angle

cos = torch.unsqueeze(torch.cos(angle), dim=1)
sin = torch.unsqueeze(torch.sin(angle), dim=1)

# Bx1 arrays
rx, ry, rz = torch.split(rot_dir, 1, dim=1)
K = torch.zeros((batch_size, 3, 3), dtype=dtype, device=device)

zeros = torch.zeros((batch_size, 1), dtype=dtype, device=device)
K = torch.cat([zeros, -rz, ry, rz, zeros, -rx, -ry, rx, zeros], dim=1) \
    .view((batch_size, 3, 3))

ident = torch.eye(3, dtype=dtype, device=device).unsqueeze(dim=0)
rot_mat = ident + sin * K + (1 - cos) * torch.bmm(K, K)
return rot_mat
```

Axis-angle representation



$$\mathbf{K} = \begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix}$$

$$\mathbf{R} = \mathbf{I} + (\sin\theta)\mathbf{K} + (1 - \cos\theta)\mathbf{K}^2$$

*논문 수식이 아닌 위키피디아 수식

Code: lbs.py>lbs()

```
v_posed = pose_offsets + v_shaped
```

$$\bar{\mathbf{T}} + B_S(\vec{\beta}; \mathcal{S}) + B_P(\vec{\theta}; \mathcal{P})$$

```
# 4. Get the global joint location
J_transformed, A = batch_rigid_transform(rot_mats, J, parents, dtype=dtype)
```

$$G_k(\vec{\theta}, \mathbf{J}) = \prod_{j \in A(k)} \left[\begin{array}{c|c} \exp(\vec{\omega}_j) & \mathbf{j}_j \\ \hline \vec{0} & 1 \end{array} \right]$$

$$G'_k(\vec{\theta}, \mathbf{J}) = G_k(\vec{\theta}, \mathbf{J}) G_k(\vec{\theta}^*, \mathbf{J})^{-1}$$

```
def batch_rigid_transform(
    rot_mats: Tensor,
    joints: Tensor,
    parents: Tensor,
    dtype=torch.float32
) -> Tensor:
```

Code: lbs.py>batch_rigid_transform()

```
joints = torch.unsqueeze(joints, dim=-1)

rel_joints = joints.clone()
rel_joints[:, 1:] -= joints[:, parents[1:]]

transforms_mat = transform_mat(
    rot_mats.reshape(-1, 3, 3),
    rel_joints.reshape(-1, 3, 1)).reshape(-1, joints.shape[1], 4, 4)
```

$$\left[\begin{array}{c|c} \exp(\vec{\omega}_j) & \mathbf{j}_j \\ \hline \vec{0} & 1 \end{array}\right]$$

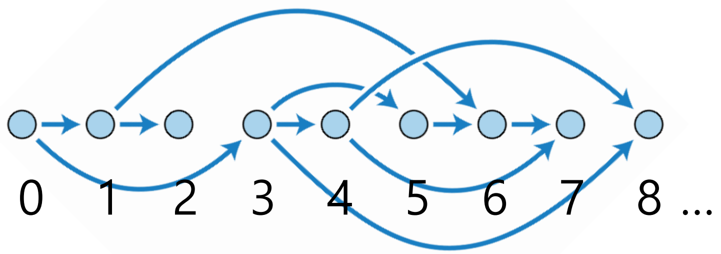
*rel_joints

```
def transform_mat(R: Tensor, t: Tensor) -> Tensor:
```

Code: lbs.py>batch_rigid_transform()

```
transform_chain = [transforms_mat[:, 0]]
for i in range(1, parents.shape[0]):
    # Subtract the joint location at the rest pose
    # No need for rotation, since it's identity when at rest
    curr_res = torch.matmul(transform_chain[parents[i]],
                            transforms_mat[:, i])
    transform_chain.append(curr_res)

transforms = torch.stack(transform_chain, dim=1)
```

$$G_k(\vec{\theta}, \mathbf{J}) = \prod_{j \in A(k)} \left[\begin{array}{c|c} \exp(\vec{\omega}_j) & \mathbf{j}_j \\ \hline \vec{0} & 1 \end{array} \right]$$



parent: directed acyclic tree(?)

Code: lbs.py>batch_rigid_transform()

```
# The last column of the transformations contains the posed joints
posed_joints = transforms[:, :, :3, 3]

joints_homogen = F.pad(joints, [0, 0, 0, 1])

rel_transforms = transforms - F.pad(
    torch.matmul(transforms, joints_homogen), [3, 0, 0, 0, 0, 0, 0, 0])

return posed_joints, rel_transforms
```

$$\left[\begin{array}{c|c} \exp(\vec{\omega}_j) & \boxed{\mathbf{j}_j} \\ \hline \vec{0} & 1 \end{array}\right]$$

$$G'_k(\vec{\theta}, \mathbf{J}) = G_k(\vec{\theta}, \mathbf{J}) G_k(\vec{\theta}^*, \mathbf{J})^{-1}$$

rel T = **T** – [**0** | **T**j] = **TI** – (**T**[**0** | j]) = **T**(**I** – [**0** | j])

T(**I** – [**0** | j])**v** = **T**(**v** – j) # note that pad value in joints_homogen matrix = 0

```
# Subtract the joint location at the rest pose
# No need for rotation, since it's identity when at rest
```

Code: lbs.py>lbs()

```python
# 4. Get the global joint location
J_transformed, A = batch_rigid_transform(rot_mats, J, parents, dtype=dtype)
```

```python
# 5. Do skinning:
# W is N x V x (J + 1)
W = lbs_weights.unsqueeze(dim=0).expand([batch_size, -1, -1])
# (N x V x (J + 1)) x (N x (J + 1) x 16)
num_joints = J_regressor.shape[0]
T = torch.matmul(W, A.view(batch_size, num_joints, 16)) \
    .view(batch_size, -1, 4, 4)
```

blend weights:
It represent how much the rotation matrix of each part affects each vertex

$$\sum_{k=1}^{K} w_{k,i} G'_k(\vec{\theta}, \mathbf{J})$$

Code: lbs.py>lbs()

```python
homogen_coord = torch.ones([batch_size, v_posed.shape[1], 1],
                           dtype=dtype, device=device)
v_posed_homo = torch.cat([v_posed, homogen_coord], dim=2)
v_homo = torch.matmul(T, torch.unsqueeze(v_posed_homo, dim=-1))

verts = v_homo[:, :, :3, 0]

return verts, J_transformed
```

$$\bar{\mathbf{t}}'_i = \sum_{k=1}^{K} w_{k,i} G'_k(\vec{\theta}, \mathbf{J}) \bar{\mathbf{t}}_i$$