Face Processing

IN4393 – Computer Vision

Introduction

- Face processing/analysis comprises a number of different tasks:
  - Face detection (*“where is a face?”*)
  - Face recognition (*“of whom is this face?”*)
  - Face verification (*“are these faces the same?”*)
  - Expression recognition (*“is this face happy or not?”*)

Face detection: Viola & Jones

- Train a *classifier* to predict whether a bounding box contains a face or not:

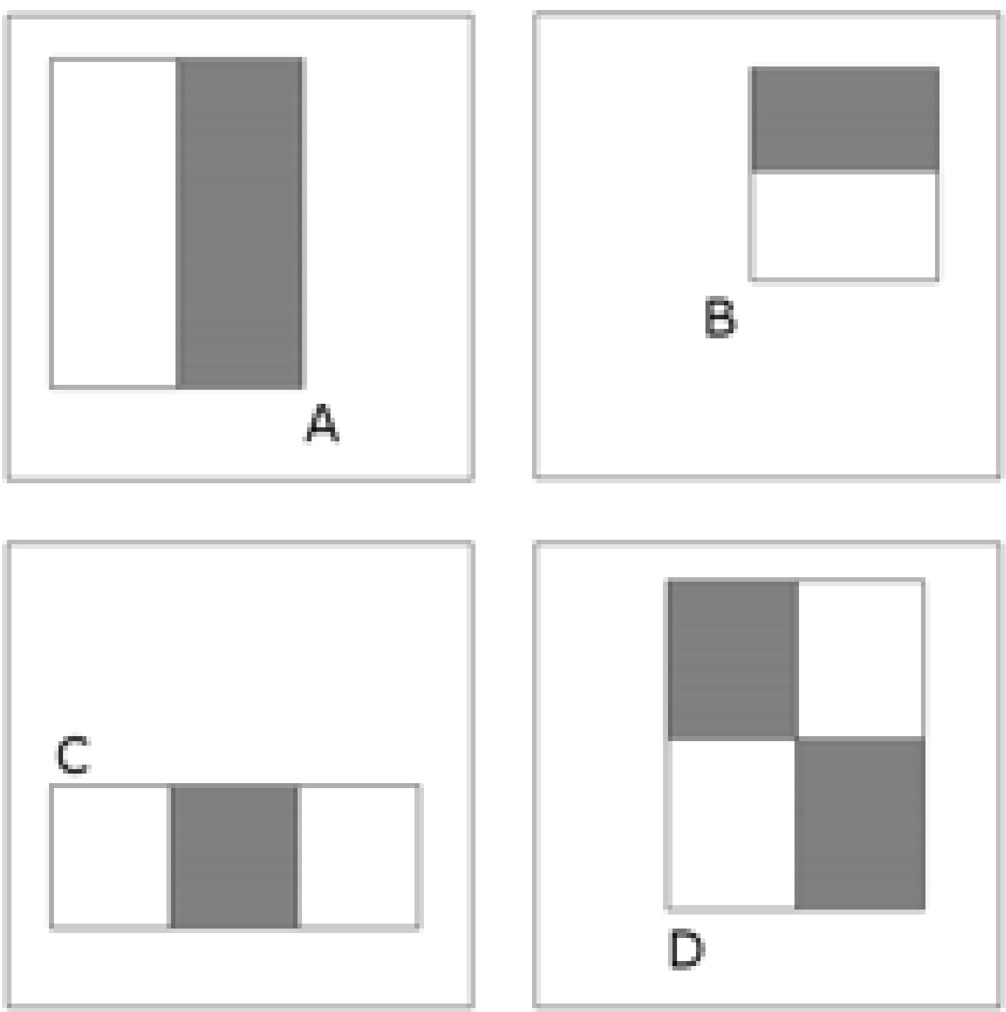
Face detection: Viola & Jones

- At test time, use a sliding window detector for multiple scales:

- Use *non-maxima suppression* in x-y-scale space to filter classifier predictions

Face detection: Viola & Jones

- Extract *Haar features* from the image patch, using the *integral image*:



Feature Value =

∑ (pixels in white area) – ∑ (pixels in black area)

- We find features that are common in faces, and use these as *weak learners*

Face detection: Viola & Jones

- Features can be computed efficiently using the *integral image*:

12	8	2	4	7
2	11	3	6	8
3	2	0	1	10
1	5	2	7	2
0	0	2	3	2

12	20	22	26	33
14	33	38	48	63
17	38	43	54	79
18	44	51	69	96
18	44	53	74	103

Face detection: Viola & Jones

- Features can be computed efficiently using the *integral image*:

12	8	2	4	7
2	11	3	6	8
3	2	0	1	10
1	5	2	7	2
0	0	2	3	2

12	20	22	26	33
14	33	38	48	63
17	38	43	54	79
18	44	51	69	96
18	44	53	74	103

Face detection: Viola & Jones

- Features can be computed efficiently using the *integral image*:

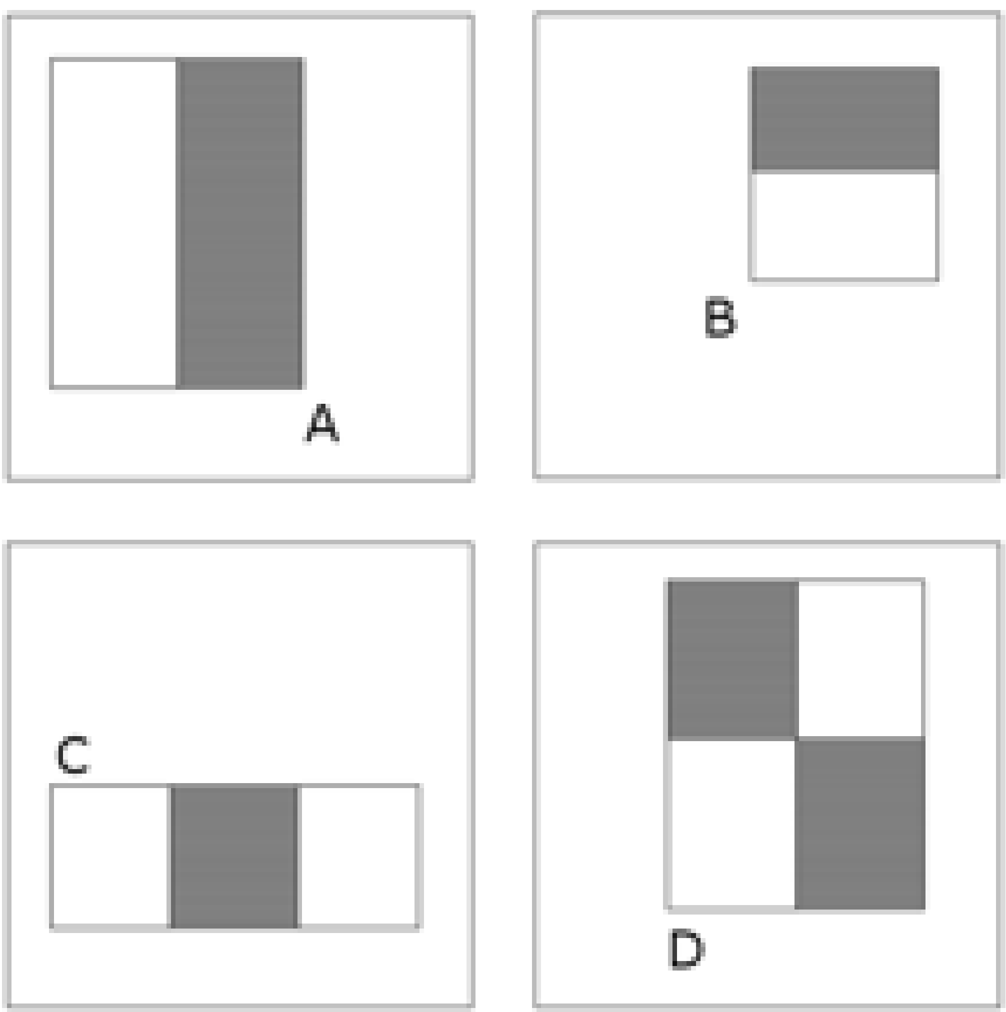
12	8	2	4	7
2	11	3	6	8
3	2	0	1	10
1	5	2	7	2
0	0	2	3	2

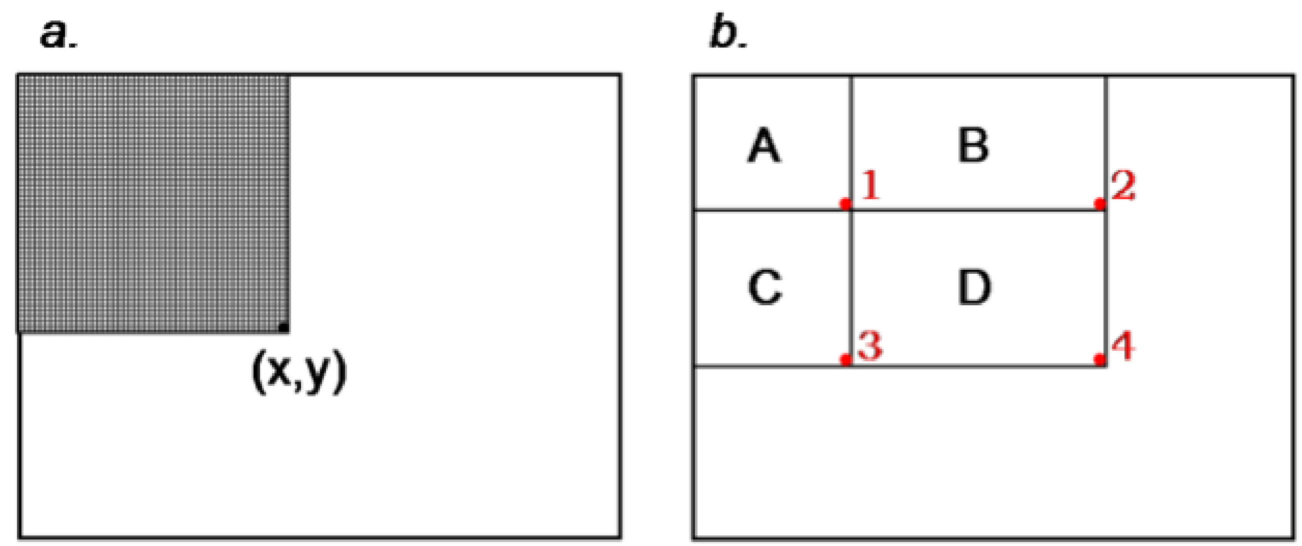
12	20	22	26	33
14	33	38	48	63
17	38	43	54	79
18	44	51	69	96
18	44	53	74	103

A-B-C+D

Face detection: Viola & Jones

- Extract *Haar features* from the image patch, using the *integral image*:





- We find features that are common in faces, and use these as *weak learners*

Face detection: Viola & Jones

- *AdaBoost* training on annotated data set; learns collection of *weak learners*:

$$h(\mathbf{x}) = \operatorname{sign}\left[\sum_{i=1}^{m} \alpha_i h_i(\mathbf{x})\right]$$

Face detection: Viola & Jones

- *AdaBoost* training on annotated data set; learns collection of *weak learners*:

$$h(\mathbf{x}) = \text{sign}\left[\sum_{i=1}^{m} \alpha_i h_i(\mathbf{x})\right]$$

- In Viola & Jones, the weak learners are *decision stumps*:

$$h_i(\mathbf{x}) = [f_i \geq \theta_i]$$

Face detection: Viola & Jones

- *AdaBoost* training on annotated data set; learns collection of *weak learners*:

$$h(\mathbf{x}) = \operatorname{sign}\left[\sum_{i=1}^{m} \alpha_i h_i(\mathbf{x})\right]$$

- In Viola & Jones, the weak learners are *decision stumps*:

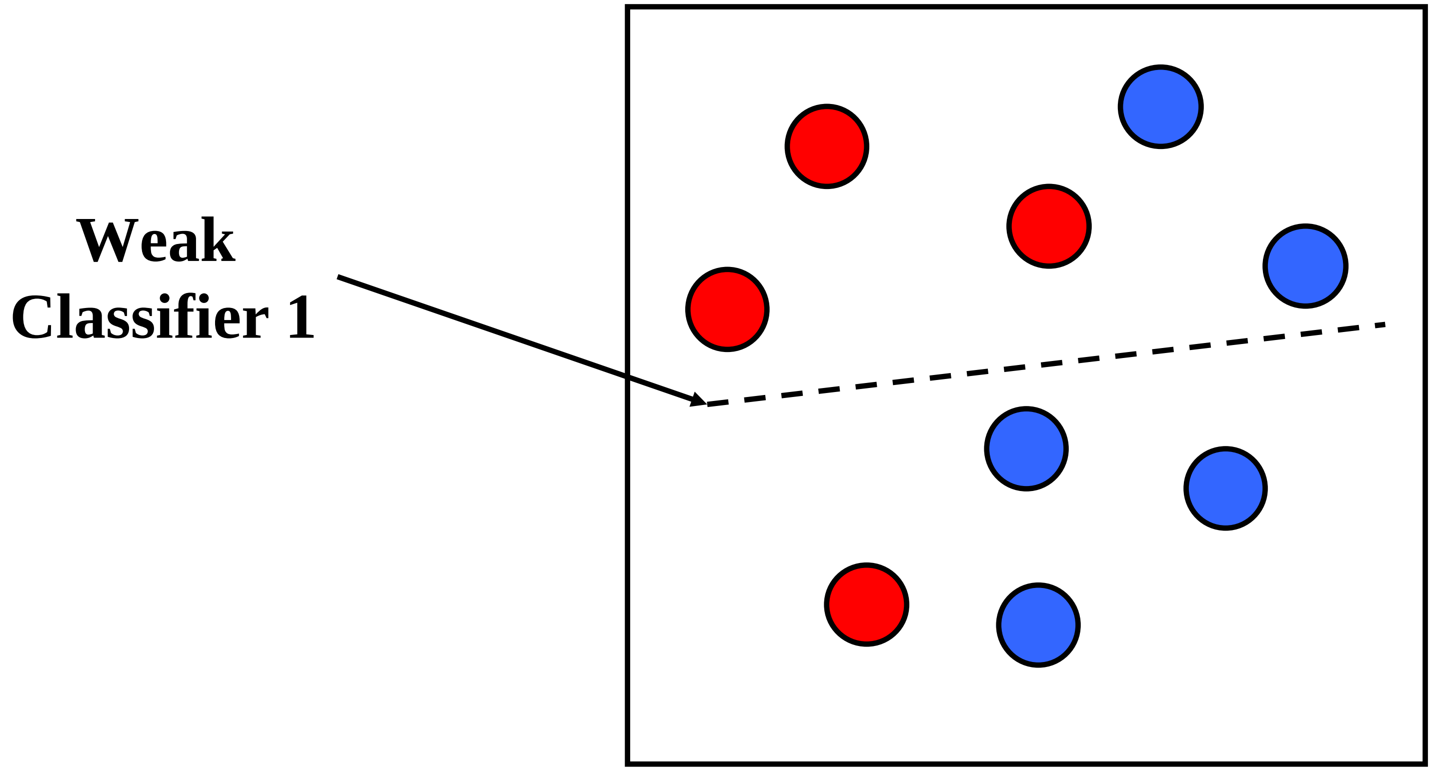
$$h_i(\mathbf{x}) = [f_i \geq \theta_i]$$

- Iteratively select the learner that minimizes the *weighted* classification error:

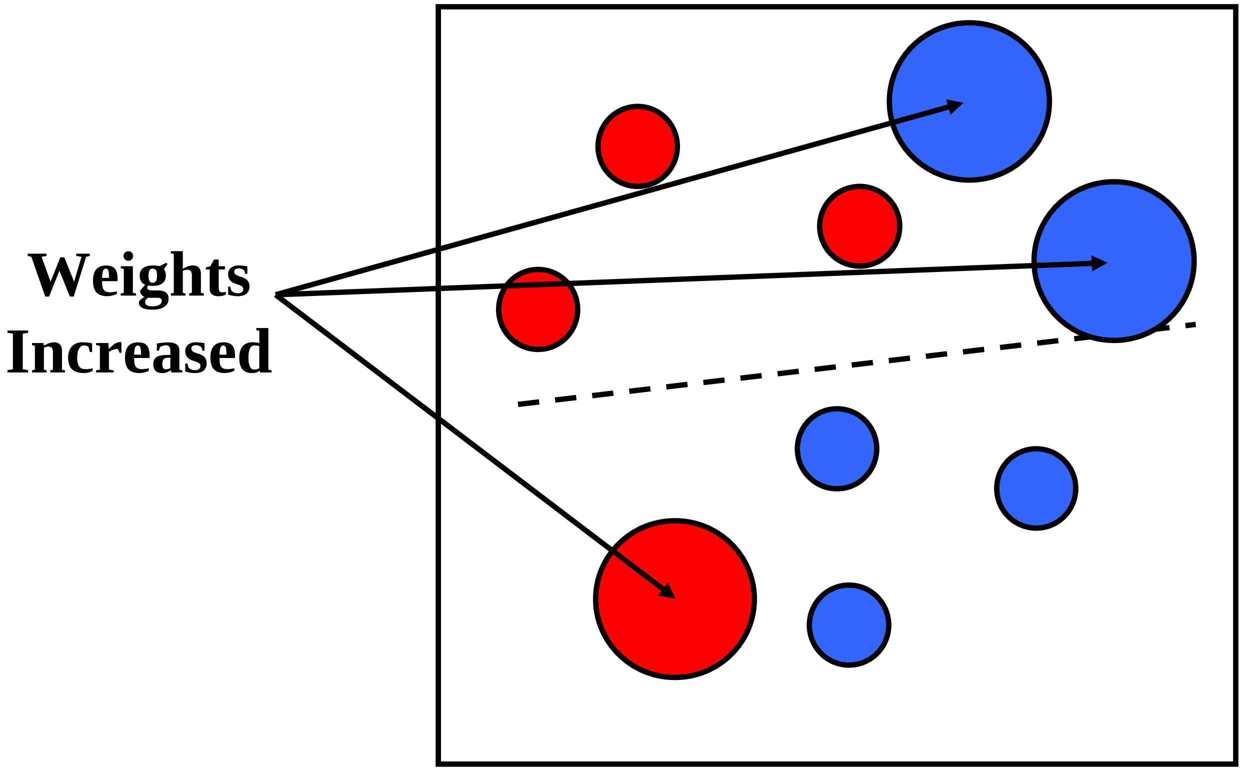
$$e_i = \sum_{n=1}^{N} w_{n,i}(1 - \delta(y_n, h_i(\mathbf{x}_n; \theta_i)))$$

  - Efficient algorithms exist to find the threshold in linear time
  - Update the *per-instance weights* based on the classification error of weak learner

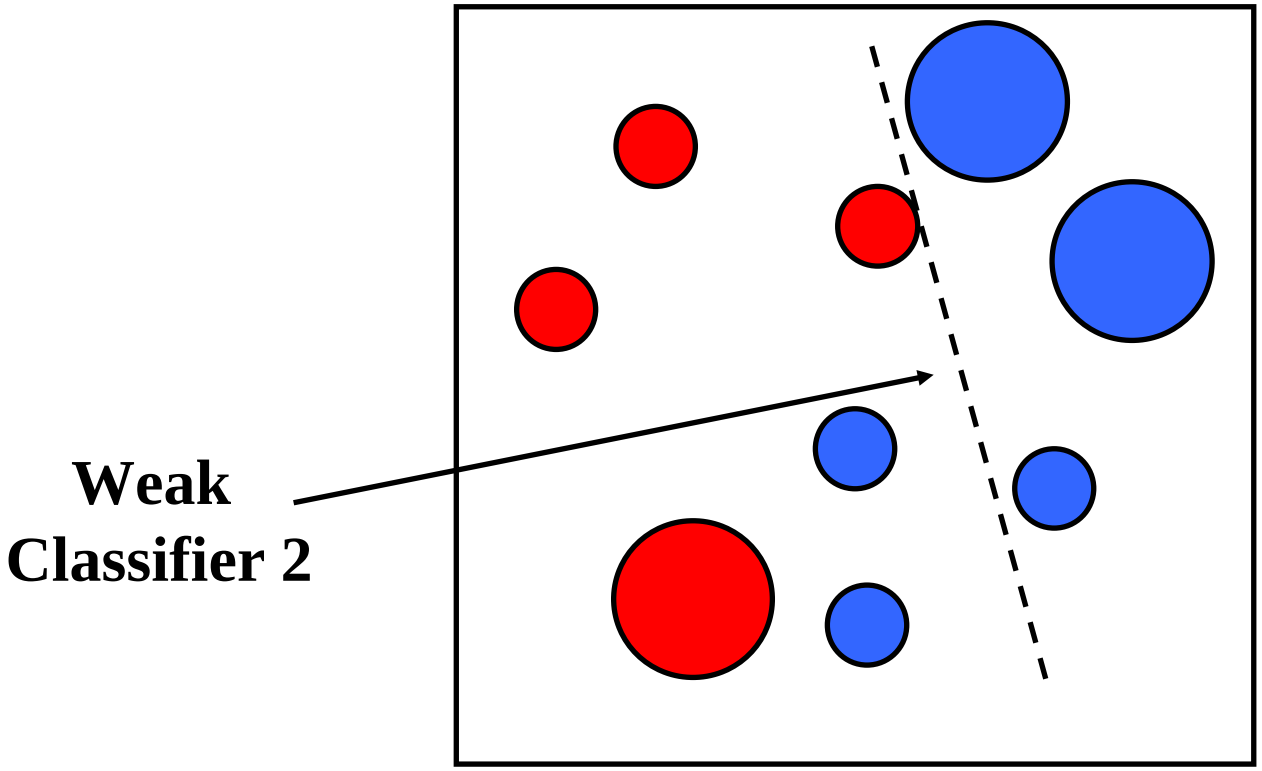
AdaBoost learning illustration



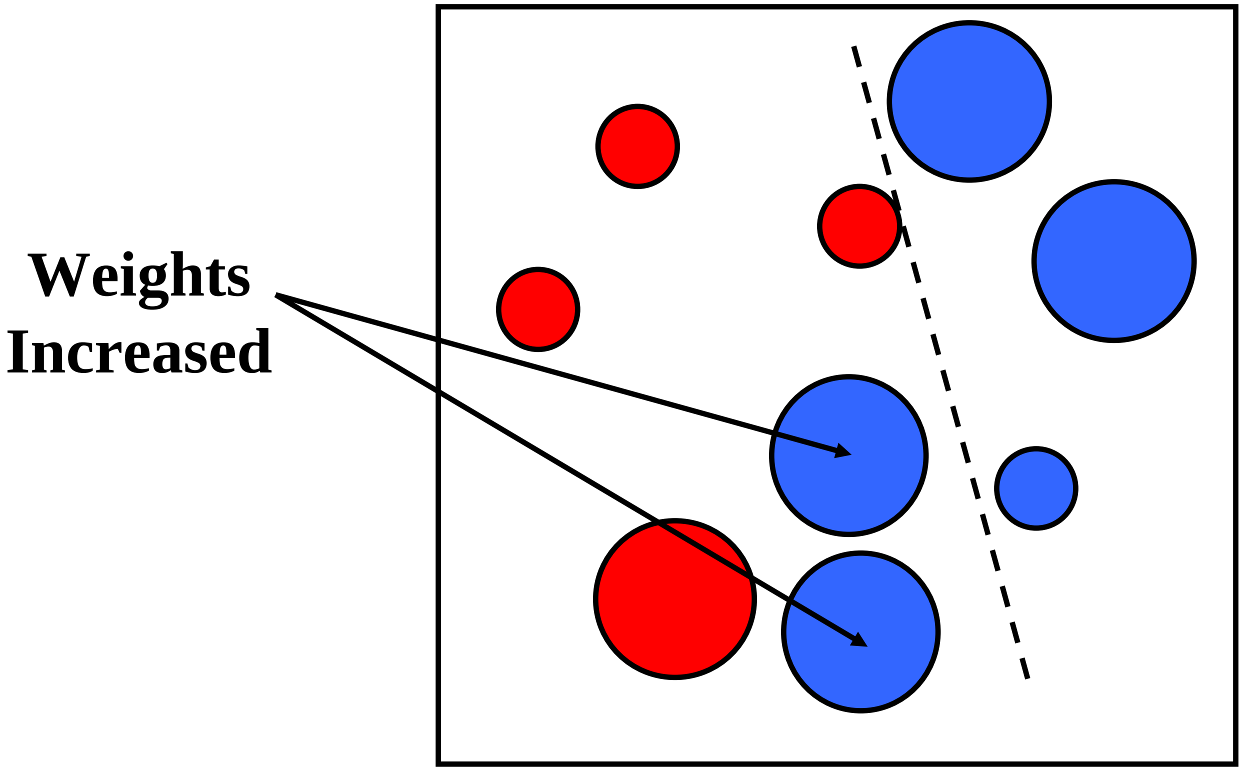
AdaBoost learning illustration



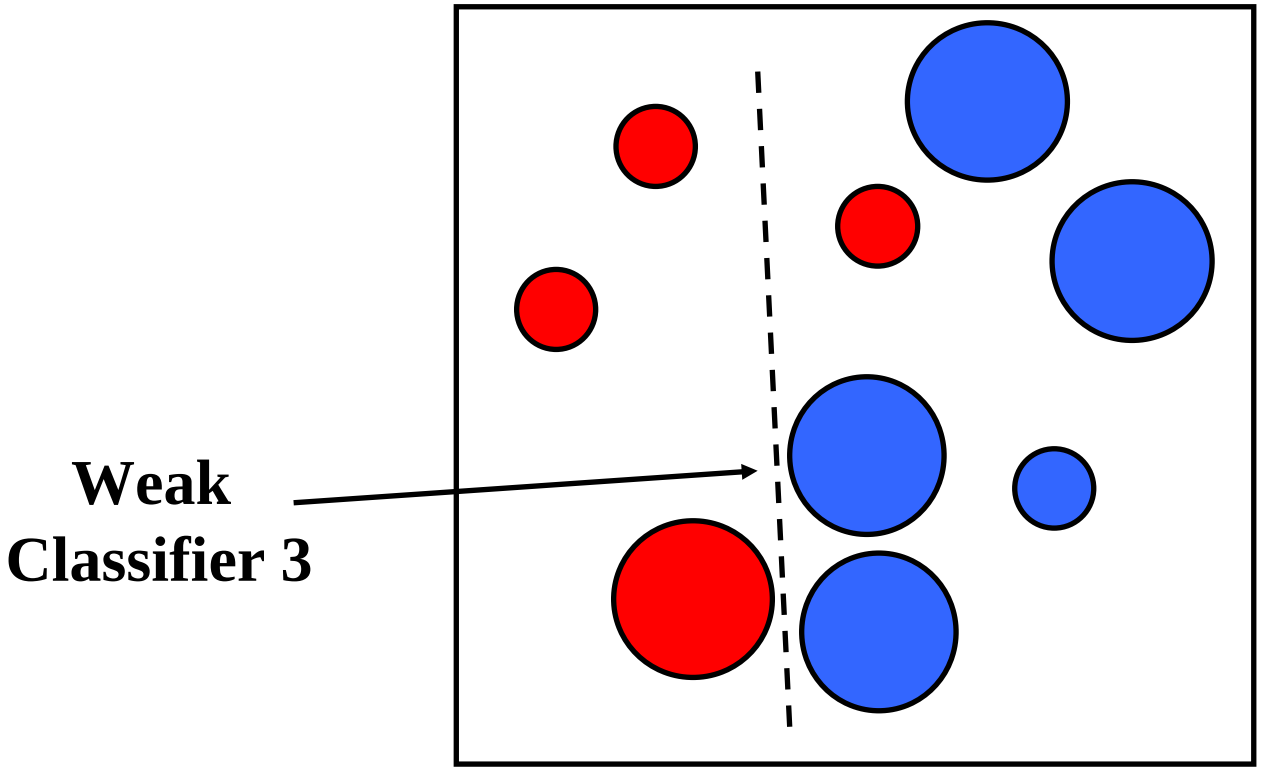
AdaBoost learning illustration



AdaBoost learning illustration

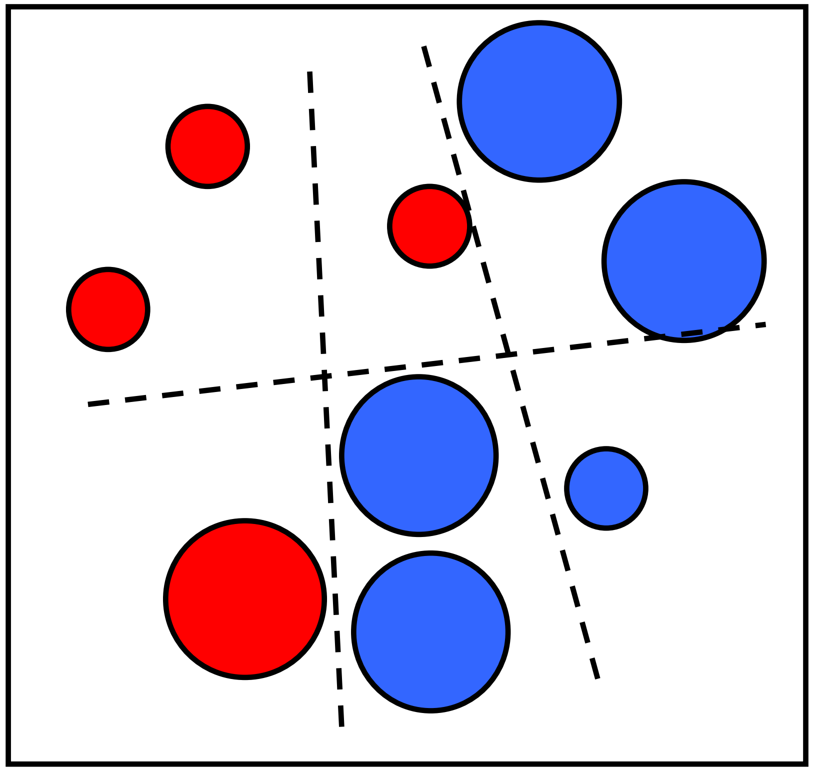


AdaBoost learning illustration



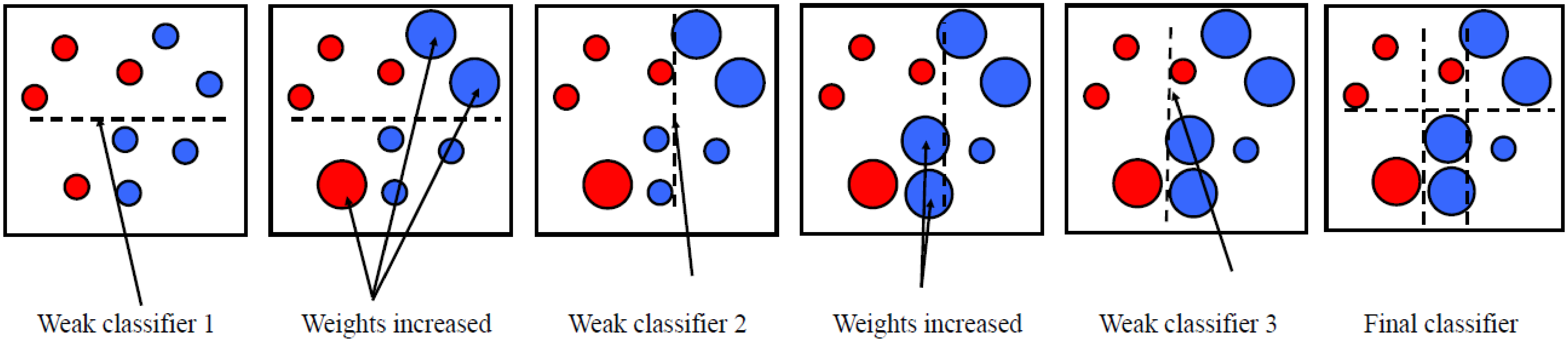
AdaBoost learning illustration

Final classifier is a combination of weak classifiers



Face detection

- Schematic overview of AdaBoost learning with decision stumps:

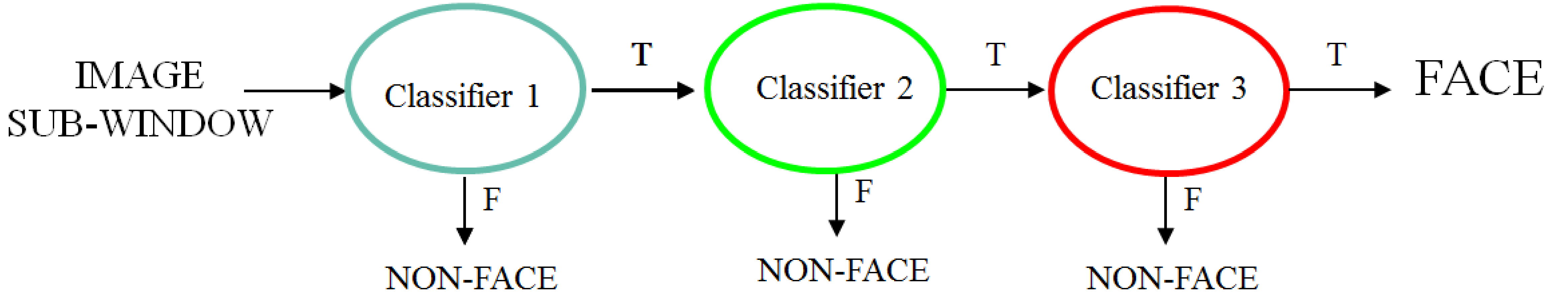


- Update for per-instance weights: $$w_{n,i+1} \leftarrow w_{n,i} \left(\frac{e_i}{1-e_i} \right)^{1-\delta(y_n, h_i(\mathbf{x}_n;\theta_i))}$$

- Weak-learner weights given by: $$\alpha_i = -\log \left(\frac{e_i}{1-e_i} \right)$$

Face detection: Viola & Jones

- To perform the detection, we use a sliding window detector (at multiple scales)
- The classification of a patch can be performed using a *cascaded classifier*:



Note that this is extremely fast at test time: for negative examples, we typically only need to compute a very small number of features!

Face detection: Viola & Jones

- False positive rate of a cascade with K classifiers: $$FPR = \prod_{i=1}^{K} FPR_i$$

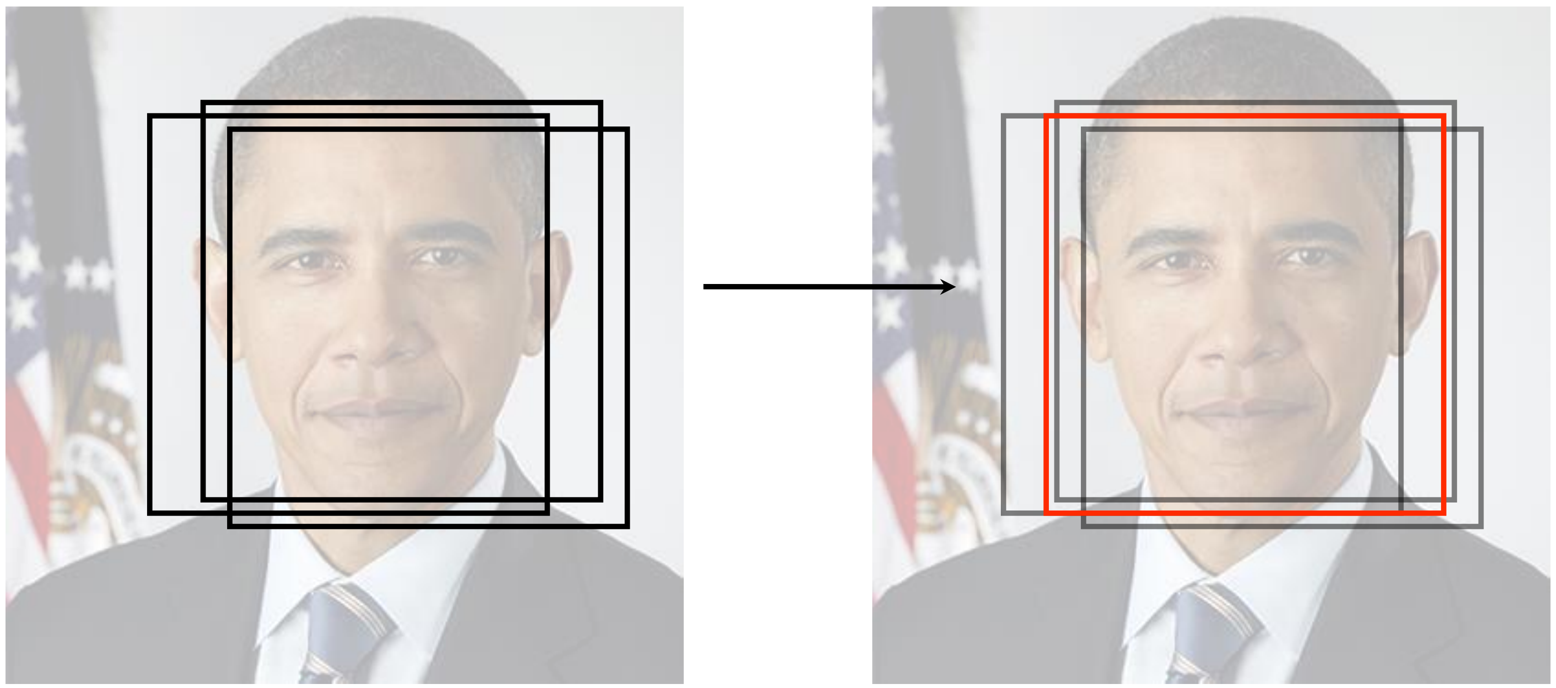
- Detection rate of a cascade with K classifiers: $$DR = \prod_{i=1}^{K} DR_i$$

Face detection: Viola & Jones

- False positive rate of a cascade with K classifiers: $$FPR = \prod_{i=1}^{K} FPR_i$$
- Detection rate of a cascade with K classifiers: $$DR = \prod_{i=1}^{K} DR_i$$
- Assume we have a cascade of K = 32 classifiers:
  - To get a false positive rate of 10^{-6}, each classifier may have FPR of 65%
  - To get a detection rate of 90%, each classifier should have DR of 99.7%

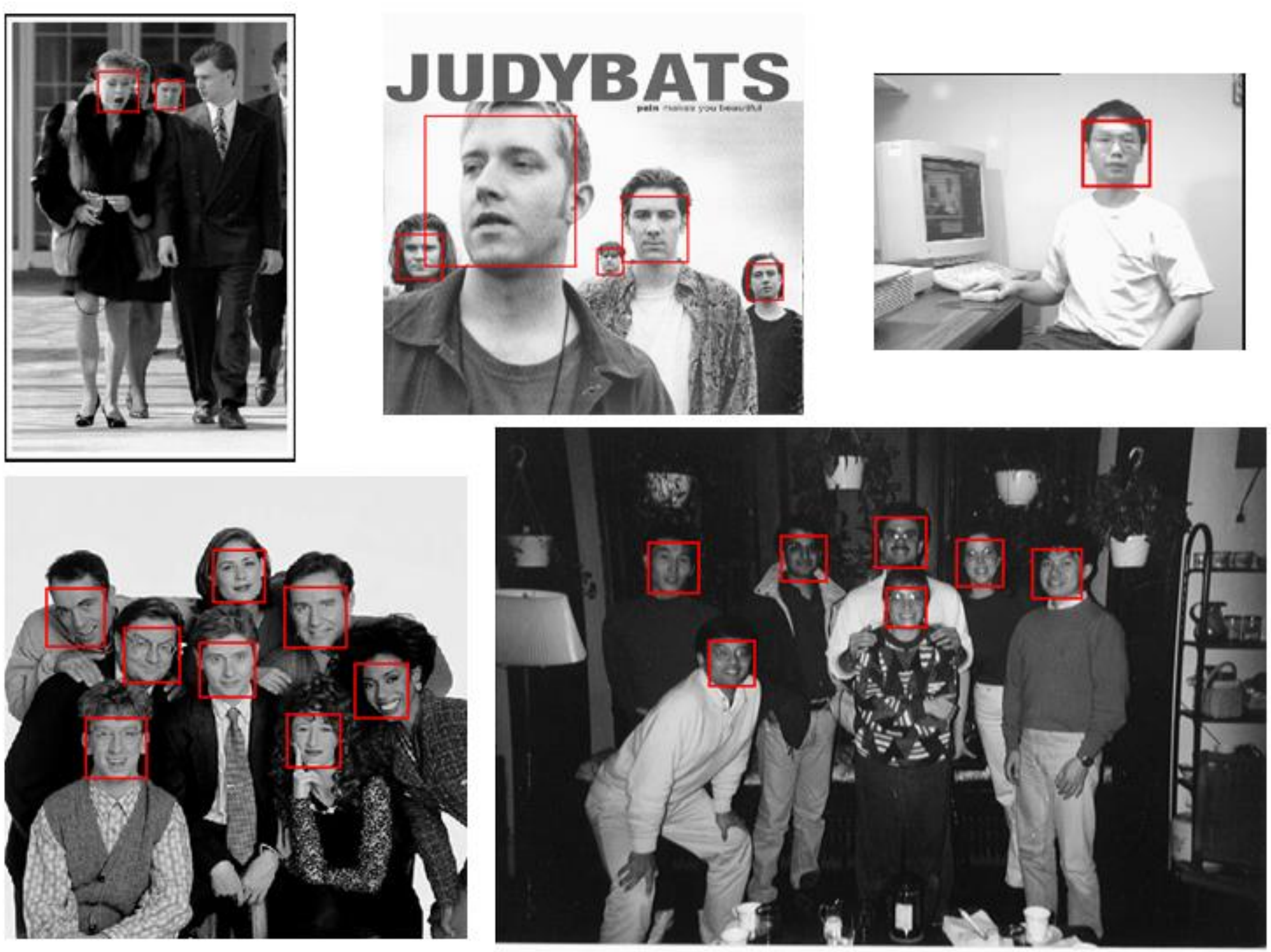
Face detection: Viola & Jones

- Multiple locations near a face will typically yields multiple detections
- In the original V&J detector, the detections are post-processed as follows:
  - Whenever two detections overlap, the bounding boxes are merged
  - The final detection is the average of the corners of all merged detections:



Face detection: Viola & Jones

- Examples of face detections (using V&J implementation in OpenCV):



Face detection: Viola & Jones

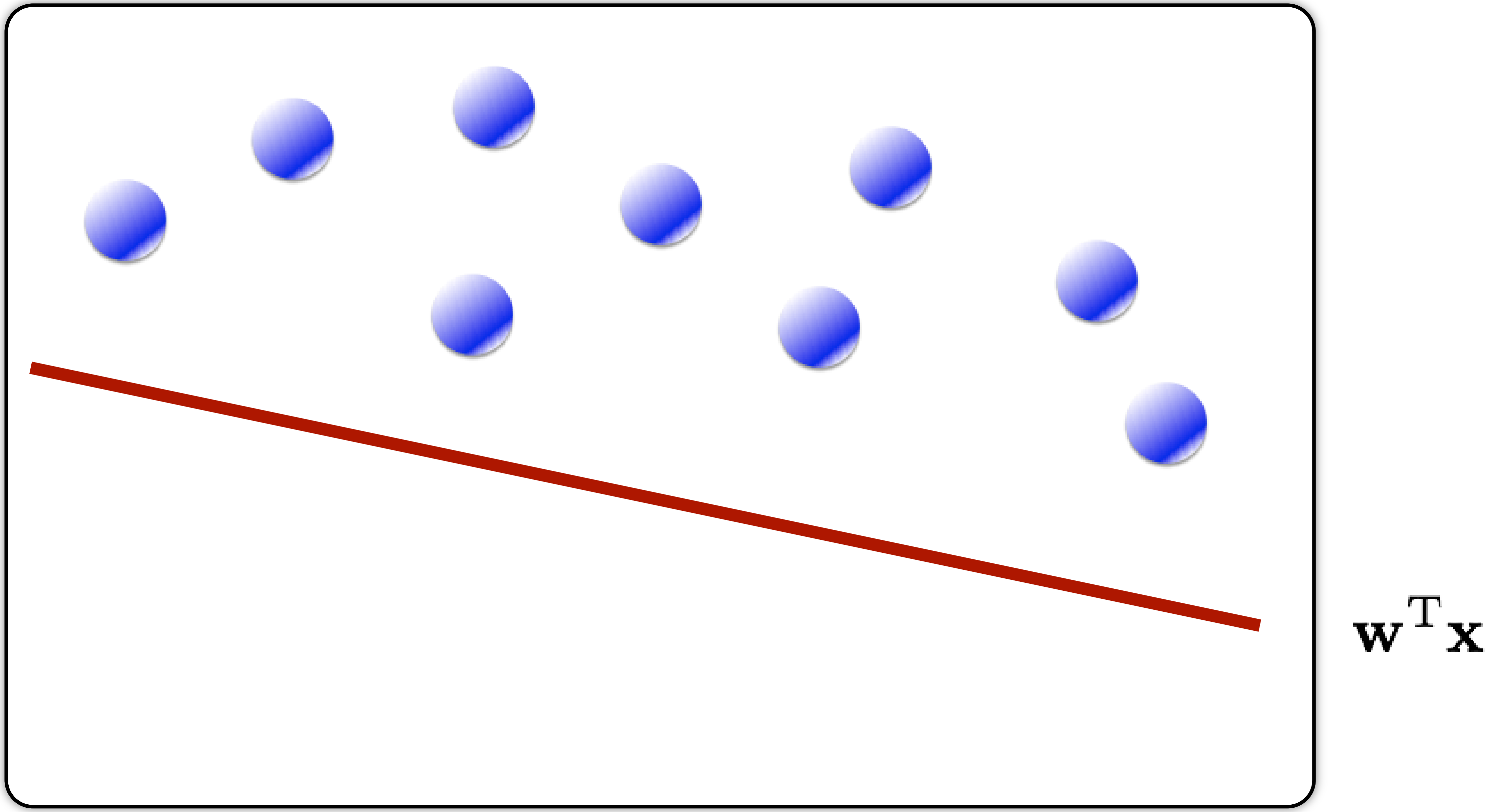
- Detection of profile faces requires training on separate data set:

Face recognition: Eigenfaces

- Consider face images as high-dimensional data points
- Apply *dimension reduction* on the images to obtain low-dimensional features
- The reduction is performed using *principal components analysis*

Face recognition: Eigenfaces

- Principal Components Analysis maps the data in a *linear subspace*, such that the *variance* of the projected data is maximized:



Face recognition: Eigenfaces

- Our objective is to maximize variance: $\max_{\|\mathbf{w}\|^2=1} var(\mathbf{w}^T\mathbf{X})$
- Assuming zero-mean data: $var(\mathbf{w}^T\mathbf{X}) = [\mathbf{w}^T\mathbf{X}\mathbf{X}^T\mathbf{w}] = [\mathbf{w}^T\mathbf{C}\mathbf{w}]$
- Enforce constraint using *Lagrange multipliers*:

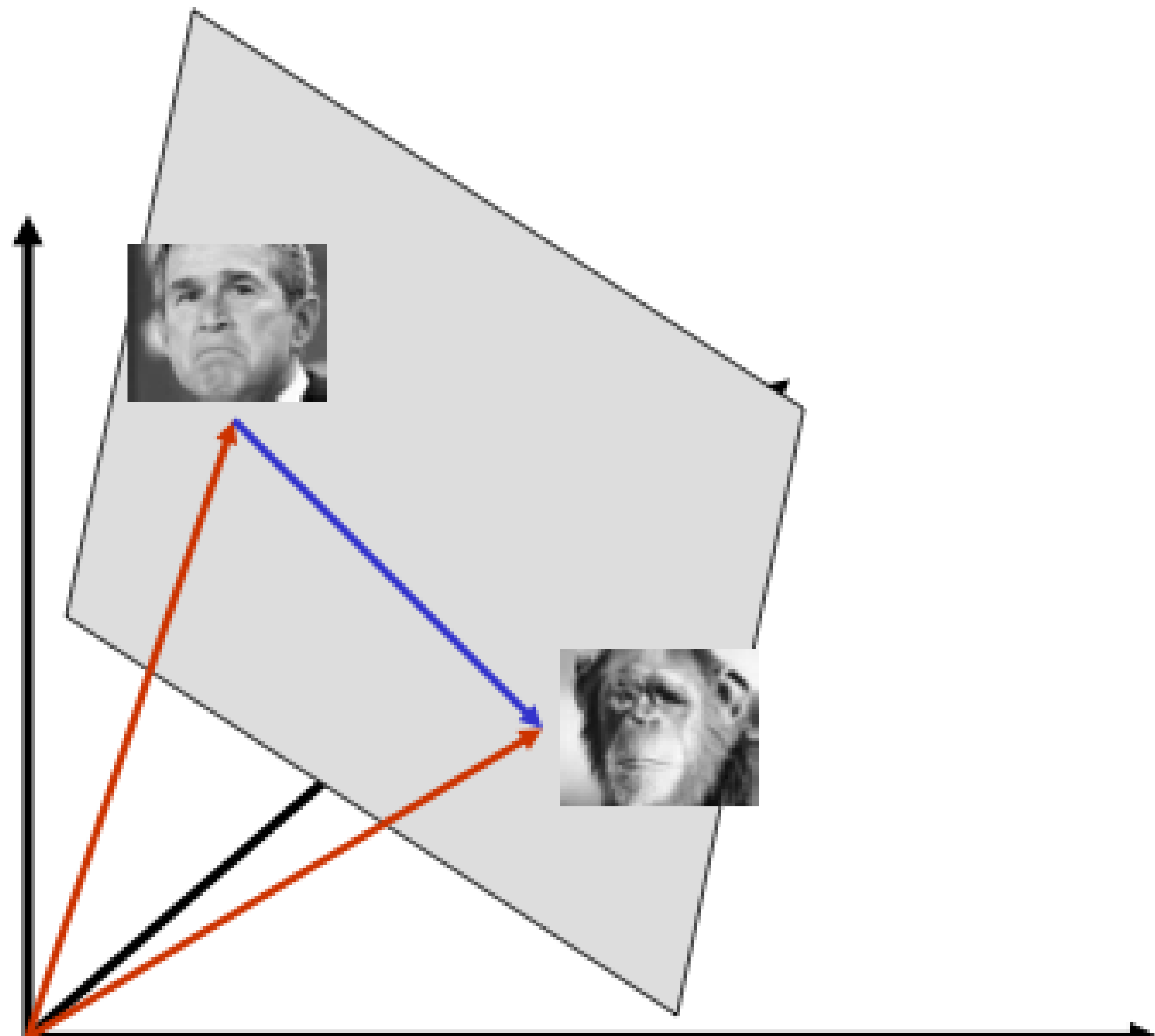
$$\max_{\|\mathbf{w}\|^2=1} var(\mathbf{w}^T\mathbf{X}) = \max_{\mathbf{w},\lambda} \mathbf{w}^T\mathbf{C}\mathbf{w} - \lambda(\mathbf{w}^T\mathbf{w} - 1)$$

- Set gradient with respect to $\mathbf{w}$ to zero: $\mathbf{C}\mathbf{w} - \lambda\mathbf{w} = 0$

$$\mathbf{C}\mathbf{w} = \lambda\mathbf{w}$$

Face recognition: Eigenfaces

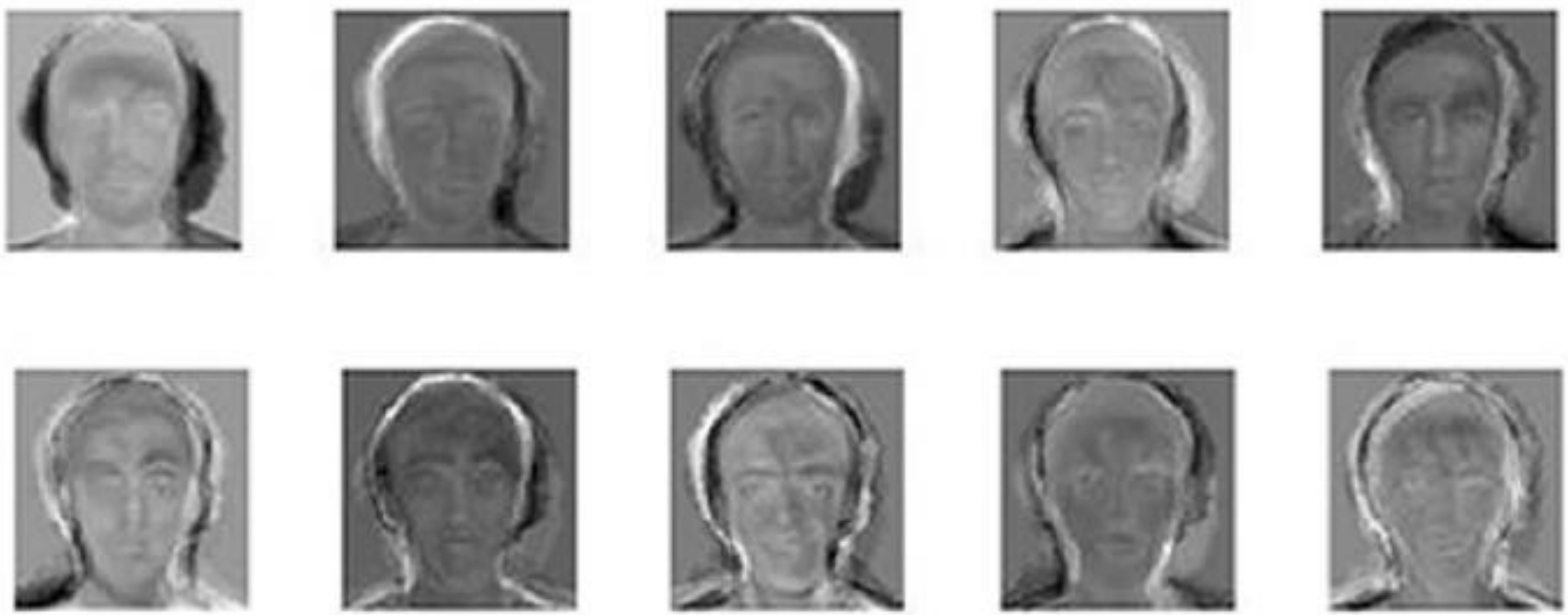
- We can move through the PCA subspace to generate new faces:



$$\tilde{\mathbf{x}} = \bar{\mathbf{x}} + \sum_{i=1}^{p} z_i \mathbf{w}_i$$

Face recognition: Eigenfaces

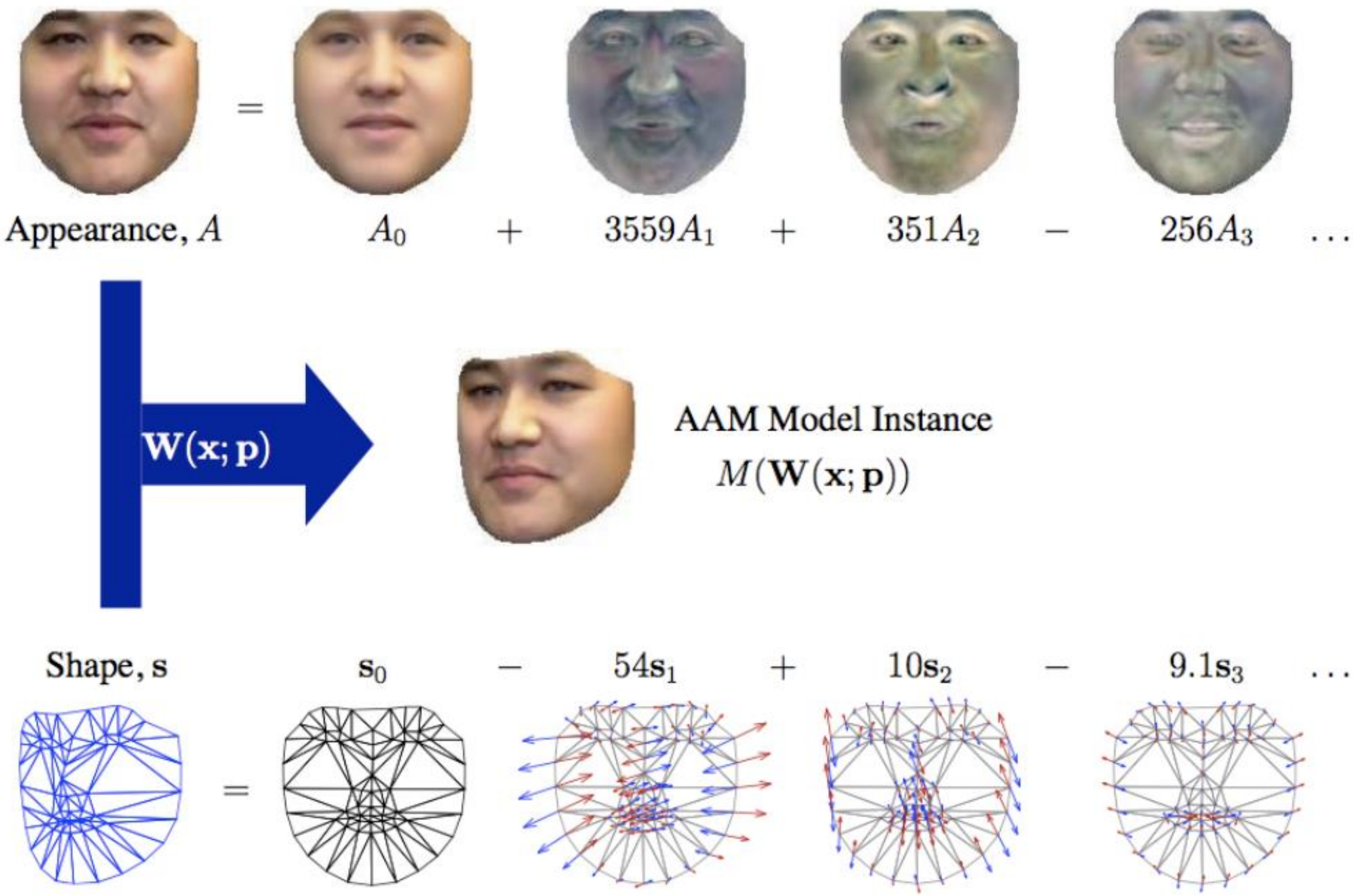
- We can visualize the eigenfaces to show the main sources of variation:



- We may use $\mathbf{z} = \left[\mathbf{w}_1^{\mathrm{T}}\mathbf{x}, \ldots, \mathbf{w}_p^{\mathrm{T}}\mathbf{x}\right]^{\mathrm{T}}$ as features for identity recognition

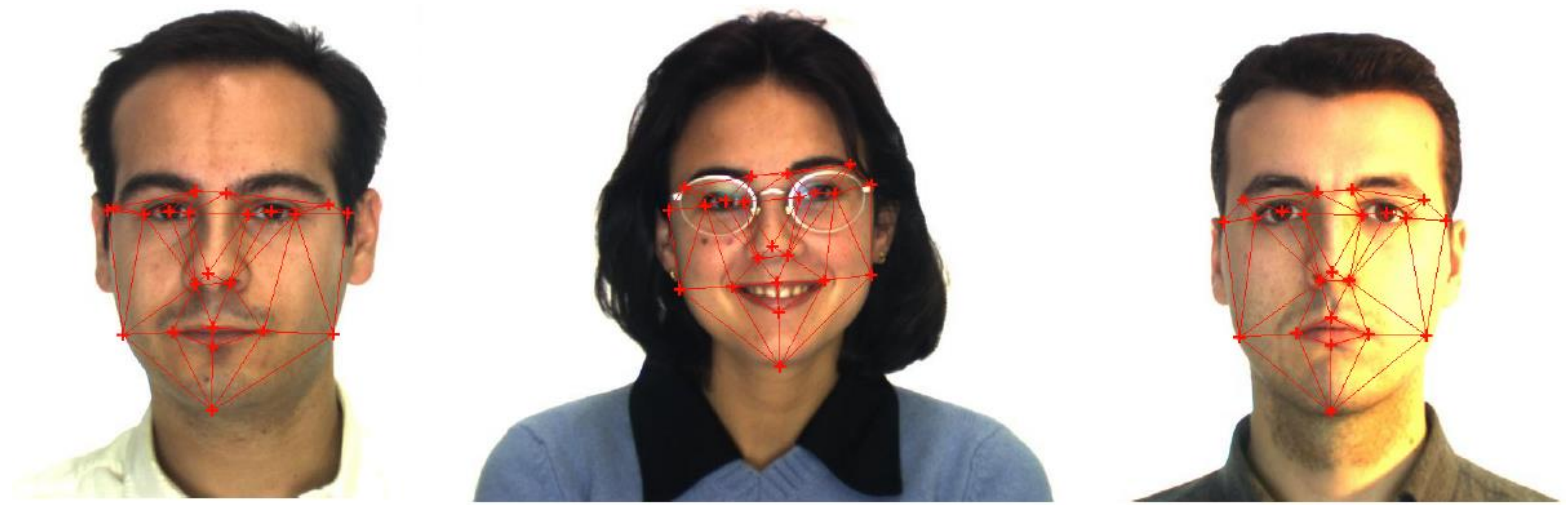
Face recognition: Active appearance models

- Separates face variations into *shape* and *texture* variation:



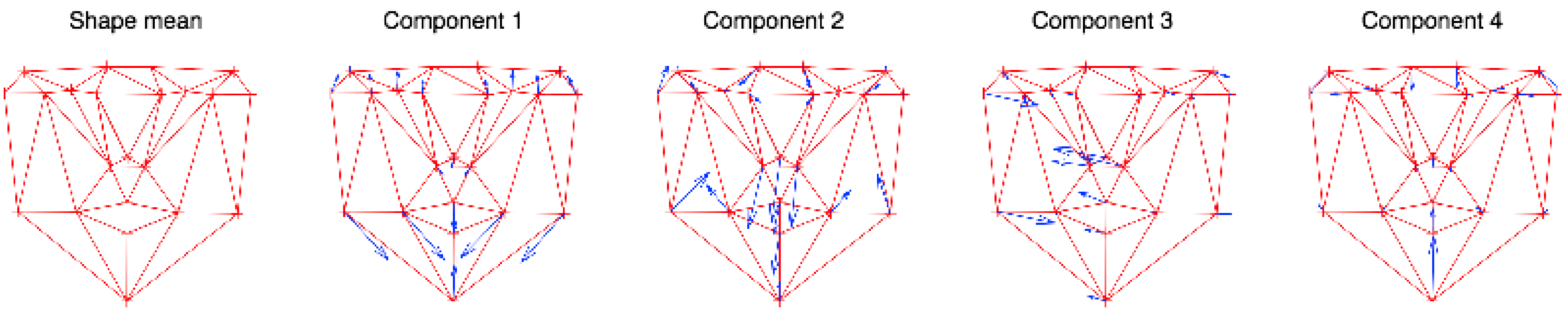
Face recognition: Active appearance models

- Gather a data set of face images with annotated feature points
- Remove translations and rotations from point annotations (Procrustes alignment)
- Learn *point distribution model* and *texture model* from the data

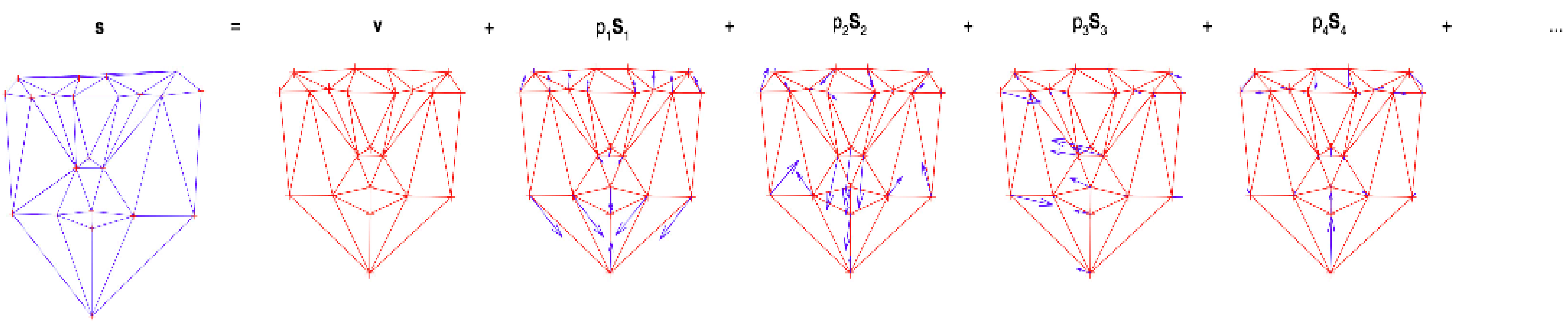


Face recognition: Active appearance models

- *Point distribution model* is obtained using PCA:



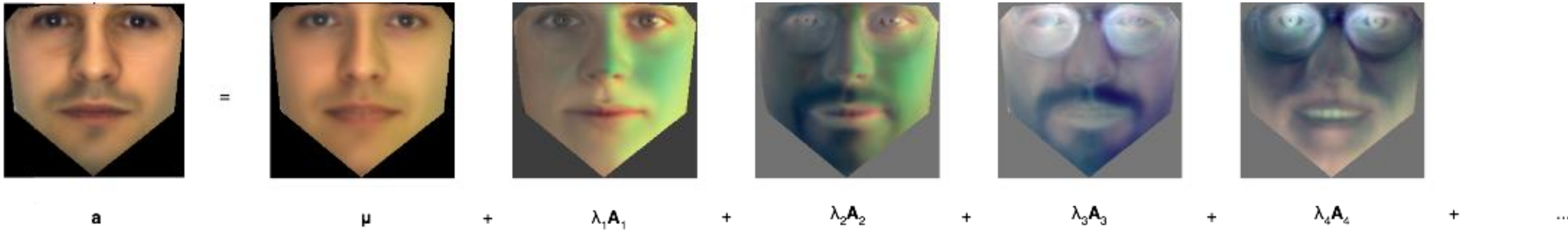
- New facial shape is generated by linearly combining the components:



Face recognition: Active appearance models

- *Texture model* is also obtained using PCA:

- New facial textures are generated using a linear combination of components:



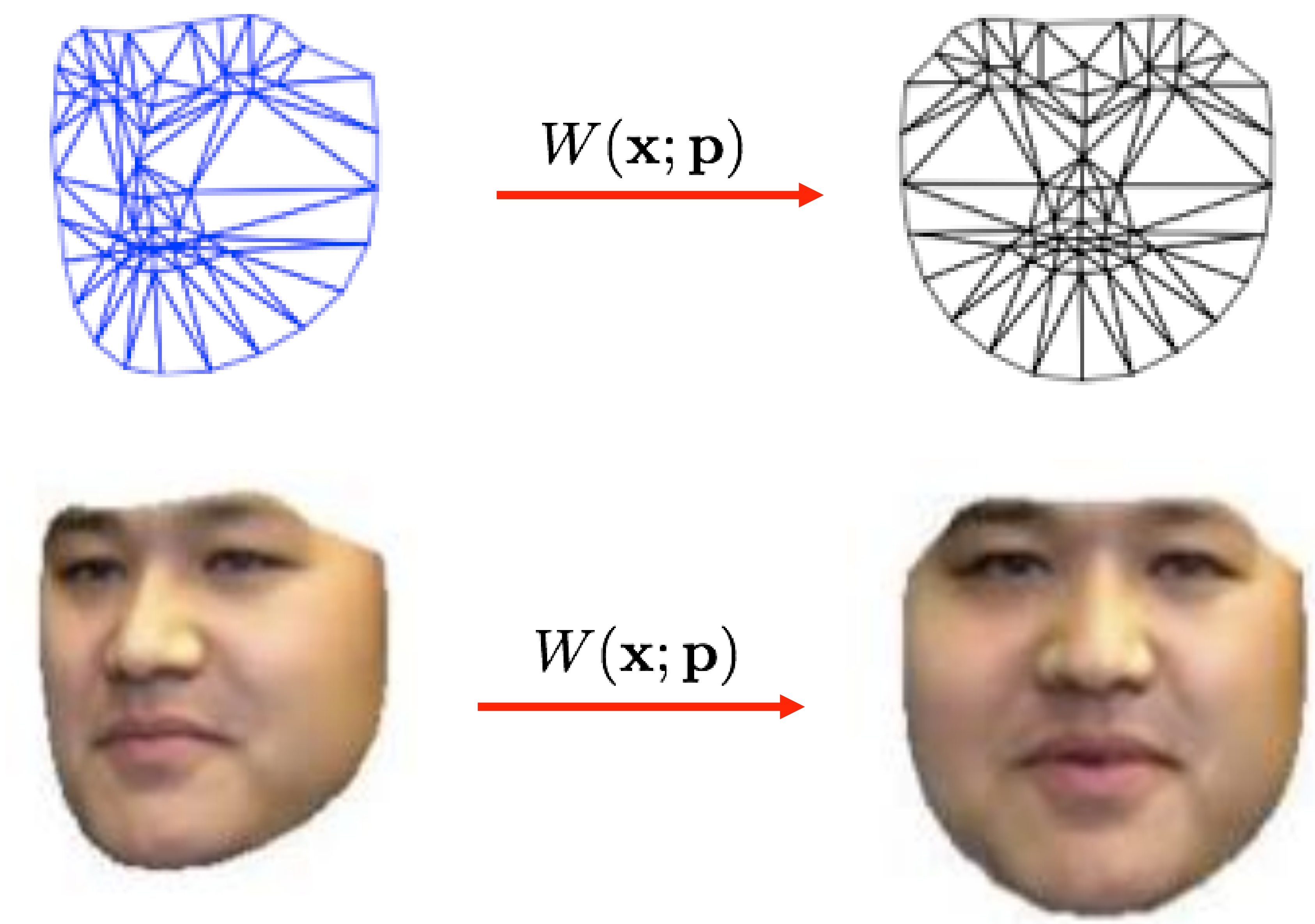
Face recognition: Active appearance models

- We receive a new face image in which we want to measure facial features:

- Now what do we do to fit the active appearance model to this new face?
    - Find parameters of the model that best fit the face (minimizing sum of squared errors) using *Lucas-Kanade algorithm*

Image warp

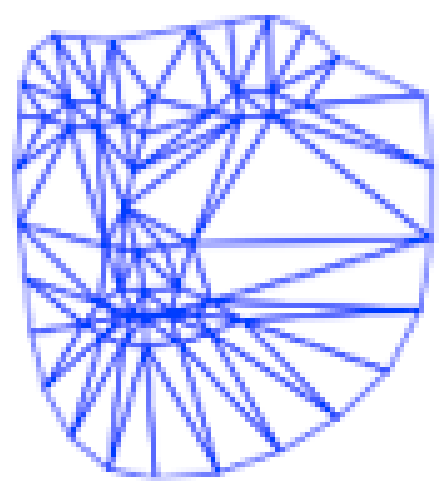
- We can *warp* between an arbitrary shape and the base shape (and vice versa):



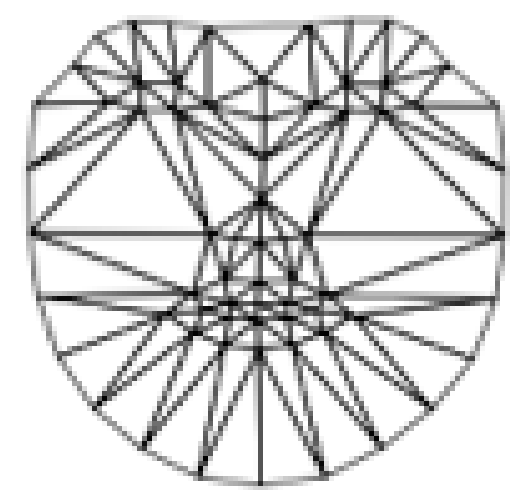
$I(\mathbf{x})$ $I(W(\mathbf{x};\mathbf{p}))$

Image warp

- We can *warp* between an arbitrary shape and the base shape (and vice versa):



$W(\mathbf{x};\mathbf{p})$



This shape is governed by the shape parameters **p**

$W(\mathbf{x};\mathbf{p})$

This is a part of the image on which we are fitting the AAM
(*viz.* the convex hull of the current shape)

$I(\mathbf{x})$

$I(W(\mathbf{x};\mathbf{p}))$

Lucas-Kanade algorithm

- Minimizes sum of squared error w.r.t. $\mathbf{p}$ using *Gauss-Newton* algorithm:

$$\sum_{\mathbf{x}} [A_0(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p}))]^2$$

Mean texture

-

Test image
after warp

Lucas-Kanade algorithm

- Minimizes sum of squared error w.r.t. $\mathbf{p}$ using *Gauss-Newton* algorithm:

$$\sum_{\mathbf{x}} [A_0(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p}))]^2$$

Mean texture

-

Test image
after warp

Goal: Set the shape parameters **p** such that the left image looks as much as possible like the right image

Lucas-Kanade algorithm

- Minimizes sum of squared error w.r.t. $\mathbf{p}$ using *Gauss-Newton* algorithm:

$$\sum_{\mathbf{x}} [A_0(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p}))]^2$$

- Iteratively solve for parameter increment $\Delta\mathbf{p}$:

$$\min_{\Delta\mathbf{p}} \sum_{\mathbf{x}} [A_0(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}))]^2$$

Lucas-Kanade algorithm

- Minimizes sum of squared error w.r.t. $\mathbf{p}$ using *Gauss-Newton* algorithm:

$$\sum_{\mathbf{x}} [A_0(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p}))]^2$$

- Iteratively solve for parameter increment $\Delta\mathbf{p}$:

$$\min_{\Delta\mathbf{p}} \sum_{\mathbf{x}} [A_0(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}))]^2$$

- This is strongly non-linear, so write down *first-order Taylor expansion*:

Lucas-Kanade algorithm

- Minimizes sum of squared error w.r.t. $\mathbf{p}$ using *Gauss-Newton* algorithm:

$$\sum_{\mathbf{x}} [A_0(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p}))]^2$$

- Iteratively solve for parameter increment $\Delta\mathbf{p}$:

$$\min_{\Delta\mathbf{p}} \sum_{\mathbf{x}} [A_0(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}))]^2$$

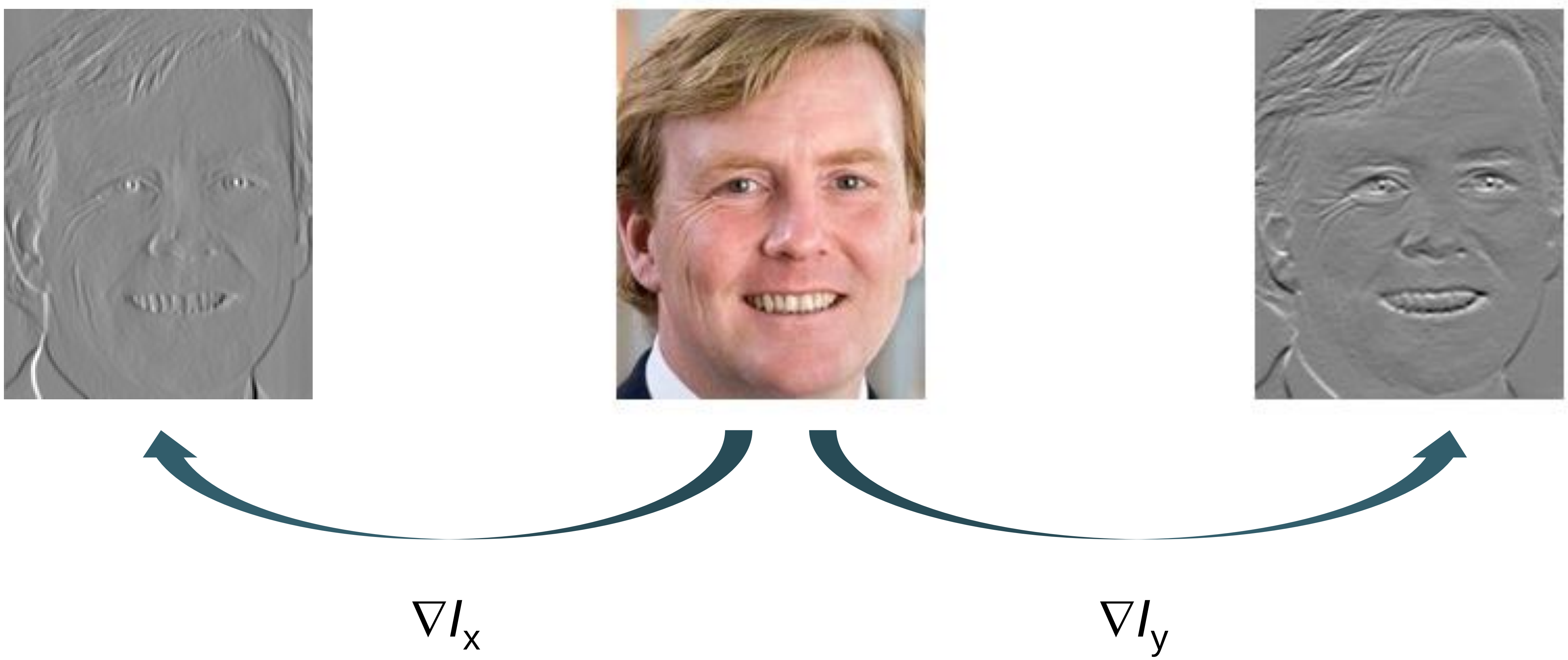
- This is strongly non-linear, so write down *first-order Taylor expansion*:

$$\min_{\Delta\mathbf{p}} \sum_{\mathbf{x}} \left[A_0(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p})) - \nabla I \frac{\partial W}{\partial \mathbf{p}} \Delta\mathbf{p} \right]^2$$

- As expected, this is a standard *linear least squares* problem

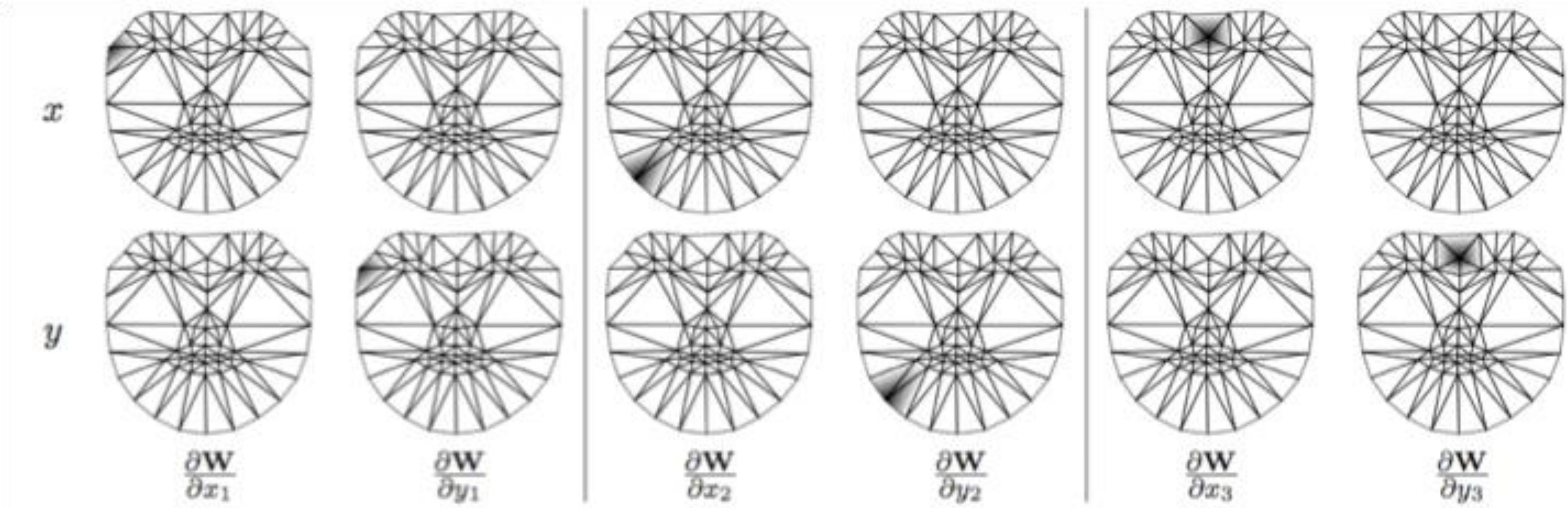
Gradient Images

- Illustration of the image gradient $\boldsymbol{\nabla} I$



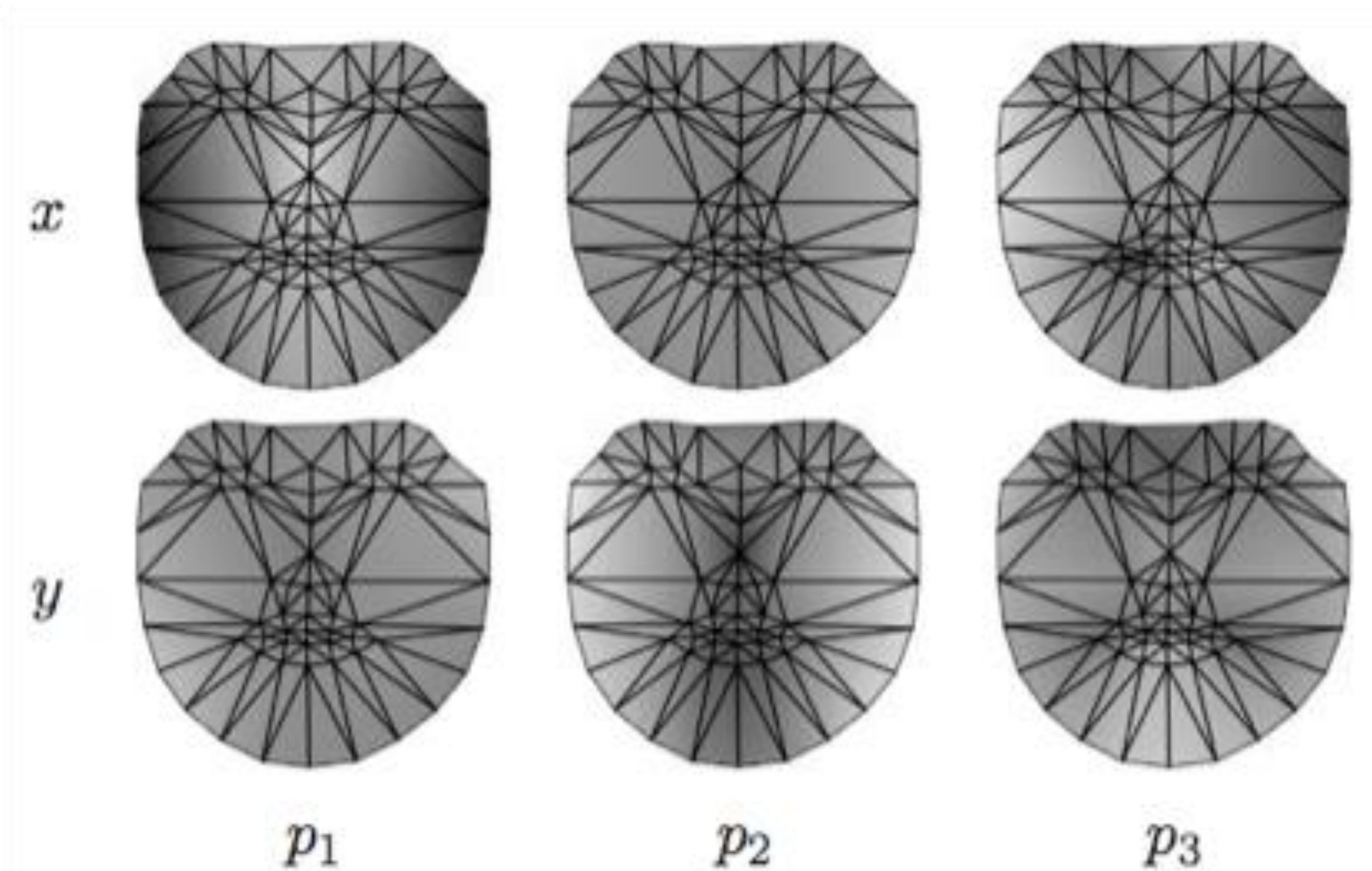
Lucas-Kanade Algorithm

- Illustration of the warp Jacobian: $$\frac{\partial W}{\partial \mathbf{p}} = \frac{\partial W}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{p}}$$



Lucas-Kanade Algorithm

- Illustration of the warp Jacobian: $$\frac{\partial W}{\partial \mathbf{p}} = \frac{\partial W}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{p}}$$



Lucas-Kanade algorithm

- Closed-form solution for the parameter update:

$$\Delta\mathbf{p} = \mathbf{H}^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[A_0(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p})) \right]$$

- Herein, $\mathbf{H}$ is the Gauss-Newton approximation to the Hessian:

$$\mathbf{H} = \sum_{\mathbf{x}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}} \right]$$

Lucas-Kanade algorithm

- Closed-form solution for the parameter update:

$$\Delta\mathbf{p} = \mathbf{H}^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[A_0(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p})) \right]$$

- Herein, $\mathbf{H}$ is the Gauss-Newton approximation to the Hessian:

$$\mathbf{H} = \sum_{\mathbf{x}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}} \right]$$

- Every iteration requires computation of warp Jacobian and Hessian

Lucas-Kanade Algorithm

1. Warp I with $\mathbf{W}(\mathbf{x};\mathbf{p}) \Rightarrow I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
2. Compute error image $A_o(x) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
3. Warp gradient of I to compute ∇I
4. Evaluate Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
5. Compute Hessian
6. Compute $\Delta\mathbf{p}$
7. Update parameters $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$

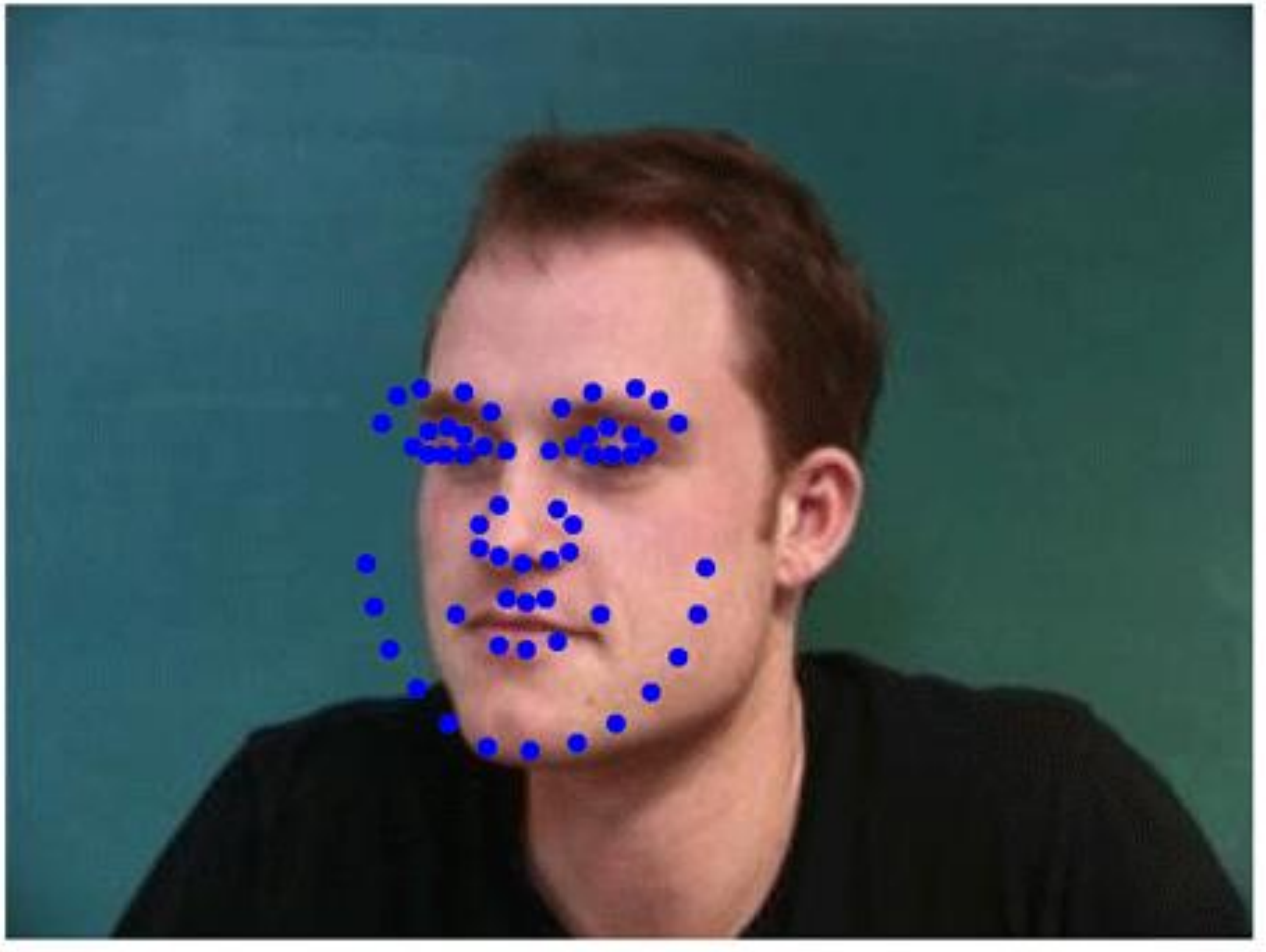
Face recognition: Active appearance models

Face recognition: Active appearance models

Face recognition: Active appearance models

Face recognition: Active appearance models

- Illustration of fitting a shape model:



Face recognition: Active appearance models

Face recognition: Active appearance models

Face recognition and expression analysis

- Facial feature points (landmarks) can be used for a number of tasks:
  - *Facial expression analysis*:
    - Measure variations of landmark locations over time (shape variation); use texture features to measure presence of wrinkles (texture variation), *etc.*

Face recognition and expression analysis

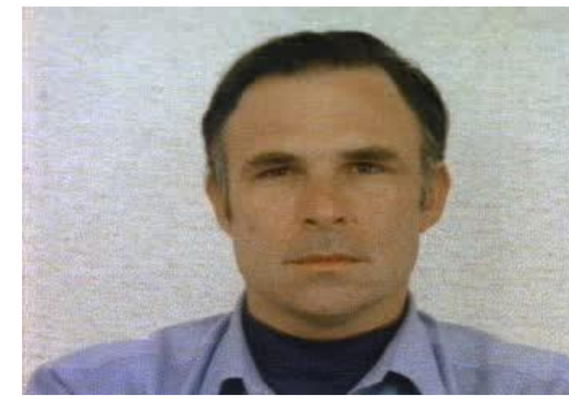
- Facial feature points (landmarks) can be used for a number of tasks:
  - *Facial expression analysis*:
    - Measure variations of landmark locations over time (shape variation); use texture features to measure presence of wrinkles (texture variation), *etc.*
  - *Facial identity recognition* or *face verification* (passport control):
    - Measure characteristics that are invariant under expressions but person-specific: inter-ocular distance, relative position of nose, *etc.*
    - Build skin models:

Example: Recognition of Action Units (FACS)

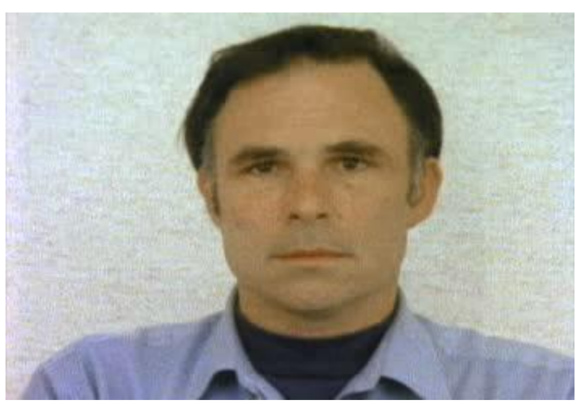
AU 1
Inner brow raise

AU 2
Outer brow raise

AU 4
Brow lower



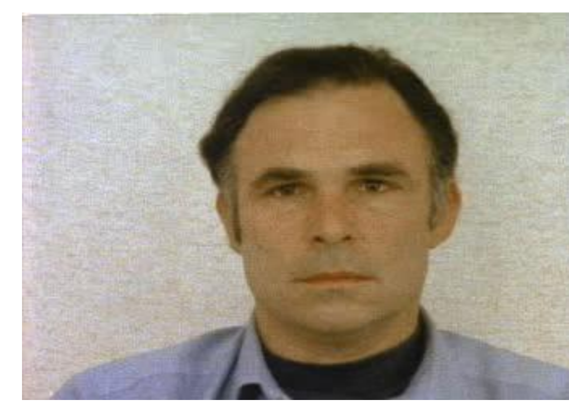
AU 6
Cheek raise



AU 9
Nose wrinkler

AU 12
Lip corner pull

AU 15
Lip corner depress



AU 20
Lip strecher

Example: Recognition of Action Units (FACS)

- Inner brow raiser:

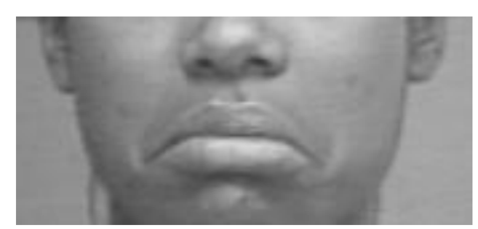
- Outer brow raiser:

- Brow lowerer:

- Upper lid raiser:

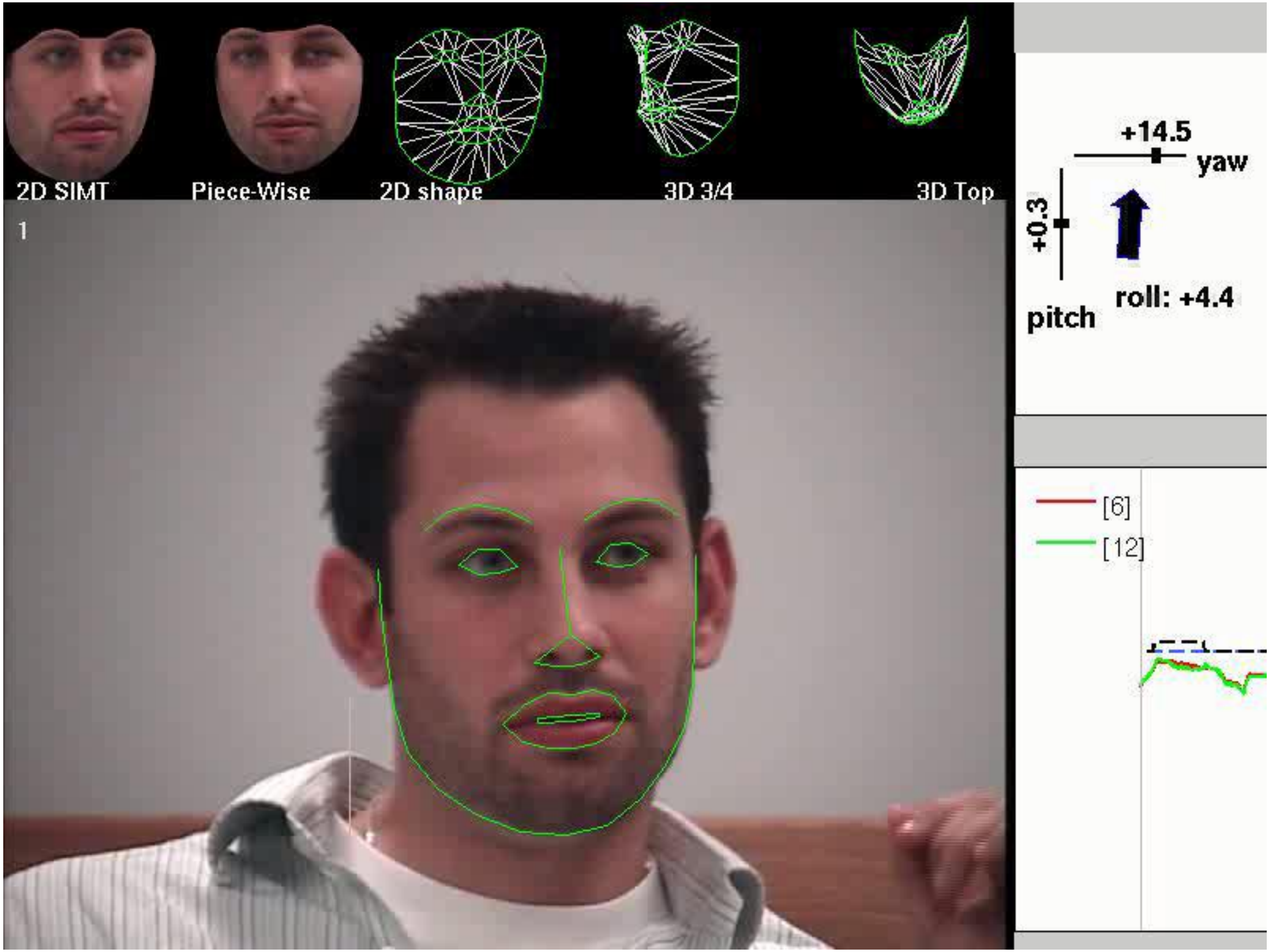
- Nose wrinkler:

- Lip corner depressor:



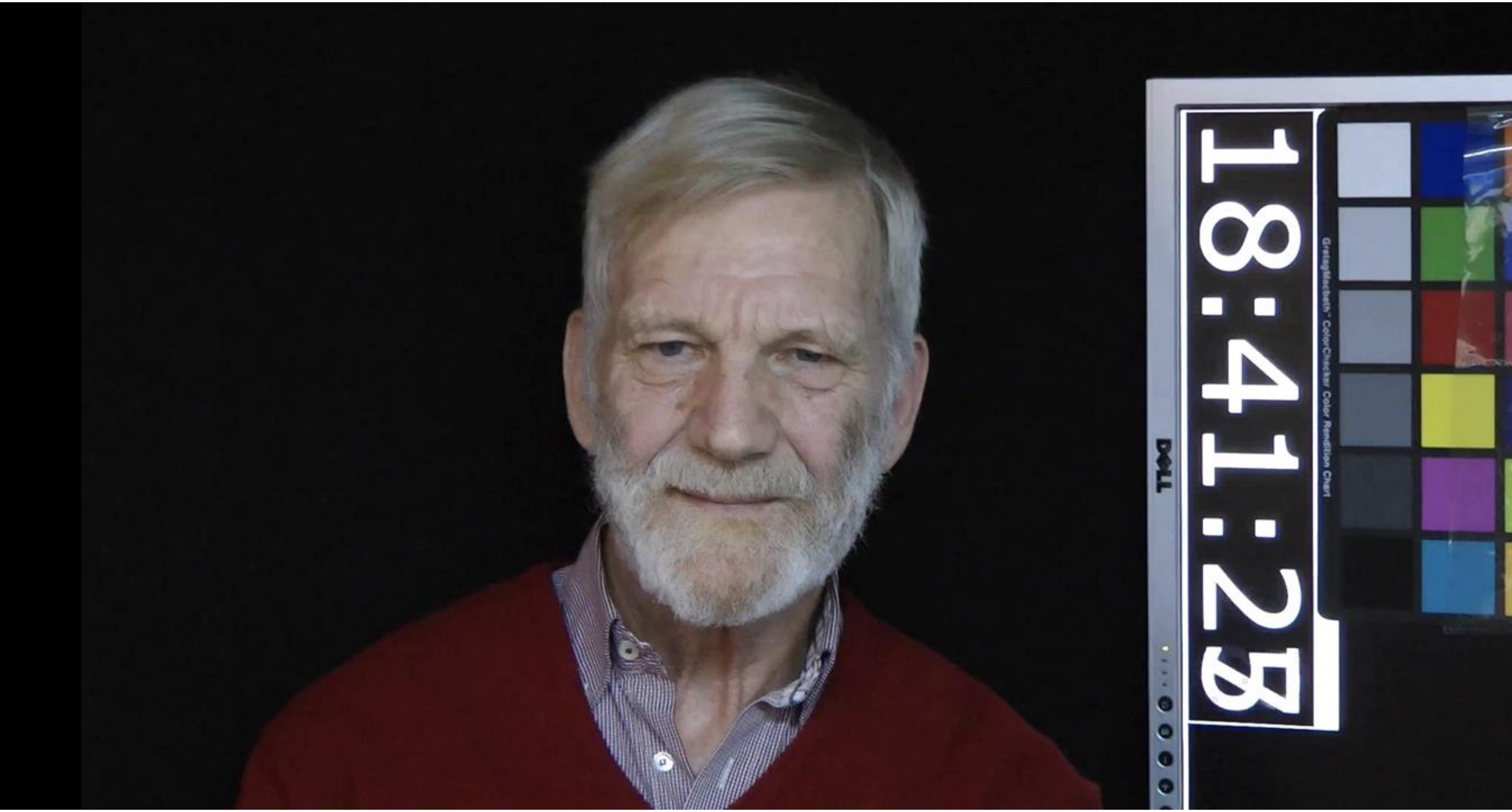
- Etcetera...

Example: Recognition of Action Units (FACS)

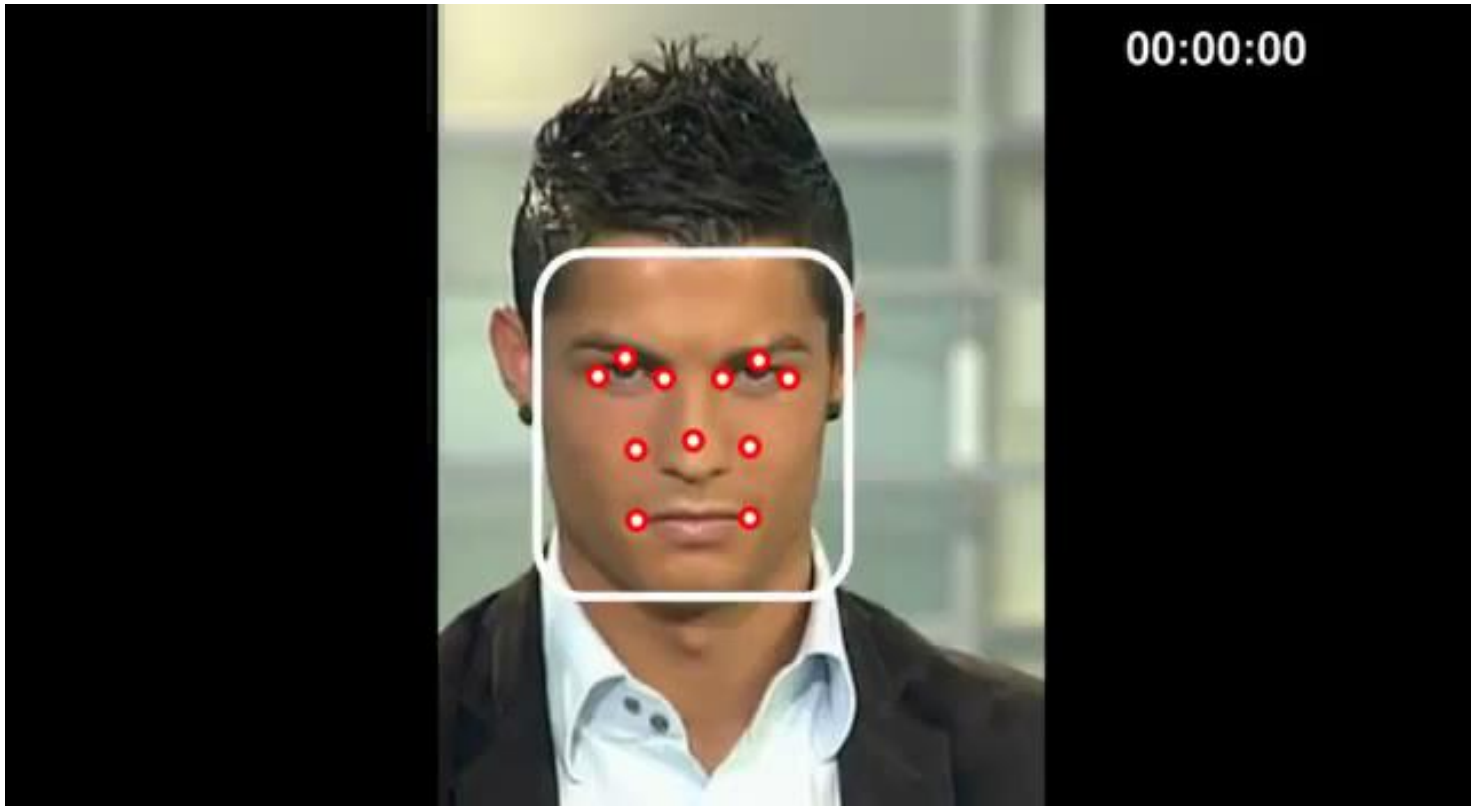


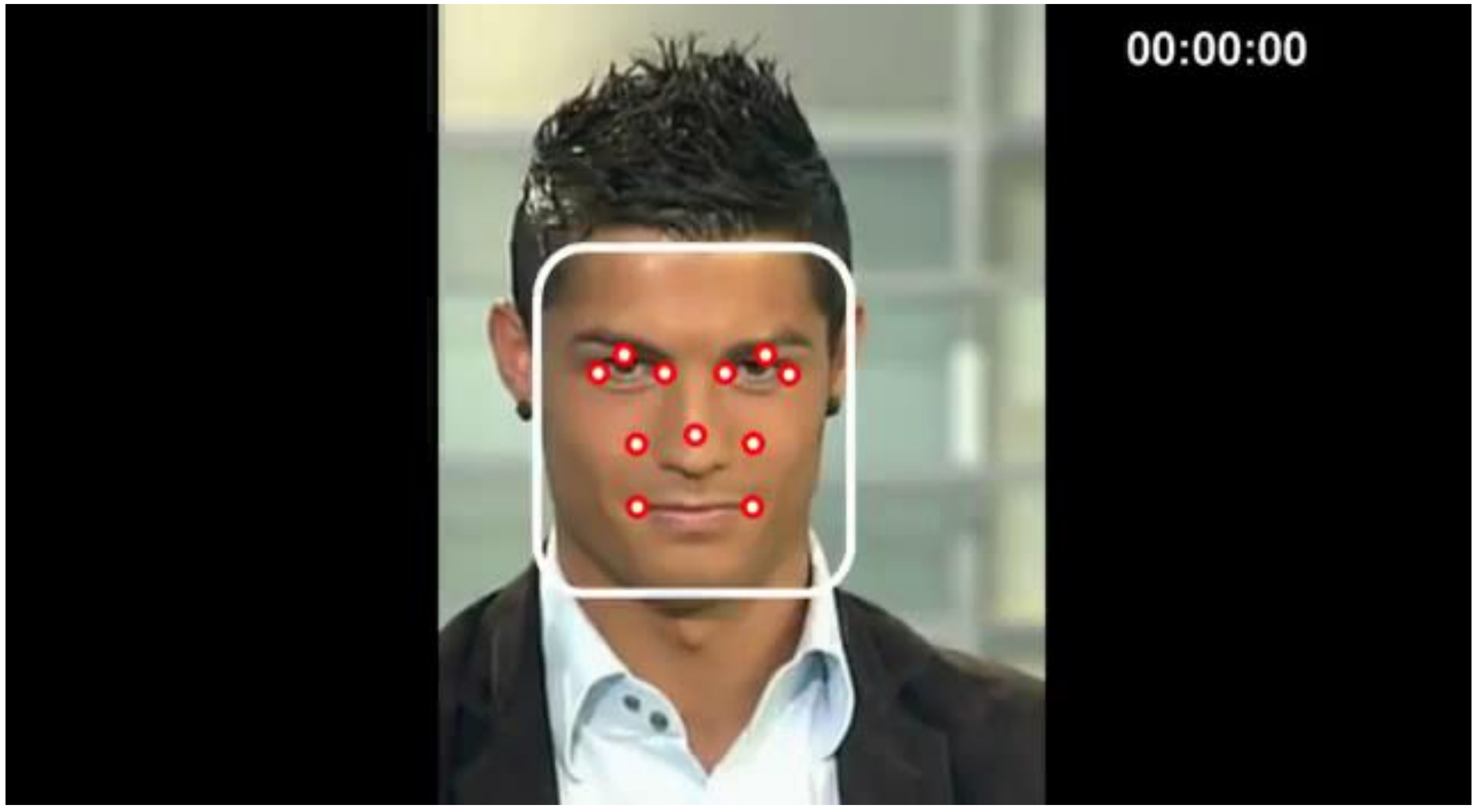
Example: Expression Recognition

Example: Detection of Expression Spontaneity



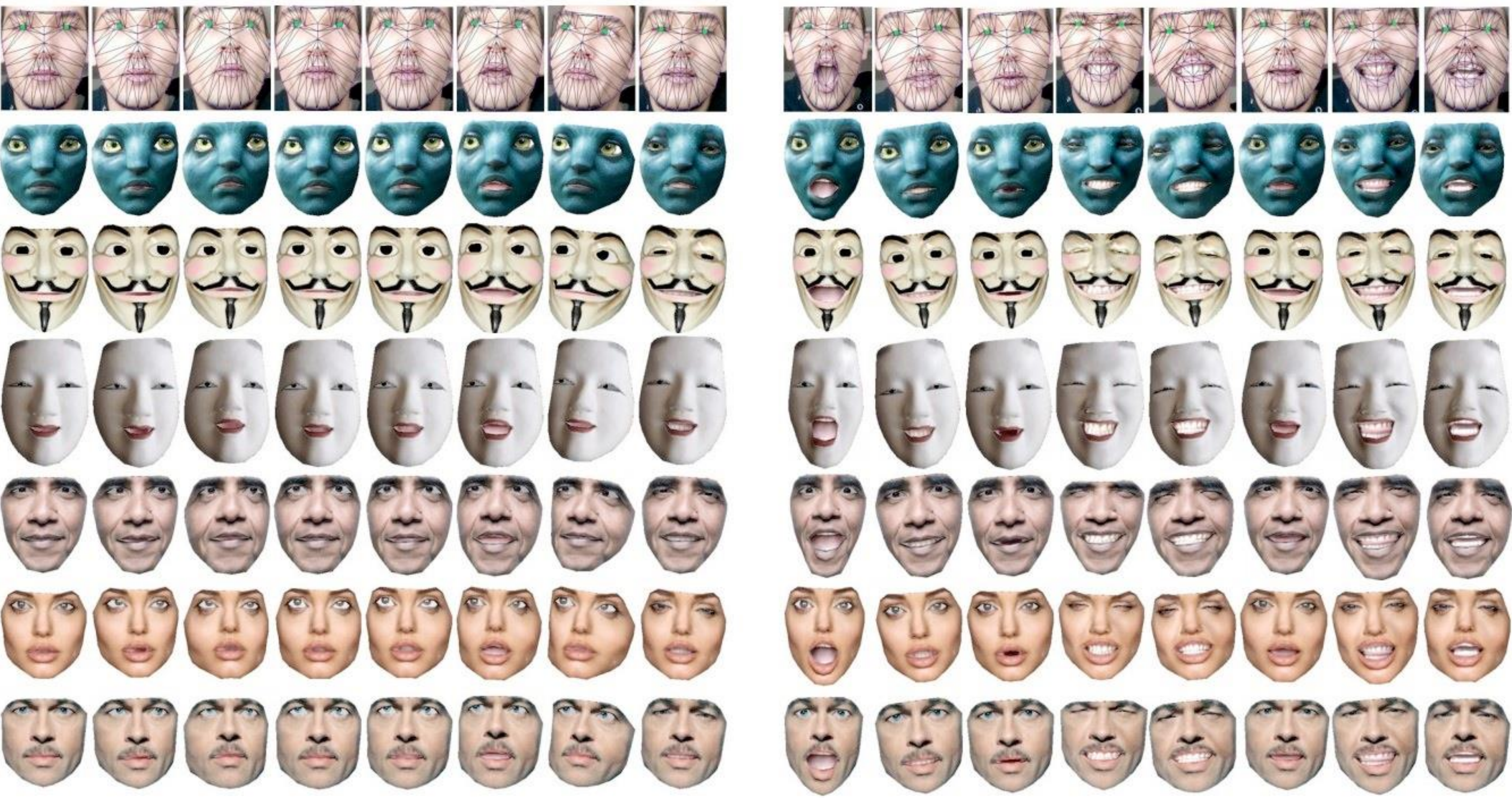
Example: Detection of Expression Spontaneity



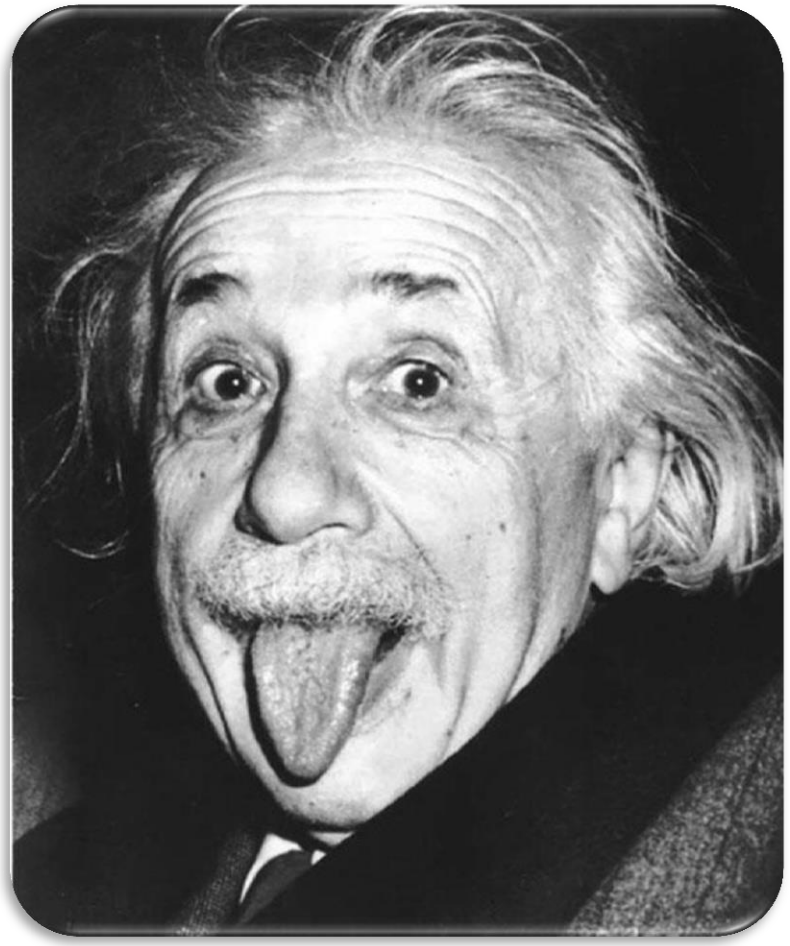


Example: Expression cloning

Example: Expression cloning



MSc Projects



- Are you interested in face processing?
  - Face Tracking
  - Expression Analysis
  - Age Estimation
  - Face Recognition
  - Kinship Verification/Recognition
  - Automatic Assessment of Depression
  - And more… (only limit is your imagination)

Reading material:
Section 14.1 and 14.2
Section 1 and 2 of *“Lucas-Kanade 20 Years on”*