<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.04.2016</td>
<td>Introduction</td>
</tr>
<tr>
<td>18.04.2016</td>
<td>Graphical Models 1</td>
</tr>
<tr>
<td>25.04.2016</td>
<td>Graphical Models 2 (Sand 6/7)</td>
</tr>
<tr>
<td>02.05.2016</td>
<td>Graphical Models 3</td>
</tr>
<tr>
<td>09.05.2016</td>
<td>Graphical Models 4</td>
</tr>
<tr>
<td>23.05.2016</td>
<td>Body Models 1</td>
</tr>
<tr>
<td>30.05.2016</td>
<td>Body Models 2</td>
</tr>
<tr>
<td>06.06.2016</td>
<td>Body Models 3</td>
</tr>
<tr>
<td><strong>13.06.2016</strong></td>
<td><strong>Body Models 4</strong></td>
</tr>
<tr>
<td>20.06.2016</td>
<td>Stereo</td>
</tr>
<tr>
<td>27.06.2016</td>
<td>Optical Flow</td>
</tr>
<tr>
<td>04.07.2016</td>
<td>Segmentation</td>
</tr>
<tr>
<td>11.07.2016</td>
<td>Object Detection 1</td>
</tr>
<tr>
<td>18.07.2016</td>
<td>Object Detection 2</td>
</tr>
</tbody>
</table>
What have we learned so far about bodies?

• BM1: Procrustes for rigid alignment
solve with procrustes
single step

\[ f = \arg \min_f \sum_i \left\| f(x_i) - y_i \right\|^2 \]
What have we learned so far about bodies?

- BM1: Procrustes for rigid alignment
- BM2: ICP, gradient-based ICP
solve with procrustes or gradient-based

\[
x_1^i = \arg \min_{x \in X} \| f^0(x) - y_i \|^2
\]

\[
f^1 = \arg \min_{f} \sum_i \| f(x_1^i) - y_i \|^2
\]
and iterate!

$$f^1(X) = \arg \min_{f} \sum_{i} \| f(x^1_i) - y_i \|^2$$

$$x^2_i = \arg \min_{x \in X} \| f^1(x) - y_i \|^2$$
and iterate!

\[
f^j(X) = \arg\min_{f} \sum_{i} \|f(x^j_i) - y_i\|^2
\]

\[
x^{j+1}_i = \arg\min_{x \in X} \|f^j(x) - y_i\|^2
\]
and iterate!

\[ f^j(X) = \arg \min_{f} \sum_{i} \|f(x^j_i) - y_i\|^2 \]

\[ x^j_i + 1 = \arg \min_{x \in X} \|f^j(x) - y_i\|^2 \]
and iterate!

\[
f^j(X) = \arg \min_{f} \sum_i \|f(x^j_i) - y_i\|^2
\]

\[
x_i^{j+1} = \arg \min_{x \in X} \|f^j(x) - y_i\|^2
\]
What have we learned so far about bodies?

- BM1: Procrustes for rigid alignment
- BM2: ICP, gradient-based ICP
- BM3: Articulated models, Blendshapes, SMPL
Parameterized Skinning

Standard skinning: \( W(\mathbf{T}, \mathbf{J}, \mathcal{W}, \vec{\theta}) \rightarrow \text{vertices} \)

SMPL model:

\[
M(\vec{\theta}, \vec{\beta}) = W(\mathbf{T}_F(\vec{\beta}, \theta), \mathbf{J}(\vec{\beta}), \mathcal{W}, \vec{\theta}) \rightarrow \text{vertices}
\]

SMPL is skinning parameterized by pose \( \vec{\theta} \) and shape \( \vec{\beta} \)
What is missing: today

- How do we fit SMPL to meshes without correspondences?
- This is a computer vision course. Where is the color in those meshes?
- Autodiff in images? OpenDR
- Fitting bodies to images
Fitting SMPL to a scan/mesh

- Problem: Given a registration, find the model pose and shape.

\[ \vec{\theta}, \vec{\beta} = \arg \min_{\vec{\theta}, \vec{\beta}} d(M(\vec{\theta}, \vec{\beta}) - \mathbf{V})^2 \]

Some distance function between the two meshes
Fitting SMPL to a scan/mesh

- Problem: Given a registration, find the model pose and shape.

```python
from smpl.serialization import load_model
sm = load_model(path_to_downloaded_model)
ch.minimize(point2point_squared(dst_pts=sm, org_pts=Xch),
x0=[sm.betas, sm.pose])
```
SMPL tree: sm.show_tree()
Chumpy minimizes the **sum of squares** of a **vector valued error** function

\[
e(x) = \sum_{i} e_i(x)^2 = e(x)^T e(x)
\]

**Optimization variables** (vector)

**Sum of squares** (scalar)

**Residuals** (vector valued error function)
Chumpy minimizes the **sum of squares** of a **vector valued error** function

\[ e(x) = \sum_{i} e_i(x)^2 = e(x)^T e(x) \]

```python
ipdb> p2p_yx = point2point_squared(org_pts=Xch, dst_pts=sm)
ipdb> print(p2p_yx)
[ 0.001  0.  0.001 ...,  0.012  0.012  0.012]
ipdb> p2p_yx.shape
(6890,)  as many elements as corresppences between model and scan
```
Jacobian of the vector valued error function:

\[ J_e(x) = \frac{d e(x)}{dx} = \begin{bmatrix}
\frac{\partial e_1}{\partial x_1} & \cdots & \frac{\partial e_1}{\partial x_P} \\
\vdots & \ddots & \vdots \\
\frac{\partial e_N}{\partial x_1} & \cdots & \frac{\partial e_N}{\partial x_P}
\end{bmatrix} \]

N residuals

P parameters
\[ J_e(x) = \frac{de(x)}{dx} = \begin{bmatrix} \frac{\partial e_1}{\partial x_1} & \cdots & \frac{\partial e_1}{\partial x_P} \\ \vdots & \ddots & \vdots \\ \frac{\partial e_N}{\partial x_1} & \cdots & \frac{\partial e_N}{\partial x_P} \end{bmatrix} \]

P parameters

N residuals
Try it!
Which one will fail?
Which one will fail?
Problems?

- Unlikely pose
Problems?

- Unlikely pose
- Unlikely shape
Problems?

- Unlikely pose
- Unlikely shape
- Bad initialization
Fitting SMPL to a scan/mesh

\[ \vec{\theta}, \vec{\beta} = \arg \min_{\vec{\theta}, \vec{\beta}} \| M(\vec{\theta}, \vec{\beta}) - \mathbf{V} \|^2 \]
Fitting SMPL to a scan/mesh

\[ \bar{\theta}, \bar{\beta} = \arg \min_{\hat{\theta}, \hat{\beta}} \| M(\hat{\theta}, \hat{\beta}) - \mathbf{V} \|^2 \]

\[ + E_\theta(\hat{\theta}) \]
Fitting SMPL to a scan/mesh

$$\vec{\theta}, \vec{\beta} = \arg\min_{\vec{\theta}, \vec{\beta}} \| M(\vec{\theta}, \vec{\beta}) - \mathbf{V} \|^2$$

$$+ E_\theta(\vec{\theta})$$

$$+ E_\beta(\vec{\beta})$$
Fitting SMPL to a scan/mesh

\[ \tilde{\theta}, \tilde{\beta} = \arg \min_{\theta, \beta} \| M(\tilde{\theta}, \tilde{\beta}) - V \|^2 \]

\[ + E_\theta(\tilde{\theta}) \]

\[ + E_\beta(\tilde{\beta}) \]

Mahalanobis distance induced by distribution \( \mathcal{N}(\mu_\theta, \Sigma_\theta) \)

\[ E_\theta(\tilde{\theta}) \equiv (\tilde{\theta} - \mu_\theta)^T \Sigma_\theta^{-1} (\tilde{\theta} - \mu_\theta) \]

\[ E_\beta(\tilde{\beta}) \equiv (\tilde{\beta} - \mu_\beta)^T \Sigma_\beta^{-1} (\tilde{\beta} - \mu_\beta) \]
Fitting SMPL to a scan/mesh

- What makes it so jumpy?
  - Correspondences change abruptly!
Point-to-point distance

\[ v_0 \in V \]

\[ v_2 \in V \]

\[ x \in M \]

\[ v_1 \in V \]
Point-to-point distance

\[ v_0 \in V \]

\[ v_1 \in V \]

\[ v_2 \in V \]

\[ x \in M \]
Point-to-point distance

\[ \| \mathbf{v}_0 - \mathbf{v}_1 \| \]

\[ \mathbf{v}_2 \in V \]

\[ \mathbf{x} \in M \]

\[ \mathbf{v}_0 \in V \]

\[ \mathbf{v}_1 \in V \]
Point-to-surface distance

\[ v_0 \in V \quad v_1 \in V \]

\[ x \in M \quad v \in V \]

\[ v_2 \in V \]
Point-to-surface distance

$v_0 \in V$
$v_1 \in V$
$v_2 \in V$
$x \in M$
$v \in V$
Point-to-surface distance

Implementation requires taking care of special cases when \( v \) falls in edges or points
Advanced registration

• Better pose priors

• Non-parametric

A Non-parametric Bayesian Network Prior of Human Pose, Lehrman et al
Fitting SMPL to a scan/mesh

• Better pose priors
  • Non-parametric
  • Dynamic

Efficient Nonlinear Markov Models for Human Motion, Lehrman et al
Fitting SMPL to a scan/mesh

- Better pose priors
- Non-parametric
- Dynamic
- Better initialisation
  - From previous frame, from discriminative approaches, from graphical models

The Stitched Puppet: A Graphical Model of 3D Human Shape and Pose, Zuffi and Black
Fitting SMPL to a scan/mesh

- Better pose priors
  - Non-parametric
  - Dynamic
- Better initialisation
  - From previous frame, from discriminative approaches, from graphical models
- Other information: appearance (color)!
Why appearance

More realism

More accurate correspondences
Representing appearance

Vertex coloring

\[ \mathbf{V} \in \mathbb{R}^{N \times 3} \]
\[ \mathbf{F} \in \mathbb{N}^{M \times 3}, \mathbf{F}_{ij} \in [0, N) \]
\[ \mathbf{W} \in \mathbb{R}^{N \times 3}, \mathbf{W}_{ij} \in [0, 256) \]
\[ \mathbf{f}_0 = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2] \]
\[ \mathbf{x} = \alpha_0 \mathbf{v}_0 + \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 \]
\[ c(\mathbf{x}) = \alpha_0 \mathbf{w}_0 + \alpha_1 \mathbf{w}_1 + \alpha_2 \mathbf{w}_2 \]
Decouple geometry and appearance resolution

\[ \mathbf{v}' \in \mathbb{R}^2, \mathbf{v}'_i \in [0, 1] \]

\[ \mathbf{f}' \in \mathbb{N}^3, \mathbf{f}'_i \in [0, N') \]

\[ \mathbf{v} \in \mathbb{R}^3 \]

\[ \mathbf{f} \in \mathbb{N}^3, \mathbf{f}_i \in [0, N) \]

\[ \mathbf{f}'_i \equiv \mathbf{f}_i \]

Representing appearance

Texture mapping

\[ V \in \mathbb{R}^{N \times 3} \]
\[ F \in \mathbb{N}^{M \times 3}, F_{ij} \in [0, N) \]
\[ V' \in \mathbb{R}^{N' \times 2}, V'_{ij} \in [0, 1] \]
\[ F' \in \mathbb{N}^{M \times 2}, F'_{ij} \in [0, N') \]
\[ U \in \mathbb{N}^{K \times K \times 3}, U_{ijk} \in [0, 256) \]
Texture mapping

\[ f_0 = [v_0, v_1, v_2] \]
\[ x \equiv \alpha_0 v_0 + \alpha_1 v_1 + \alpha_2 v_2 \]
\[ f'_0 = [v'_0, v'_1, v'_2] \]
\[ c(x) = U[\alpha_0 v'_0 + \alpha_1 v'_1 + \alpha_2 v'_2] \]

How do we create texture maps?
From 2D images to textures

Problem: combining multiple views of a 3D surface
From 2D images to textures

Problem: combining multiple views of a 3D surface
From 2D images to textures
From 2D images to textures

original image

visibility of original pixels in U

original pixels mapped to U
From 2D images to textures
From 2D images to textures
From 2D images to textures
Generating an image
Generating an image

shape \( \beta \)

pose \( \theta \)

image
Generating an image

shape \( \beta \)

pose \( \theta \)

camera \( K \)

image
Generating an image

- shape
- pose
- UV map
- camera
- image

\[ \beta, \theta, U, K \]
That’s all, no?

shape $\beta$

pose $\theta$

UV map $U$

camera $K$

image
This slide is wrong: have all the vertices the same albedo?

- shape
- pose
- camera
- image

\[ \beta \quad \theta \quad K \]
This one has a single albedo

\[ \beta \] shape

\[ \theta \] pose

image

camera

K
Generating an image

shape $\beta$

pose $\theta$

lighting $l$

camera $K$

image
Albedo and shading

Albedo is constant: depends on physical properties of the surface
Shading is transient: given by the interplay between surface reflectance and lighting
Reflectance models

Lambertian reflectance

\[ i_x = (n_x \cdot l_x) a_x l \]

- surface color
- surface normal
- direction from \( x \) to light source
- albedo
- light intensity
Lighting models

Point light sources
Lighting models

Spherical Harmonics (SH)

Lighting as a function over the sphere, projected onto a low-order SH basis

Simple and efficient for diffuse environments

Sloan et al., SIGGRAPH 2002.
Lighting models

Spherical Harmonics (SH)

Lighting as a function over the sphere, projected onto a low-order SH basis

Simple and efficient for diffuse environments

Sloan et al., SIGGRAPH 2002.
Modeling all together

- shape
- pose
- camera

UV map
lighting
K

images
Forward rendering process

Rendering takes model parameters and produces images.

\[ f(\beta, \theta, U, l, K) \]
Gradient-based optimization?

• We want to exploit images to obtain better registrations

• We saw that we can optimise a function given its derivatives

• Most of the functions involved in the rendering are linear operators

• Anybody wants to write the jacobians by hand?
An open source differentiable rendering framework for:

- approximating a rendering process
- differentiating this approximation
- finding parameter estimates

http://open-dr.org

Loper and Black, ECCV 2014.
import chumpy as ch
from opendr.everything import *

# Load mesh
m = load_mesh('/Users/matt/geist/OpenDR/test_dr/nasa_earth.obj')
w, h = (320, 240)
trans = ch.array([[0, 0, 0]])

# Construct renderer
rn = TexturedRenderer()
Appearance-based registration
Building an appearance model

graphology-based registration

initial registrations

appearance model (texture map)

Blending
Appearance-based error term

- Light modeling
- Real albedo images
- Rendered images
- Texture map $U$
- Registration $V_j$
- Per-pixel squared difference
New registration objective

\[ \vec{\theta}, \vec{\beta} = \arg \min_{\vec{\theta}, \vec{\beta}} \| M(\vec{\theta}, \vec{\beta}) - V \|^2 \]

\[ + \ E_\theta(\vec{\theta}) \]

\[ + \ E_\beta(\vec{\beta}) \]

\[ + \ E_U(I, K, U, M(\vec{\theta}, \vec{\beta})) \]

\[ E_U \equiv \sum_i \| I_i - r(M(\vec{\theta}, \vec{\beta}), U, K_i) \|^2 \]
With OpenDR...

import chumpy as ch
import cv2
from opendr.camera import ProjectPoints
from opendr.renderers import TexturedRenderer

# Load meshes, create other objectives...
# …

# Construct renderer
rn = TexturedRenderer()
rn.camera = ProjectPoints(v=m.v, vc=m.vc, rt=ch.zeros(3), t=ch.zeros(3),
                          f=ch.array([w,w])/2., c = ch.array([w,h])/2., k=ch.zeros(5))
rn.frustum = {'near': 1., 'far': 10., 'width': w, 'height': h}
rn.set(f=m.f, texture_image=m.texture_img, ft=m.ft, vt=m.vt, bgcolor=ch.zeros(3))

# Define the error term
obj = rn - cv2.imread(real_img_path)

# Minimize
ch.minimize(obj, x0=[m.v], method='dogleg')

lighting encoded in vc
appearance encoded in
Texture-based registration

- The appearance objective function has MANY local minima

- Pyramids of blurred images help

- The dimensionality of this objective is much bigger than the geometric one

- Optimisation will be slower

- Open problems: Lighting optimisation? Occlusions?

---

Scan model

\[ \text{gradient} = 0 \]
Texture-based registration

- The appearance objective function has MANY local minima
- Pyramids of blurred images help

![Scan model with gradient](image1)

![Scan model with gradient](image2)
Texture-based registration

- The appearance objective function has MANY local minima
  - Pyramids of blurred images help
- The dimensionality of this objective is much bigger than the geometric one
  - Optimisation will be slower
Texture-based registration

- The appearance objective function has MANY local minima
  - Pyramids of blurred images help
- The dimensionality of this objective is much bigger than the geometric one
  - Optimisation will be slower
- Open problems: Lighting optimisation? Occlusions?
Take-home message

• Optimising SMPL pose and shape with chumpy is easy

• But the devil is in the details: point2surface, regularisers

• We can add color to our model either with per-vertex colors, or texture maps

• Apart from making the model match the scan geometrically, we can make it match in terms of COLOR

• OpenDR differentiates the rendering process for us