Rapid Facial Feature Detection in iOS

Instructor - Simon Lucey

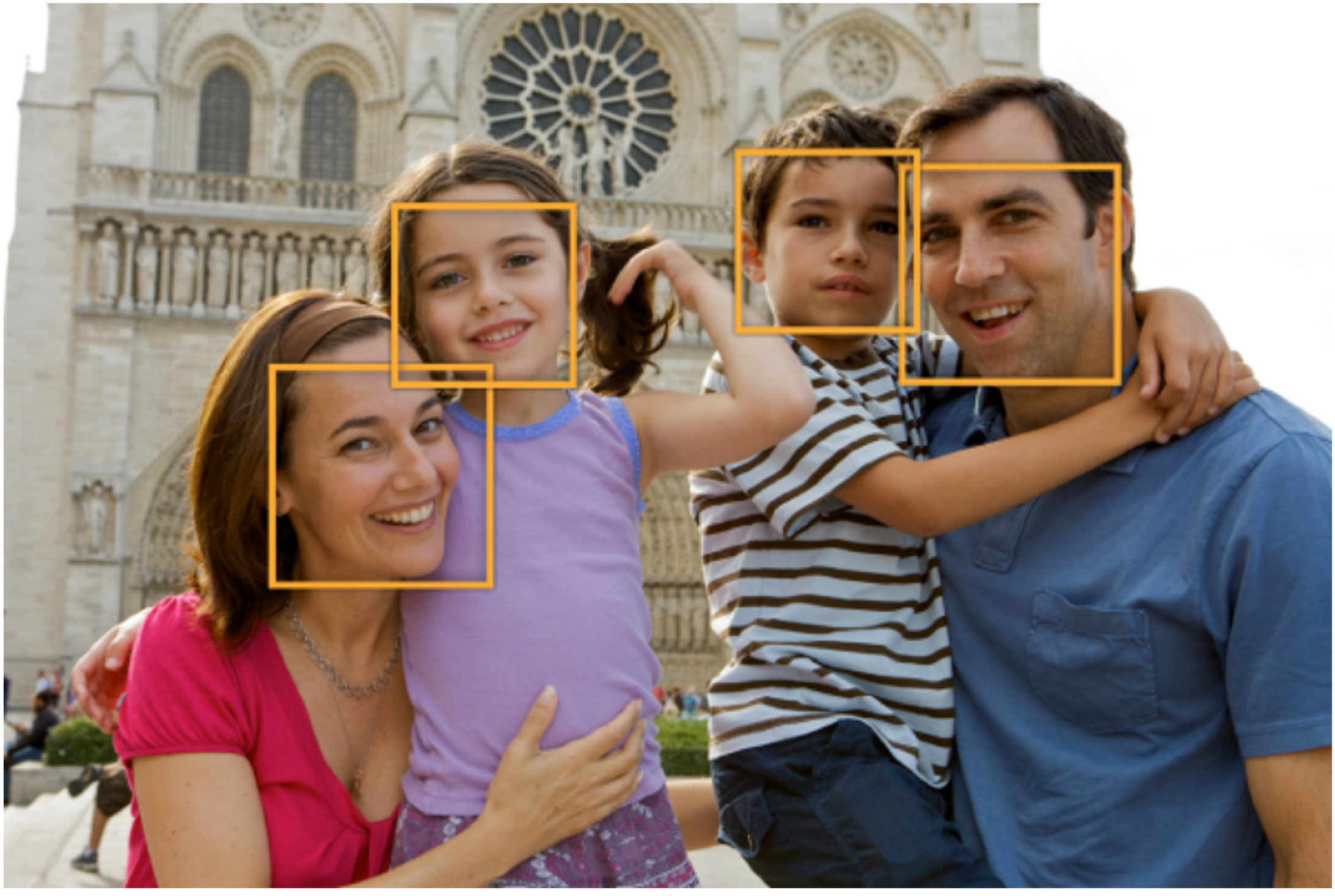
16-423 - Designing Computer Vision Apps

Carnegie Mellon
THE ROBOTICS INSTITUTE

Today

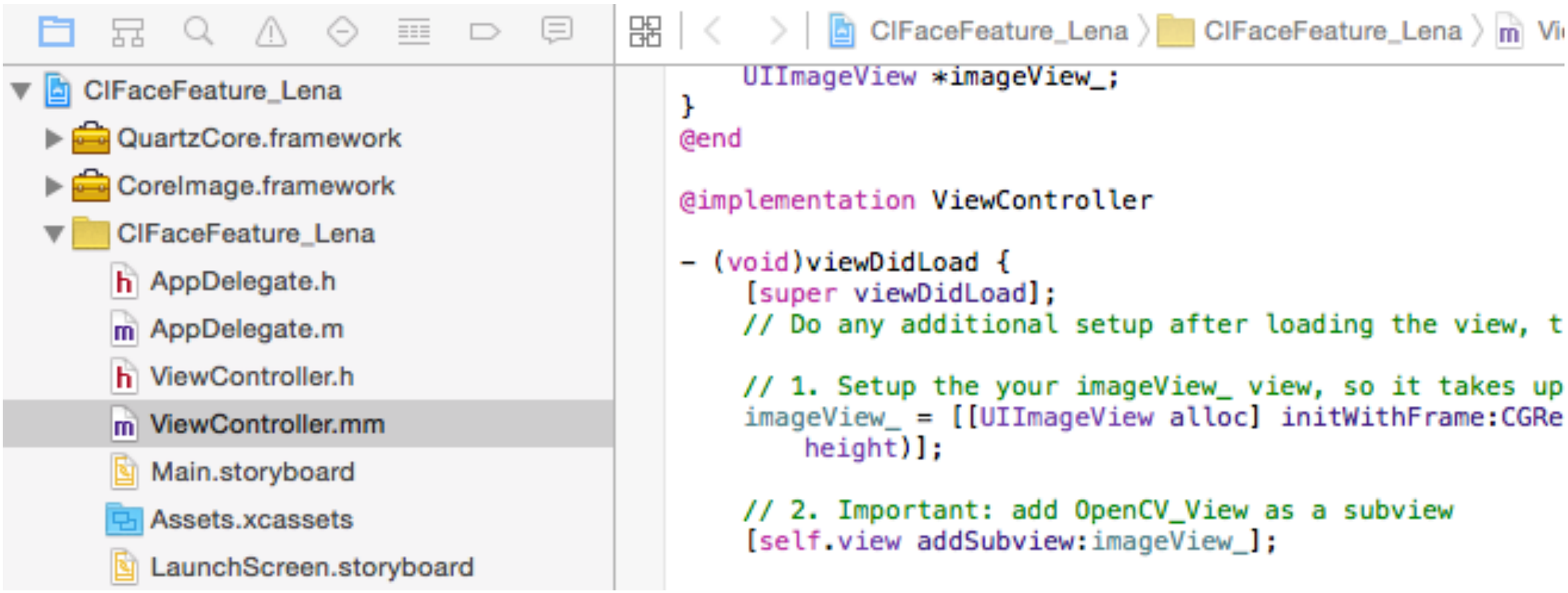
- Facial Feature Detection in iOS.
- Active Appearance Models.

Face Detection



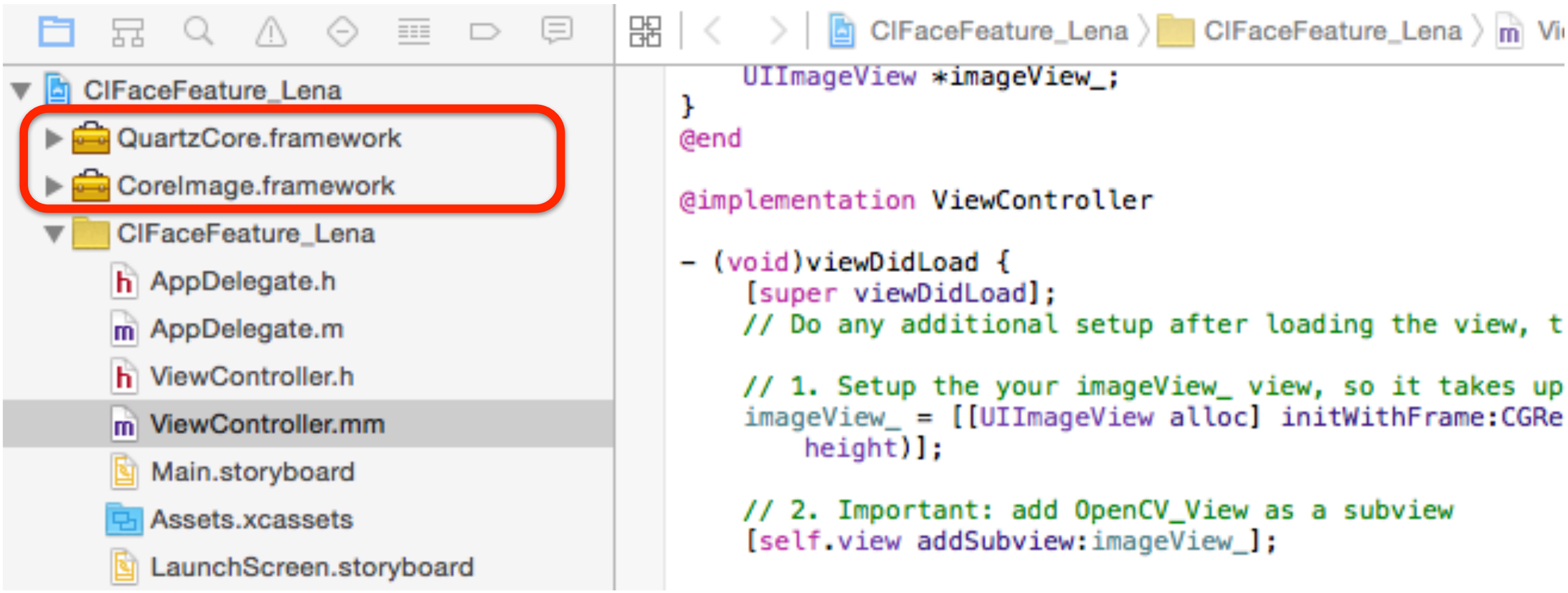
Face Detection in iOS

- In iOS face detection comes built in, and can be performed much more efficiently than standard OpenCV.
- Utilizes the **QuartzCore** and **CoreImage** frameworks within the project.

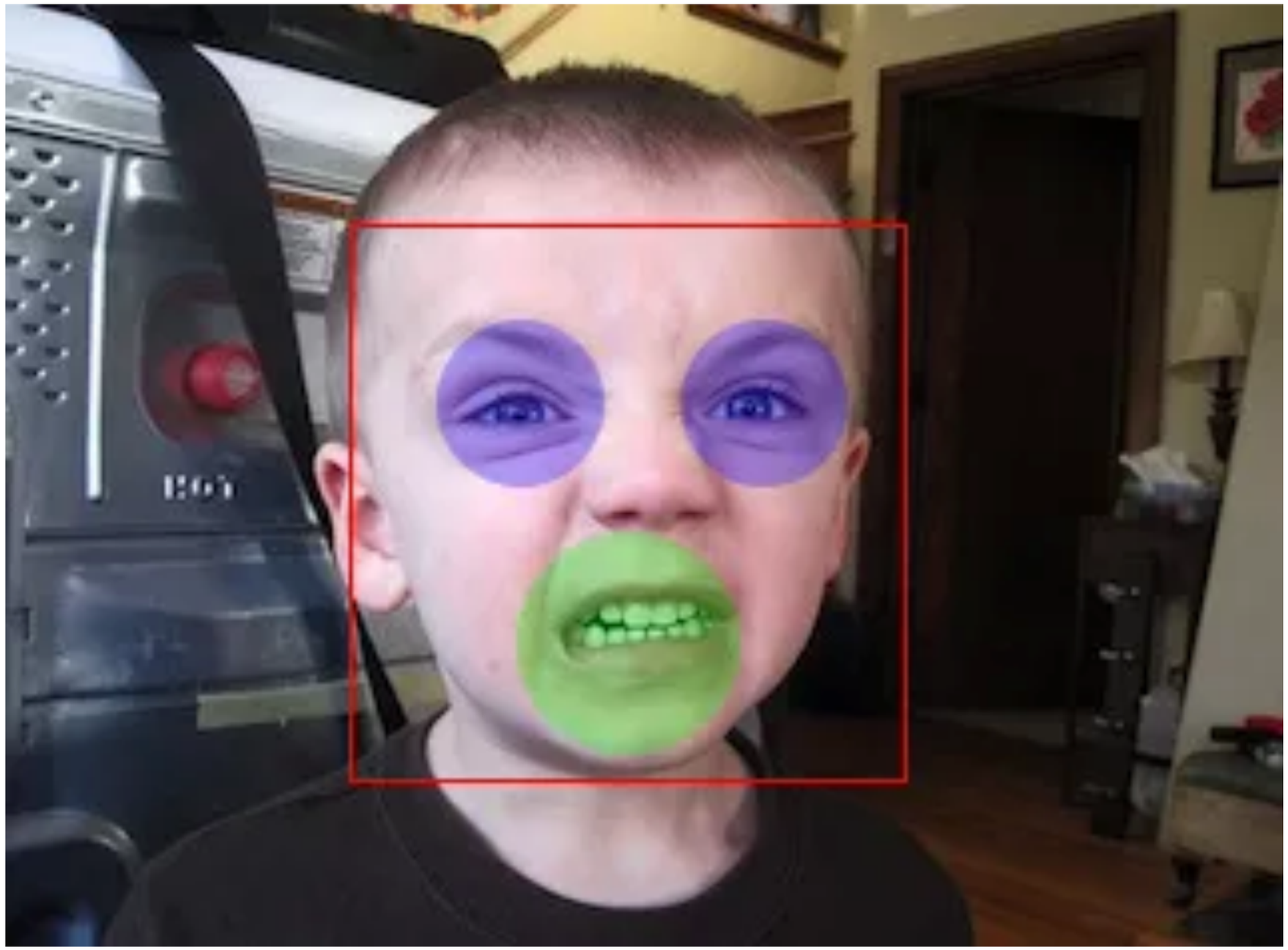


Face Detection in iOS

- In iOS face detection comes built in, and can be performed much more efficiently than standard OpenCV.
- Utilizes the **QuartzCore** and **CoreImage** frameworks within the project.



Facial Feature Detection



CIFaceFeature

CIFaceFeature

> IMPORTANT
>
> This is a preliminary document for an API or technology in development. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein for use on Apple-branded products. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future betas of the API or technology.

Inherits From

- NSObject
  - CIFeature
    - CIFaceFeature

Conforms To

NSObject

Import Statement

```
OBJECTIVE-C
@import CoreImage;
```

Availability

Available in iOS 5.0 and later

A `CIFaceFeature` object describes a face detected in a still or video image. Its properties provide information about the face's eyes and mouth. A face object in a video can also have properties that track its location over time—tracking ID and frame count.

CIFaceFeature

Locating Faces

`bounds` *Property*

`hasFaceAngle` *Property*

`faceAngle` *Property*

Identifying Facial Features

`hasLeftEyePosition` *Property*

`hasRightEyePosition` *Property*

`hasMouthPosition` *Property*

`leftEyePosition` *Property*

`rightEyePosition` *Property*

`mouthPosition` *Property*

`hasSmile` *Property*

`leftEyeClosed` *Property*

`rightEyeClosed` *Property*

CIFaceFeature Example

- We are now going to demonstrate a simple example of face detection in iOS.
- On your browser please go to the address,

https://github.com/slucey-cs-cmu-edu/CIFaceFeature_Lena

- Or better yet, if you have git installed you can type from the command line.

```
$ git clone https://github.com/slucey-cs-cmu-edu/CIFaceFeature_Lena.git
```

CIFaceFeature Example

```
36      // 3. Load the image and display
37      UIImage *image = [UIImage imageNamed:@"lena.png"];
38      if(image != nil) imageView_.image = image; // Display the image if it is there....
39      else cout << "Cannot read in the file" << endl;
40
41      // 4. Initialize the face detector and its options
42      NSDictionary *detectorOptions = @{ CIDetectorAccuracy : CIDetectorAccuracyLow };
43      CIDetector *faceDetector = [CIDetector detectorOfType:CIDetectorTypeFace context:nil options:detectorOptions
44
45      // 5. Apply the image through the face detector
46      NSArray *features = [faceDetector featuresInImage:[CIImage imageWithCGImage: [image CGImage]]];
47
48      // 6. Loop through each detected face
49      for(CIFaceFeature* faceFeature in features) {
50
51          // Check facial features
52
53          // Smile
54          if(faceFeature.hasSmile) cout << "Lena is Smiling" << endl;
55          else cout << "Lena is Not Smiling" << endl;
56
57          // Head angle
58          if(faceFeature.hasFaceAngle) cout << "Lena's head is at an angle of " << faceFeature.faceAngle << endl;
59          else cout << "Lena's head is not at an angle" << endl;
60
61          // Left Eye location
62          CGPoint lEye = faceFeature.leftEyePosition;
63          if(faceFeature.hasLeftEyePosition)
64              cout << "Lena's left eye is positioned at (" << lEye.x << "," << lEye.y << ")" << endl;
65          else
66              cout << "Lena's left eye is occluded!!" << endl;
```

CIFaceFeature Example

```
36      // 3. Load the image and display
37      UIImage *image = [UIImage imageNamed:@"lena.png"];
38      if(image != nil) imageView_.image = image; // Display the image if it is there....
39      else cout << "Cannot read in the file" << endl;
40
41      // 4. Initialize the face detector and its options
42      NSDictionary *detectorOptions = @{ CIDetectorAccuracy : CIDetectorAccuracyLow };
43      CIDetector *faceDetector = [CIDetector detectorOfType:CIDetectorTypeFace context:nil options:detectorOptions
44
45      // 5. Apply the image through the face detector
46      NSArray *features = [faceDetector featuresInImage:[CIImage imageWithCGImage: [image CGImage]]];
47
48      // 6. Loop through each detected face
49      for(CIFaceFeature* faceFeature in features) {
50
51          // Check facial features
52
53          // Smile
54          if(faceFeature.hasSmile) cout << "Lena is Smiling" << endl;
55          else cout << "Lena is Not Smiling" << endl;
56
57          // Head angle
58          if(faceFeature.hasFaceAngle) cout << "Lena's head is at an angle of " << faceFeature.faceAngle << endl;
59          else cout << "Lena's head is not at an angle" << endl;
60
61          // Left Eye location
62          CGPoint lEye = faceFeature.leftEyePosition;
63          if(faceFeature.hasLeftEyePosition)
64              cout << "Lena's left eye is positioned at (" << lEye.x << "," << lEye.y << ")" << endl;
65          else
66              cout << "Lena's left eye is occluded!!" << endl;
```

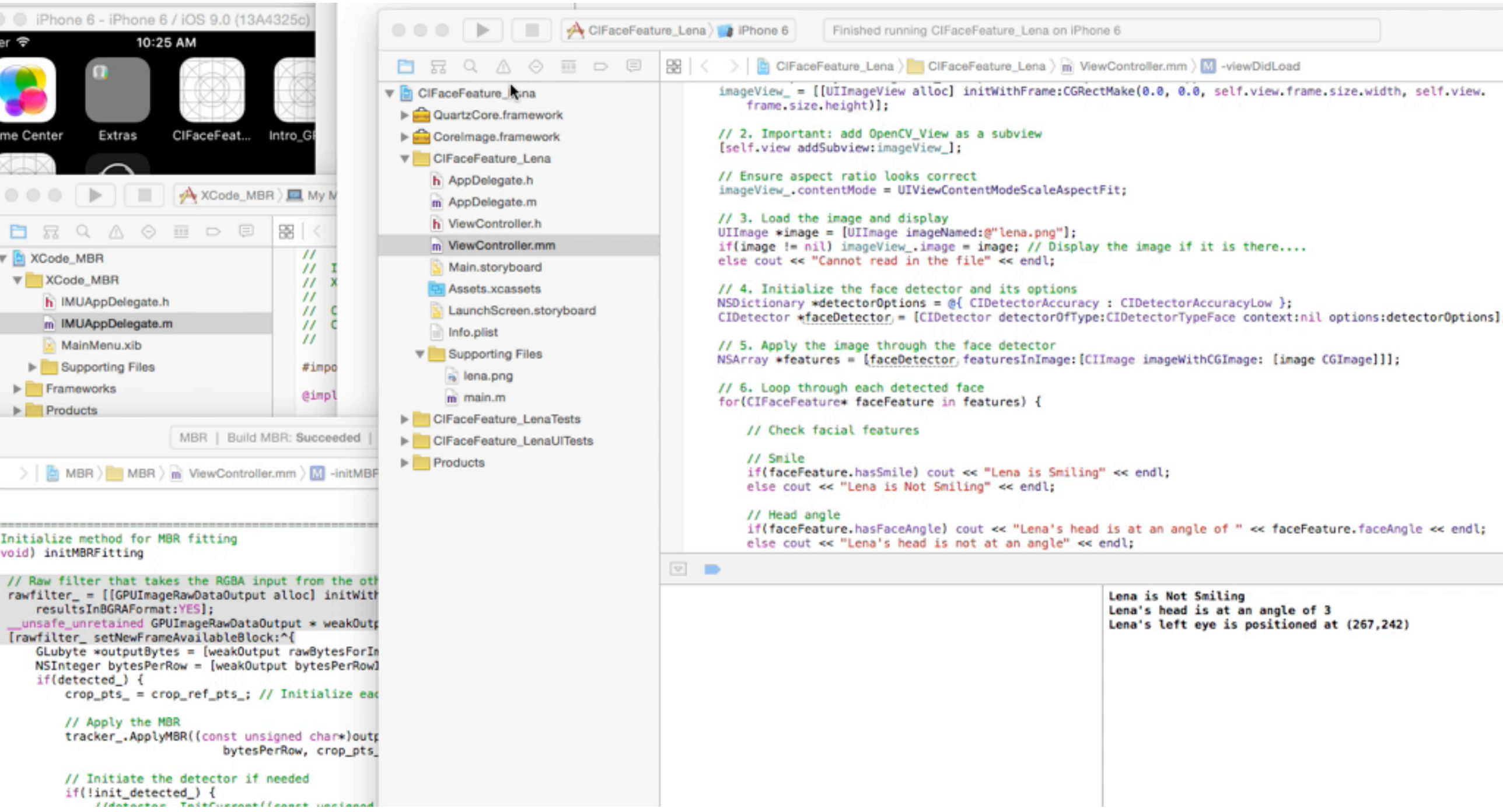
CIFaceFeature Example

```
36      // 3. Load the image and display
37      UIImage *image = [UIImage imageNamed:@"lena.png"];
38      if(image != nil) imageView_.image = image; // Display the image if it is there....
39      else cout << "Cannot read in the file" << endl;
40
41      // 4. Initialize the face detector and its options
42      NSDictionary *detectorOptions = @{ CIDetectorAccuracy : CIDetectorAccuracyLow };
43      CIDetector *faceDetector = [CIDetector detectorOfType:CIDetectorTypeFace context:nil options:detectorOptions
44
45      // 5. Apply the image through the face detector
46      NSArray *features = [faceDetector featuresInImage:[CIImage imageWithCGImage: [image CGImage]]];
47
48      // 6. Loop through each detected face
49      for(CIFaceFeature* faceFeature in features) {
50
51          // Check facial features
52
53          // Smile
54          if(faceFeature.hasSmile) cout << "Lena is Smiling" << endl;
55          else cout << "Lena is Not Smiling" << endl;
56
57          // Head angle
58          if(faceFeature.hasFaceAngle) cout << "Lena's head is at an angle of " << faceFeature.faceAngle << endl;
59          else cout << "Lena's head is not at an angle" << endl;
60
61          // Left Eye location
62          CGPoint lEye = faceFeature.leftEyePosition;
63          if(faceFeature.hasLeftEyePosition)
64              cout << "Lena's left eye is positioned at (" << lEye.x << "," << lEye.y << ")" << endl;
65          else
66              cout << "Lena's left eye is occluded!!" << endl;
```

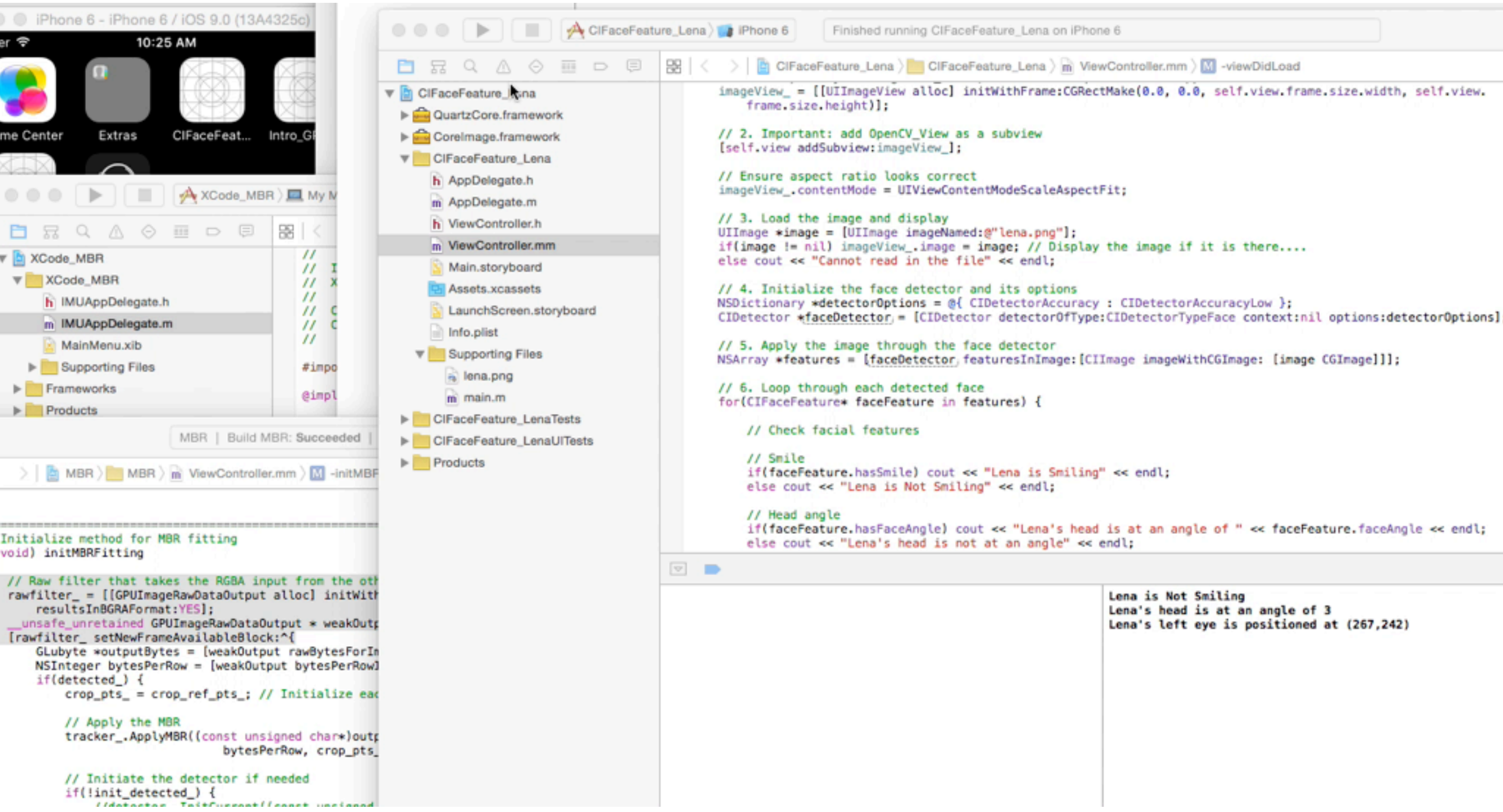
CIFaceFeature Example

```
36      // 3. Load the image and display
37      UIImage *image = [UIImage imageNamed:@"lena.png"];
38      if(image != nil) imageView_.image = image; // Display the image if it is there....
39      else cout << "Cannot read in the file" << endl;
40
41      // 4. Initialize the face detector and its options
42      NSDictionary *detectorOptions = @{ CIDetectorAccuracy : CIDetectorAccuracyLow };
43      CIDetector *faceDetector = [CIDetector detectorOfType:CIDetectorTypeFace context:nil options:detectorOptions
44
45      // 5. Apply the image through the face detector
46      NSArray *features = [faceDetector featuresInImage:[CIImage imageWithCGImage: [image CGImage]]];
47
48      // 6. Loop through each detected face
49      for(CIFaceFeature* faceFeature in features) {
50
51          // Check facial features
52
53          // Smile
54          if(faceFeature.hasSmile) cout << "Lena is Smiling" << endl;
55          else cout << "Lena is Not Smiling" << endl;
56
57          // Head angle
58          if(faceFeature.hasFaceAngle) cout << "Lena's head is at an angle of " << faceFeature.faceAngle << endl;
59          else cout << "Lena's head is not at an angle" << endl;
60
61          // Left Eye location
62          CGPoint lEye = faceFeature.leftEyePosition;
63          if(faceFeature.hasLeftEyePosition)
64              cout << "Lena's left eye is positioned at (" << lEye.x << "," << lEye.y << ")" << endl;
65          else
66              cout << "Lena's left eye is occluded!!" << endl;
67      }
```

CIFaceFeature Example

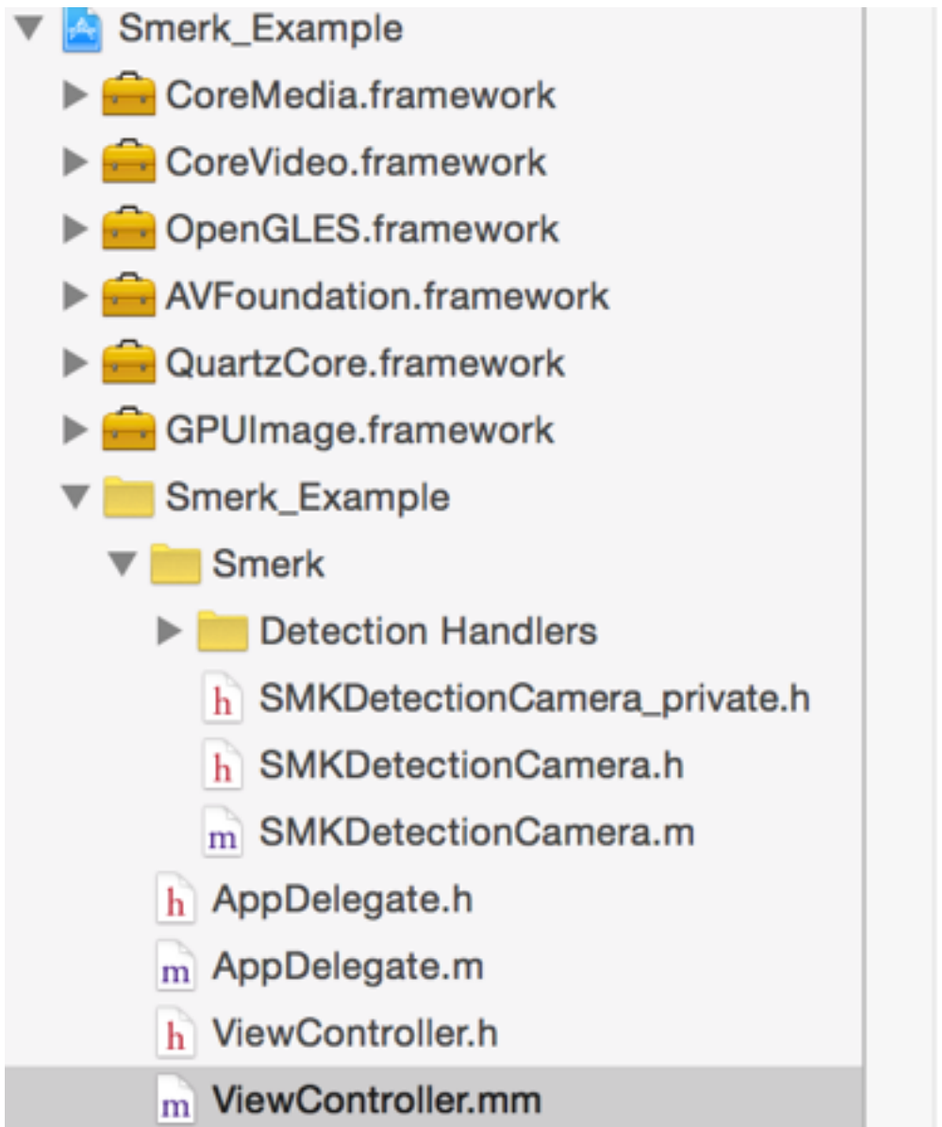


CIFaceFeature Example



Smerk and GPUImage

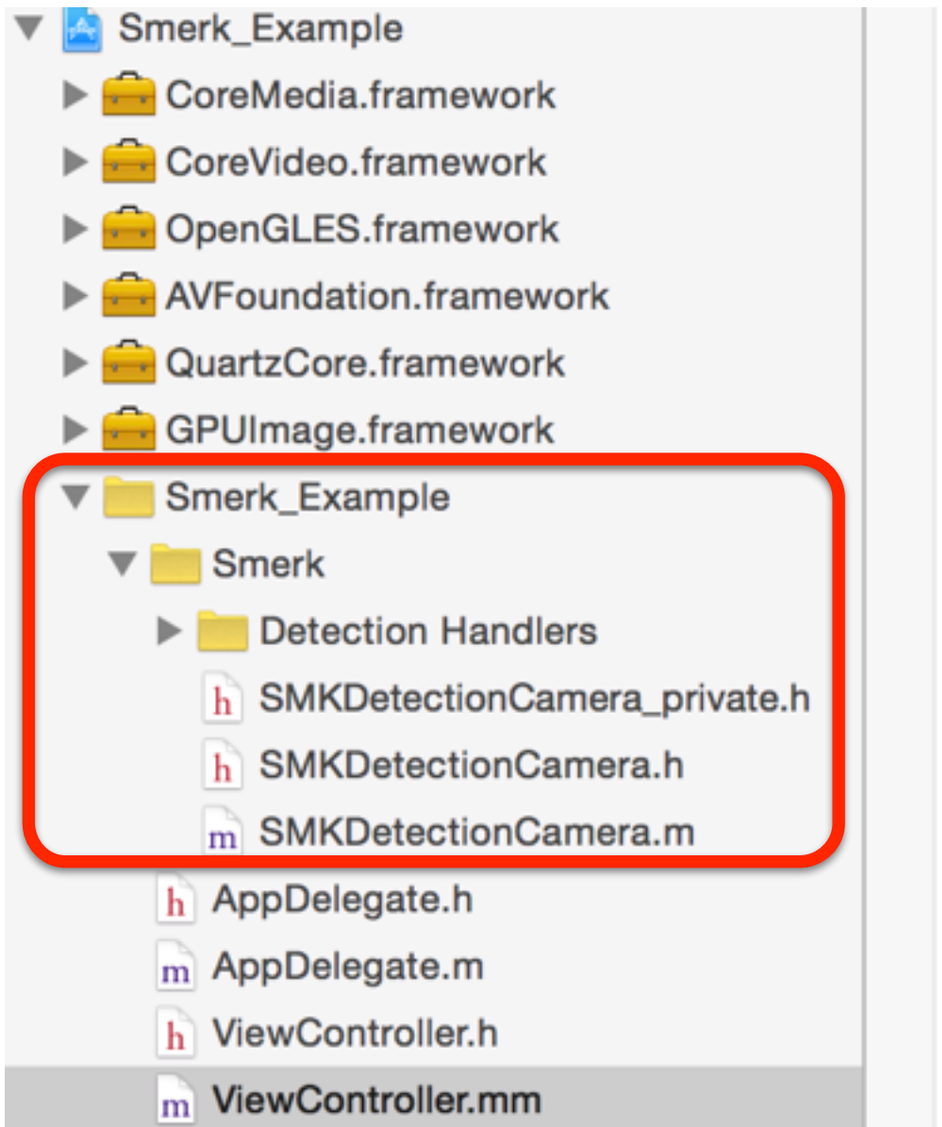
- Recently, an extension to GPUImage was proposed to allow for the utilization of iOS face detection within iOS.
- Called “Smerk” - GitHub project page can be found at:- https://github.com/MattFoley/Smerk




```
// Set the face detector to be us
detector_ = [[SMKDetectionCamera
    AVCaptureDevicePositionFront];
[detector_ setOutputImageOrientat
cameraView_.fillMode = kGPUImageF
[detector_ addTarget:cameraView_]

// Important: add as a subview
[self.view addSubview:cameraView_

// Setup the face box view
[self setupFaceTrackingViews];
[self calculateTransformations];

// Set the block for running face
[detector_ beginDetecting:kFaceFe
                    codeTypes:@[A
           withDetectionBlock:^(S
               // Check if the kF
               if (detectionType
                   [self updateFa
               }
           }];

// Finally start the camera
```

Smerk and GPUImage

- Recently, an extension to GPUImage was proposed to allow for the utilization of iOS face detection within iOS.
- Called “Smerk” - GitHub project page can be found at:- https://github.com/MattFoley/Smerk




```
// Set the face detector to be us
detector_ = [[SMKDetectionCamera
    AVCaptureDevicePositionFront];
[detector_ setOutputImageOrientat
cameraView_.fillMode = kGPUImageF
[detector_ addTarget:cameraView_]

// Important: add as a subview
[self.view addSubview:cameraView_

// Setup the face box view
[self setupFaceTrackingViews];
[self calculateTransformations];

// Set the block for running face
[detector_ beginDetecting:kFaceFe
                    codeTypes:@[A
           withDetectionBlock:^(S
               // Check if the kF
               if (detectionType
                   [self updateFa
               }
           }];

// Finally start the camera
```

Smerk Example

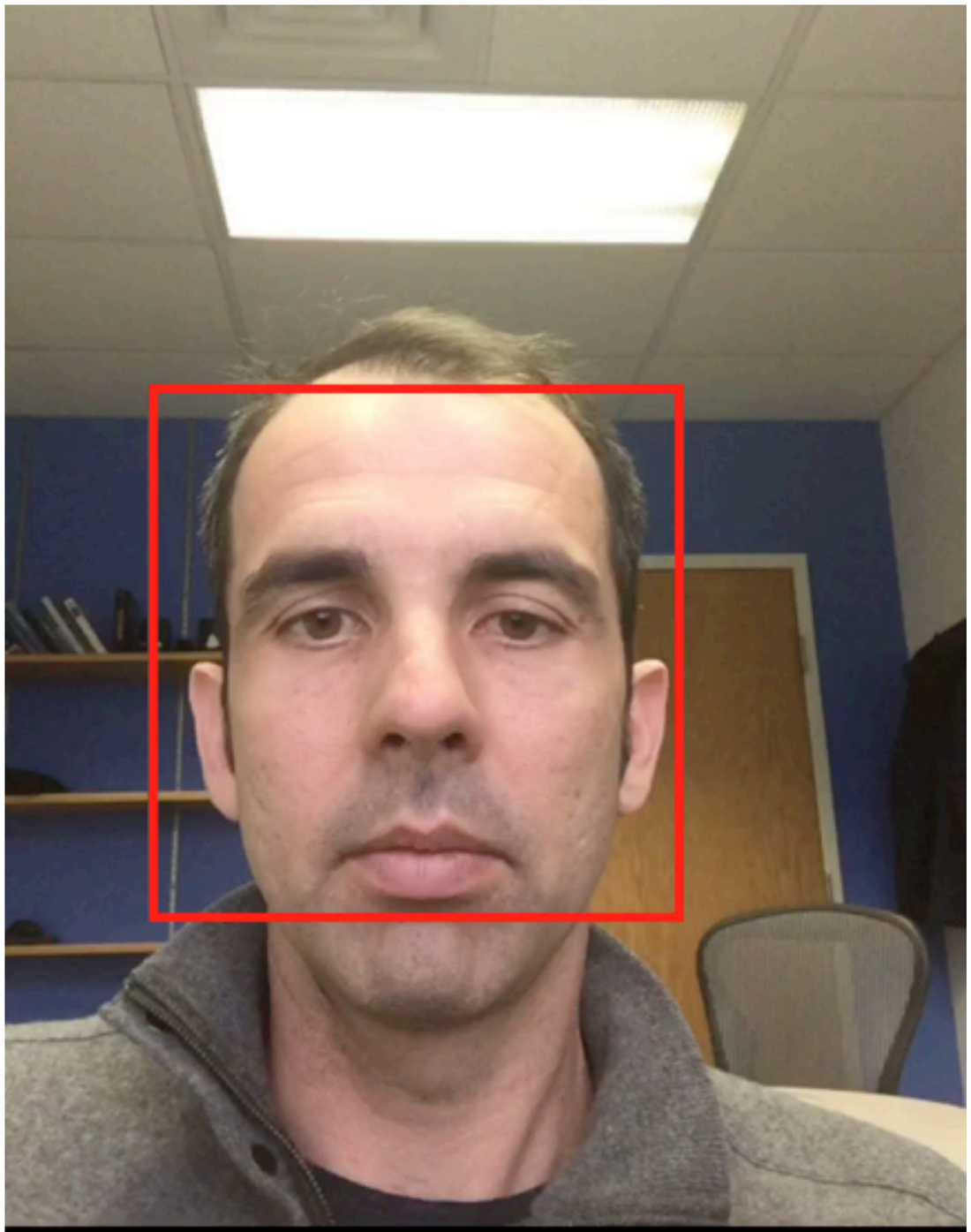
- We are now going to demonstrate how we can perform real-time face tracking through GPUImage.
- On your browser please go to the address,

https://github.com/slucey-cs-cmu-edu/Smerk_Example

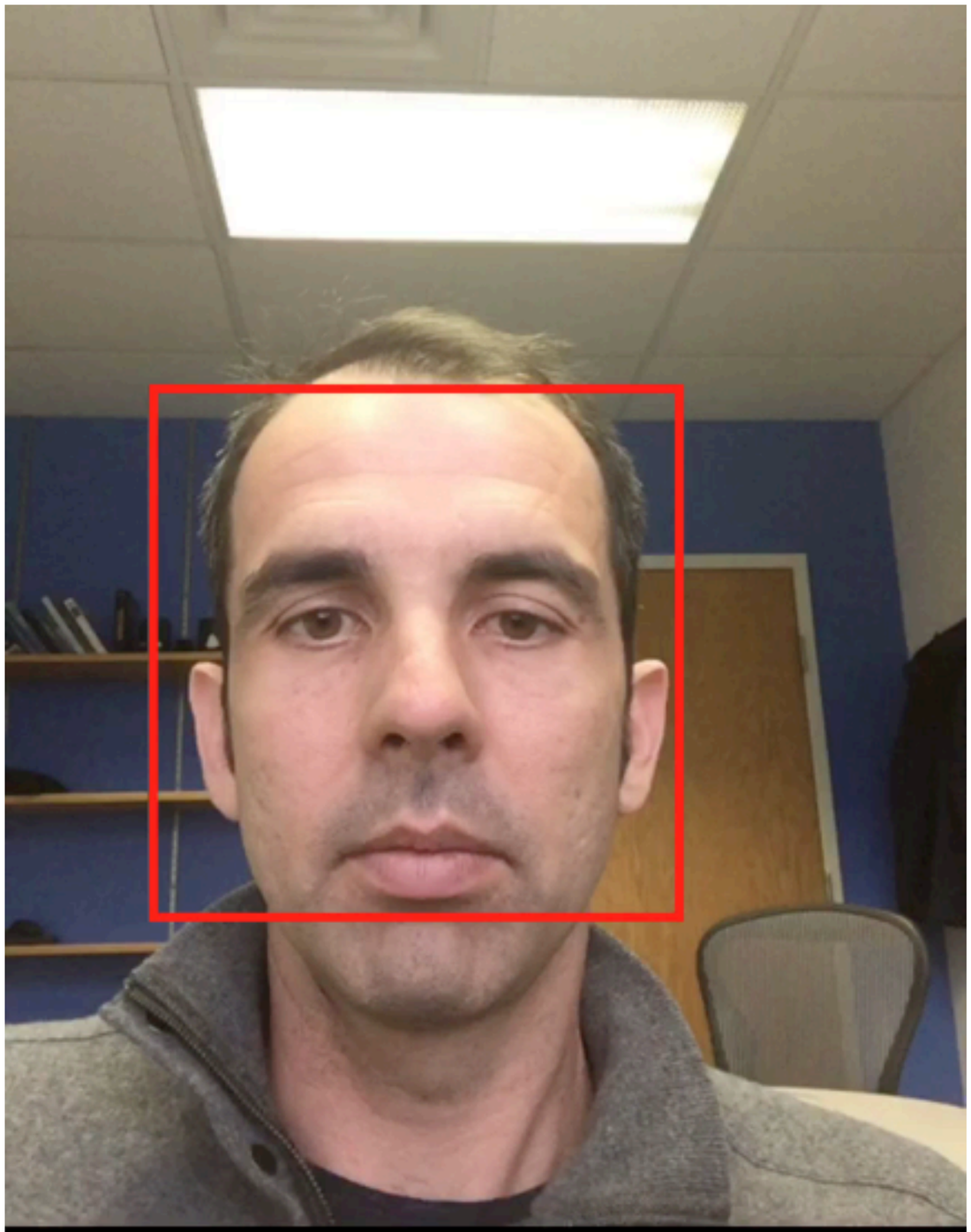
- Or better yet, if you have git installed you can type from the command line.

```
$ git clone https://github.com/slucey-cs-cmu-edu/Smerk_Example.git
```

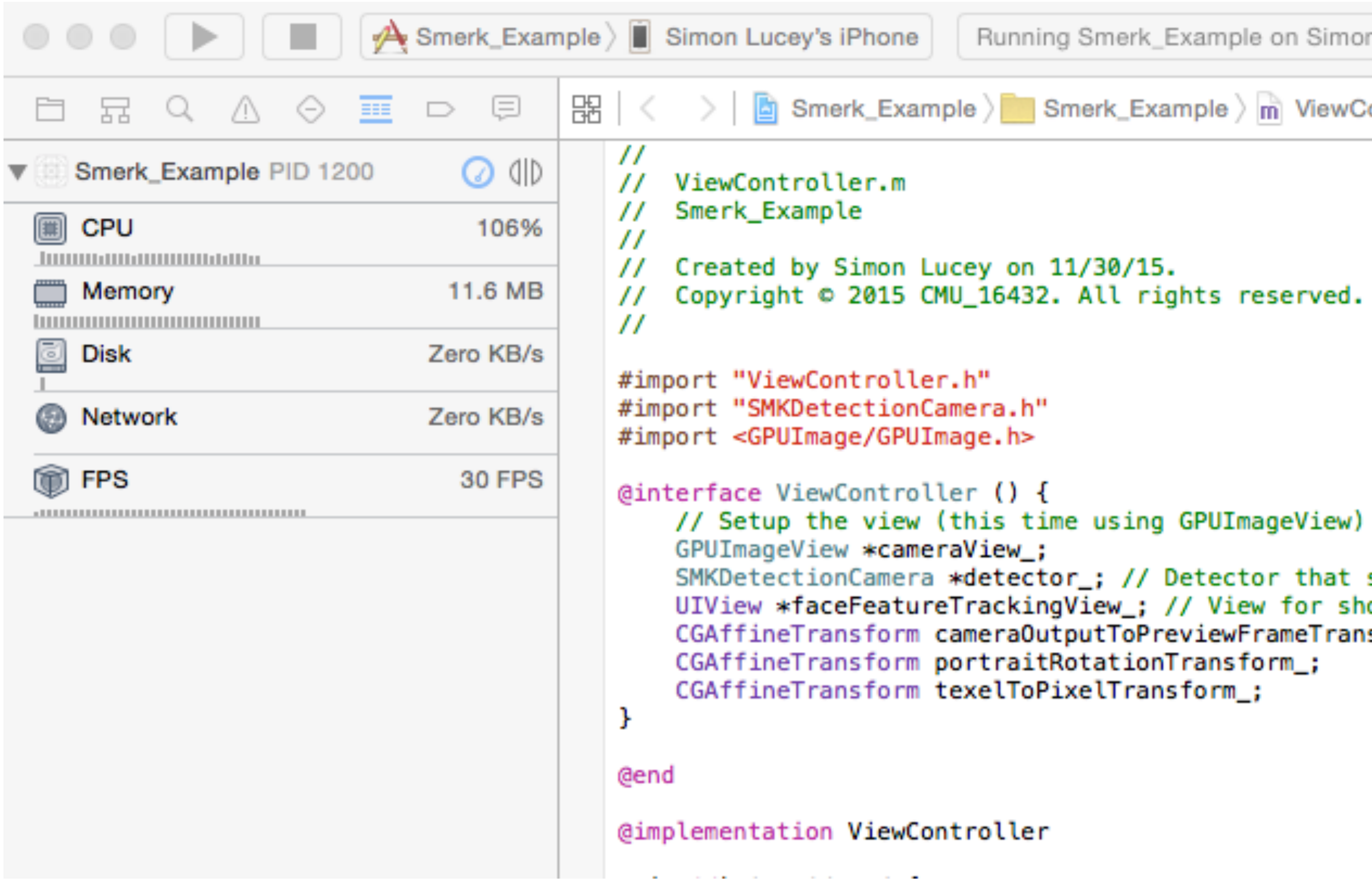
Smerk Example



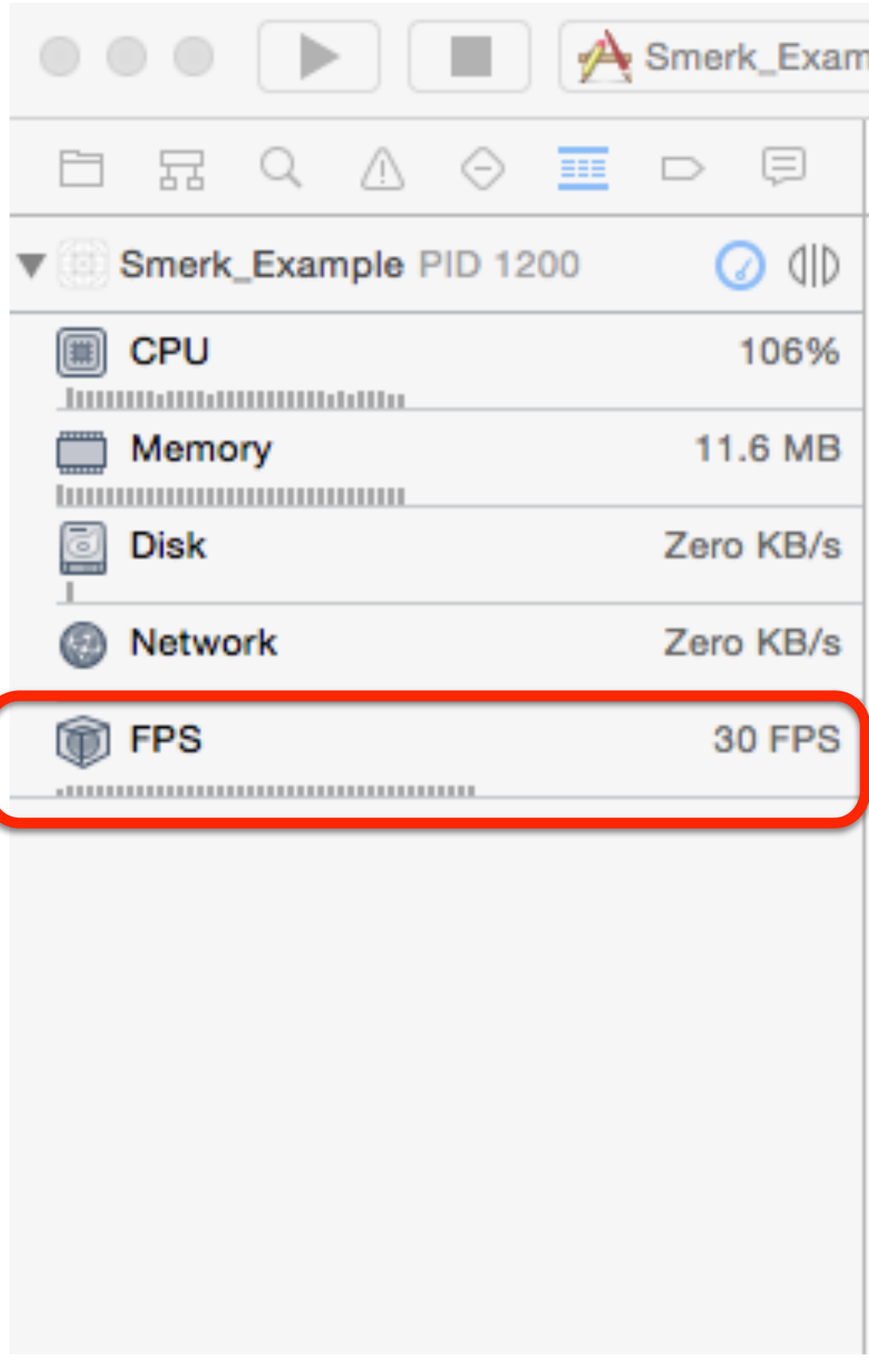
Smerk Example



Smerk Example



Smerk Example



Smerk Example

```
1  //
2  //  ViewController.m
3  //  Smerk_Example
4  //
5  //  Created by Simon Lucey on 11/30/15.
6  //  Copyright © 2015 CMU_16432. All rights reserved.
7  //
8
9  #import "ViewController.h"
10 #import "SMKDetectionCamera.h"
11 #import <GPUImage/GPUImage.h>
12
13 @interface ViewController () {
14     // Setup the view (this time using GPUImageView)
15     GPUImageView *cameraView_;
16     SMKDetectionCamera *detector_; // Detector that should be used
17     UIView *faceFeatureTrackingView_; // View for showing bounding box around the face
18     CGAffineTransform cameraOutputToPreviewFrameTransform_;
19     CGAffineTransform portraitRotationTransform_;
20     CGAffineTransform texelToPixelTransform_;
21 }
22
```

Smerk Example

```
1  //
2  //  ViewController.m
3  //  Smerk_Example
4  //
5  //  Created by Simon Lucey on 11/30/15.
6  //  Copyright © 2015 CMU_16432. All rights reserved.
7  //
8
9  #import "ViewController.h"
10 #import "SMKDetectionCamera.h"
11 #import <GPUImage/GPUImage.h>
12
13 @interface ViewController () {
14     // Setup the view (this time using GPUImageView)
15     GPUImageView *cameraView_;
16     SMKDetectionCamera *detector_; // Detector that should be used
17     UIView *faceFeatureTrackingView_; // View for showing bounding box around the face
18     CGAffineTransform cameraOutputToPreviewFrameTransform_;
19     CGAffineTransform portraitRotationTransform_;
20     CGAffineTransform texelToPixelTransform_;
21 }
22
```

Smerk Example

```
1  //
2  //  ViewController.m
3  //  Smerk_Example
4  //
5  //  Created by Simon Lucey on 11/30/15.
6  //  Copyright © 2015 CMU_16432. All rights reserved.
7  //
8
9  #import "ViewController.h"
10 #import "SMKDetectionCamera.h"
11 #import <GPUImage/GPUImage.h>
12
13 @interface ViewController () {
14     // Setup the view (this time using GPUImageView)
15     GPUImageView *cameraView_;
16     SMKDetectionCamera *detector_; // Detector that should be used
17     UIView *faceFeatureTrackingView_; // View for showing bounding box around the face
18     CGAffineTransform cameraOutputToPreviewFrameTransform_;
19     CGAffineTransform portraitRotationTransform_;
20     CGAffineTransform texelToPixelTransform_;
21 }
22
```

Smerk Example

```
32      // Setup GPUImageView (not we are not using UIImageView here).........
33      cameraView_ = [[GPUImageView alloc] initWithFrame:CGRectMake(0.0, 0.0, self.view.frame.size.width,
34
35      // Set the face detector to be used
36      detector_ = [[SMKDetectionCamera alloc] initWithSessionPreset:AVCaptureSessionPreset640x480 cameraP
37      [detector_ setOutputImageOrientation:UIInterfaceOrientationPortrait]; // Set to portrait
38      cameraView_.fillMode = kGPUImageFillModePreserveAspectRatio;
39      [detector_ addTarget:cameraView_];
40
41      // Important: add as a subview
42      [self.view addSubview:cameraView_];
43
44      // Setup the face box view
45      [self setupFaceTrackingViews];
46      [self calculateTransformations];
47
48      // Set the block for running face detector
49      [detector_ beginDetecting:kFaceFeatures | kMachineAndFaceMetaData
50                      codeTypes:@[AVMetadataObjectTypeQRCode]
51                withDetectionBlock:^(SMKDetectionOptions detectionType, NSArray *detectedObjects, CGRect
52                    // Check if the kFaceFeatures have been discovered
53                    if (detectionType & kFaceFeatures) {
54                        [self updateFaceFeatureTrackingViewWithObjects:detectedObjects];
55                    }
56                }];
57
58      // Finally start the camera
59      [detector_ startCameraCapture];
```

Smerk Example

```
32      // Setup GPUImageView (not we are not using UIImageView here).........
33      cameraView_ = [[GPUImageView alloc] initWithFrame:CGRectMake(0.0, 0.0, self.view.frame.size.width,
34
35      // Set the face detector to be used
36      detector_ = [[SMKDetectionCamera alloc] initWithSessionPreset:AVCaptureSessionPreset640x480 cameraP
37      [detector_ setOutputImageOrientation:UIInterfaceOrientationPortrait]; // Set to portrait
38      cameraView_.fillMode = kGPUImageFillModePreserveAspectRatio;
39      [detector_ addTarget:cameraView_];
40
41      // Important: add as a subview
42      [self.view addSubview:cameraView_];
43
44      // Setup the face box view
45      [self setupFaceTrackingViews];
46      [self calculateTransformations];
47
48      // Set the block for running face detector
49      [detector_ beginDetecting:kFaceFeatures | kMachineAndFaceMetaData
50                      codeTypes:@[AVMetadataObjectTypeQRCode]
51                withDetectionBlock:^(SMKDetectionOptions detectionType, NSArray *detectedObjects, CGRect
52                    // Check if the kFaceFeatures have been discovered
53                    if (detectionType & kFaceFeatures) {
54                        [self updateFaceFeatureTrackingViewWithObjects:detectedObjects];
55                    }
56                }];
57
58      // Finally start the camera
59      [detector_ startCameraCapture];
```

Smerk Example

```
32      // Setup GPUImageView (not we are not using UIImageView here).........
33      cameraView_ = [[GPUImageView alloc] initWithFrame:CGRectMake(0.0, 0.0, self.view.frame.size.width,
34
35      // Set the face detector to be used
36      detector_ = [[SMKDetectionCamera alloc] initWithSessionPreset:AVCaptureSessionPreset640x480 cameraP
37      [detector_ setOutputImageOrientation:UIInterfaceOrientationPortrait]; // Set to portrait
38      cameraView_.fillMode = kGPUImageFillModePreserveAspectRatio;
39      [detector_ addTarget:cameraView_];
40
41      // Important: add as a subview
42      [self.view addSubview:cameraView_];
43
44      // Setup the face box view
45      [self setupFaceTrackingViews];
46      [self calculateTransformations];
47
48      // Set the block for running face detector
49      [detector_ beginDetecting:kFaceFeatures | kMachineAndFaceMetaData
50                      codeTypes:@[AVMetadataObjectTypeQRCode]
51                withDetectionBlock:^(SMKDetectionOptions detectionType, NSArray *detectedObjects, CGRect
52                    // Check if the kFaceFeatures have been discovered
53                    if (detectionType & kFaceFeatures) {
54                        [self updateFaceFeatureTrackingViewWithObjects:detectedObjects];
55                    }
56                }];
57
58      // Finally start the camera
59      [detector_ startCameraCapture];
```

Today

- Facial Feature Detection in iOS.
- Active Appearance Models.

Rigid Warps

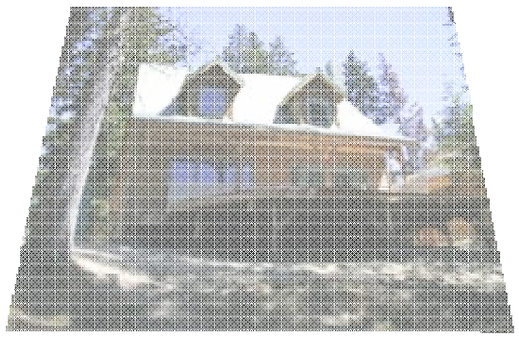
- Up until now, we have been limited to dealing with objects that lie conveniently within a rectangle.
- These objects can only deform rigidly,

translation

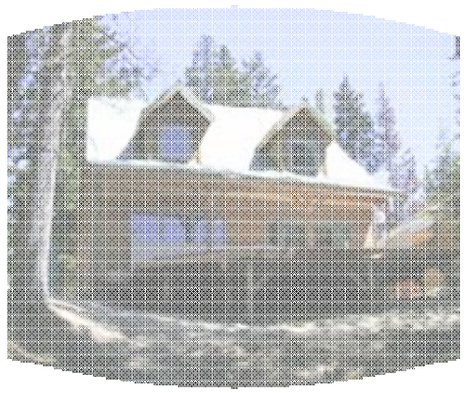
rotation

aspect

affine



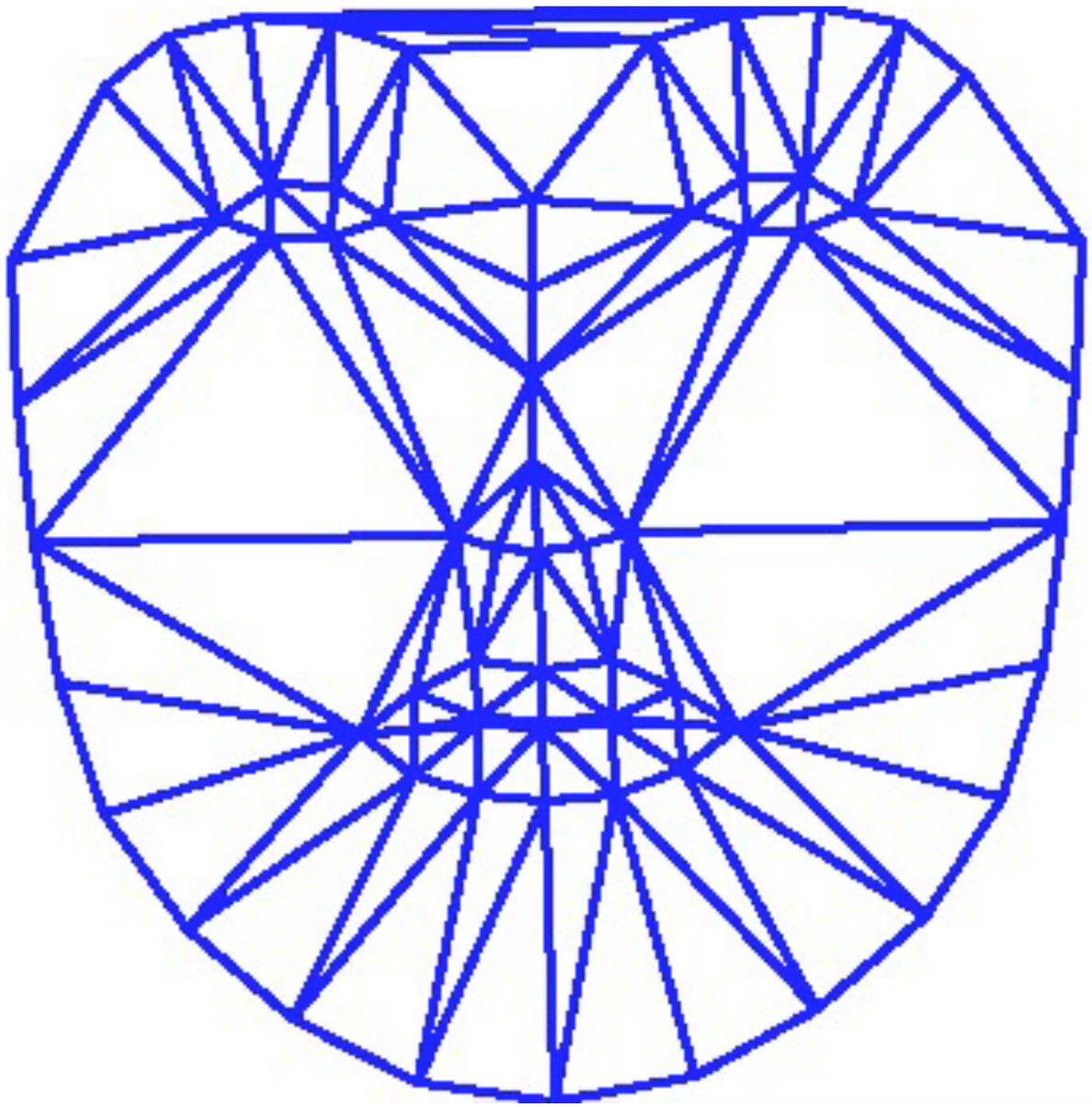
perspective



cylindrical

Non-Rigid Warps

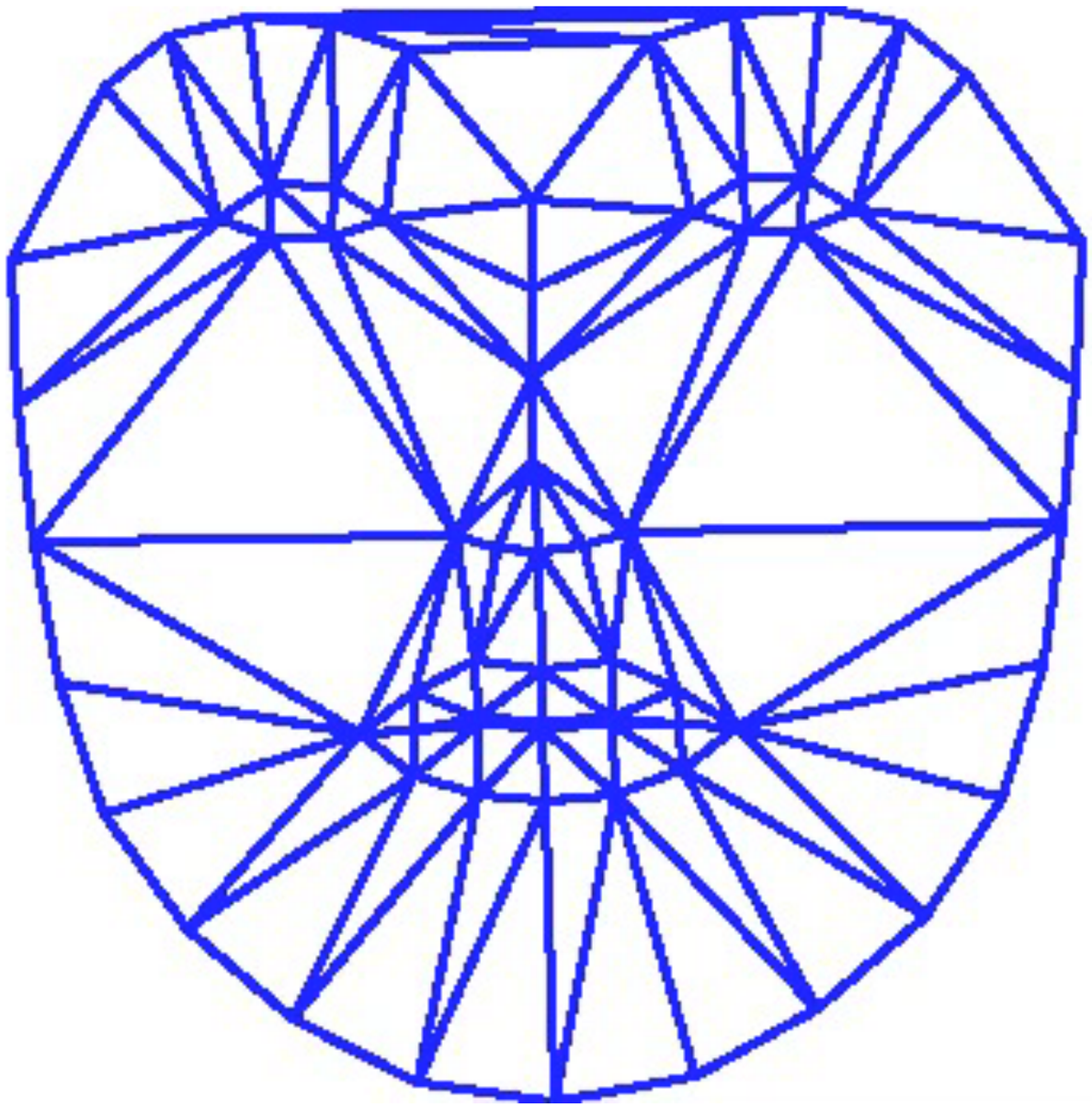
- What happens if I want to deal with non-rigid warps,



(Szeliski and Fleet)

Non-Rigid Warps

- What happens if I want to deal with non-rigid warps,



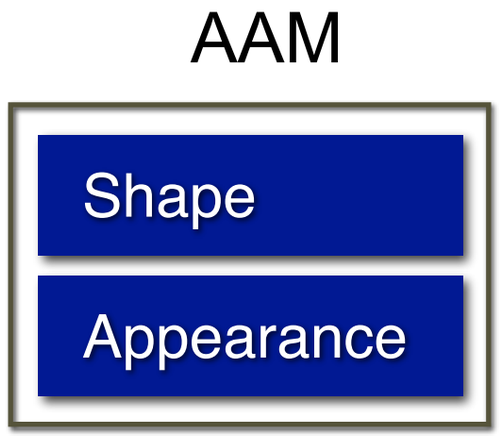
(Szeliski and Fleet)

Active Appearance Models

- Introduced by Cootes and Taylor in 1998
- Example of class of deformable models which also includes Active Blobs (Sclaroff and Isidoror), Active Shape Models (Cootes and Taylor) and Morphable Models (Blanz and Vetter)
- Used successfully on faces, hands, and in medical imaging (Matthews and Gross)

Active Appearance Models

- Separately model object shape and appearance



(Matthews and Gross)

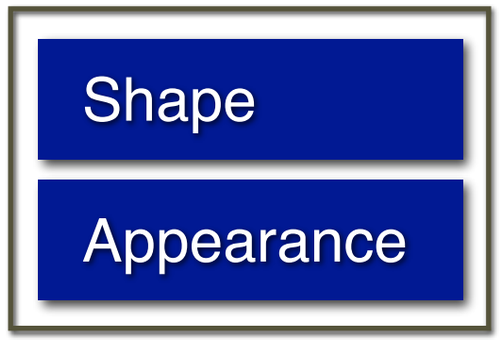
Active Appearance Models

- Separately model object shape and appearance

Parameters

0.12
-0.34
6.78
-12.2
0.01

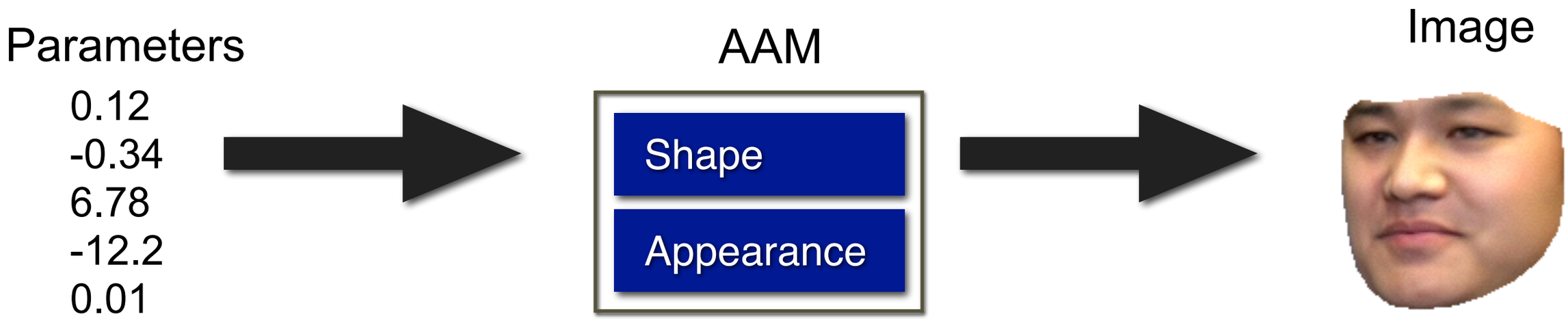
AAM



(Matthews and Gross)

Active Appearance Models

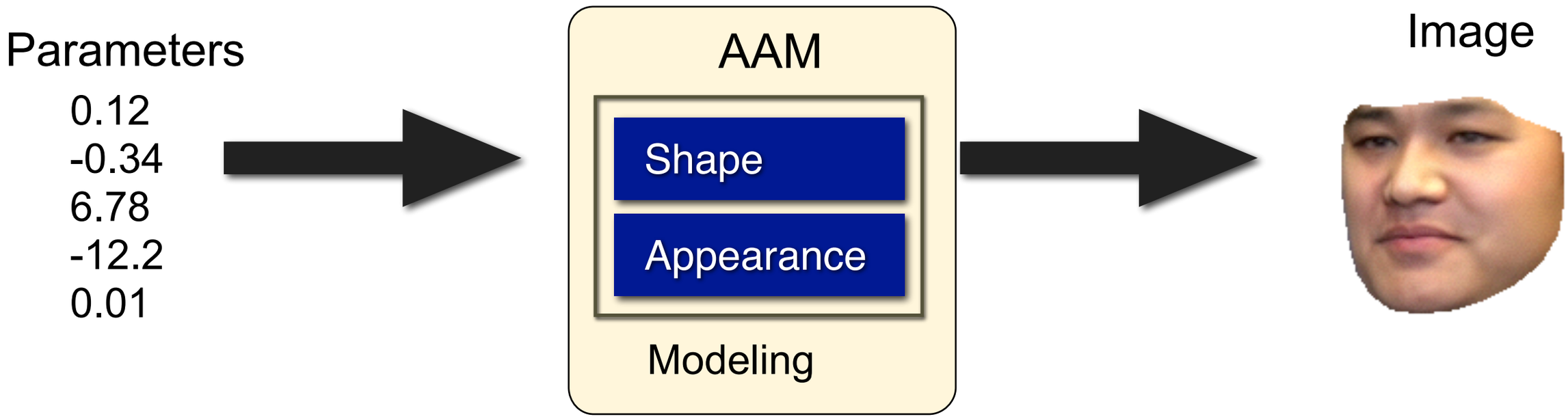
- Separately model object shape and appearance



(Matthews and Gross)

Active Appearance Models

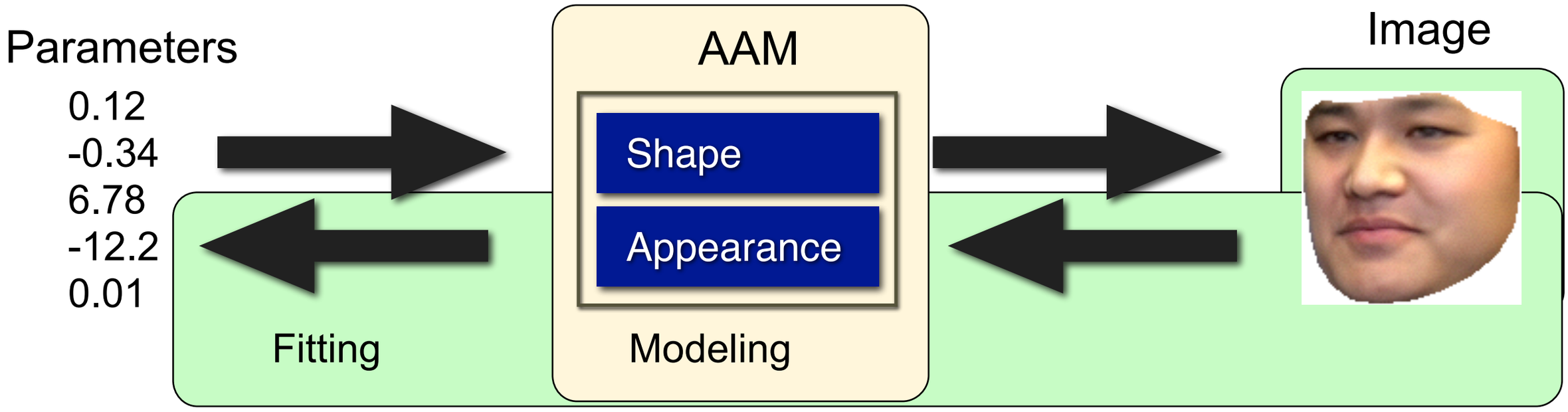
- Separately model object shape and appearance



(Matthews and Gross)

Active Appearance Models

- Separately model object shape and appearance

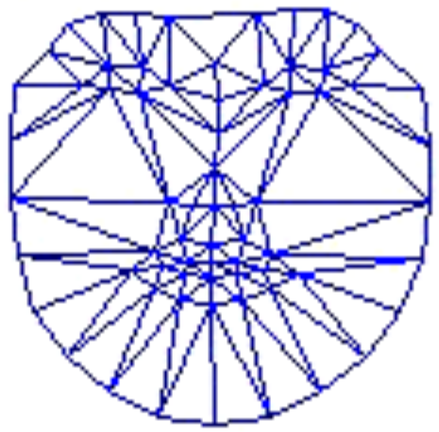


(Matthews and Gross)

Active Appearance Models

Appearance

Shape



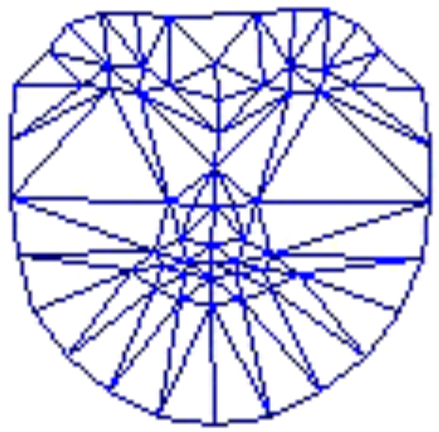
Combined

(Matthews and Gross)

Active Appearance Models

Appearance

Shape



Combined

(Matthews and Gross)

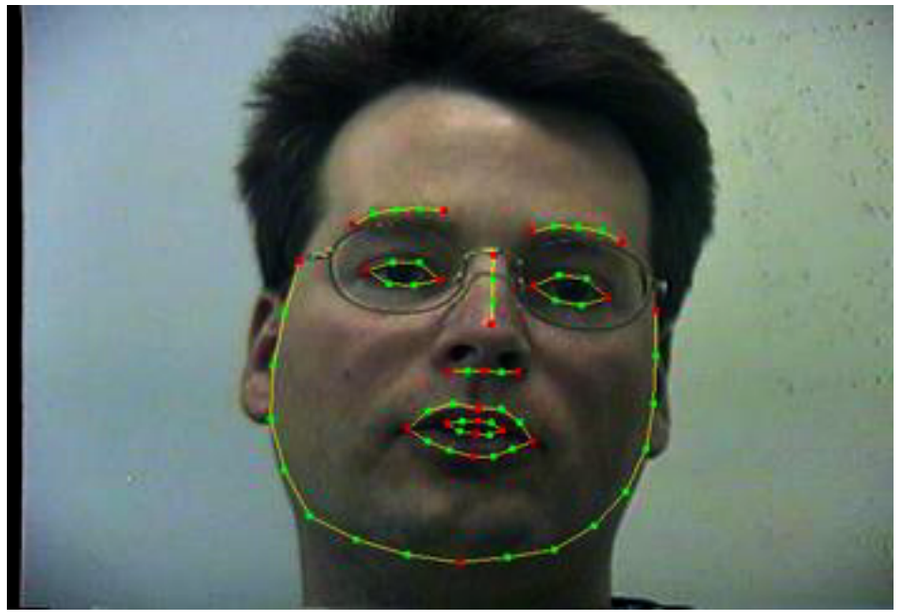
AAM Overview

- How do you build an AAM?
- How do you fit an AAM to an image?
  - What is the general approach?
  - How can we do it **efficiently**?
- Extensions

(Matthews and Gross)

Model Building - Definition

- Define a set of landmarks on a face that can be reliably located in an image

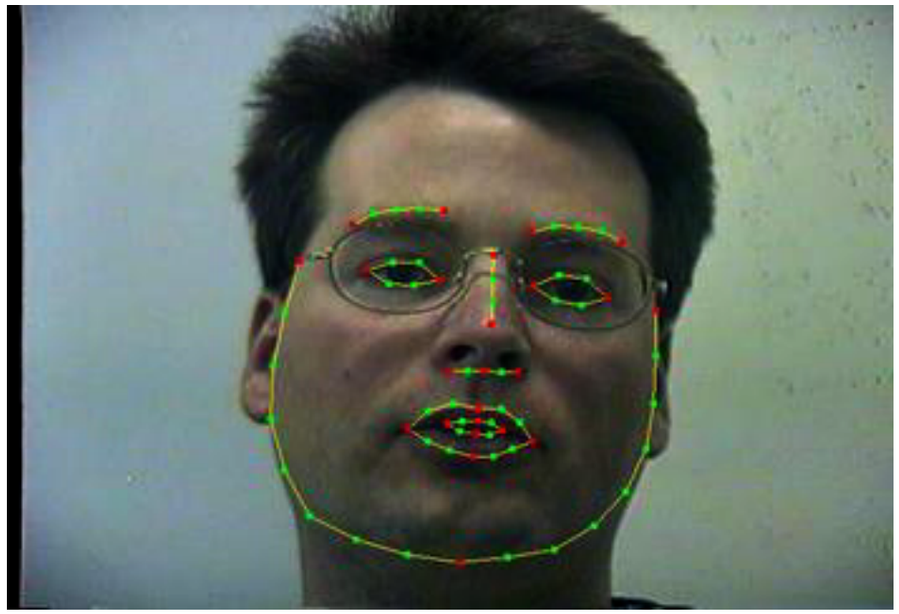


- Still open questions:
  - How many landmarks, which landmarks

(Matthews and Gross)

Model Building - Definition

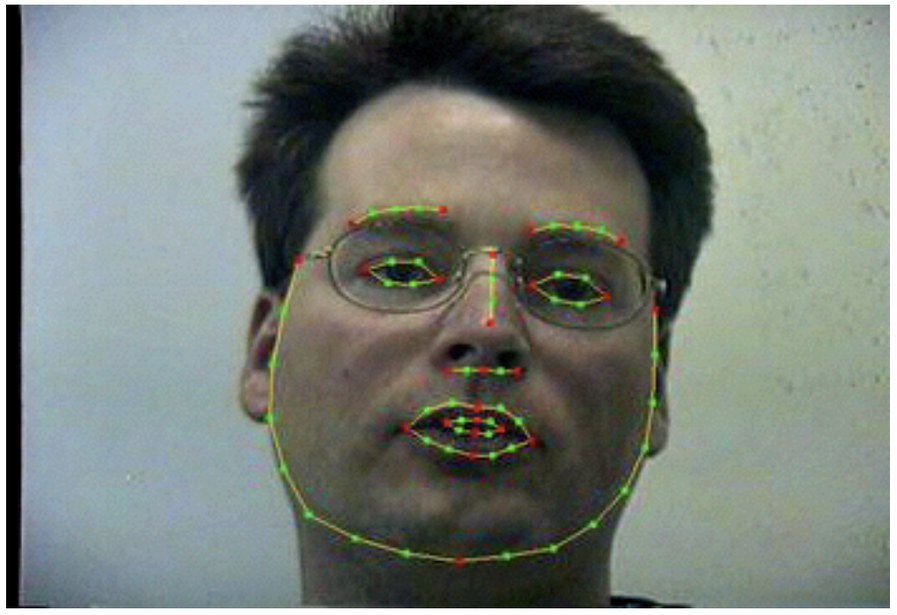
- Define a set of landmarks on a face that can be reliably located in an image



- Still open questions:
  - How many landmarks, which landmarks

(Matthews and Gross)

Model Building - Labeling



$$(x_1, y_1, x_2, y_2, \ldots, x_n, y_n)$$

Manually label lots and lots of data

(Matthews and Gross)

Model Building - Shape

- Various sources of shape variations between individual images
  - Identity
  - Facial expression
  - Position in the image

(Matthews and Gross)

Model Building - Shape

- Various sources of shape variations between individual images
  - Identity
  - Facial expression
  - Position in the image

Don't Care!

(Matthews and Gross)

Shape Alignment

- Procrustes analysis
  - Removes variation due to predetermined shape transformation

(Matthews and Gross)

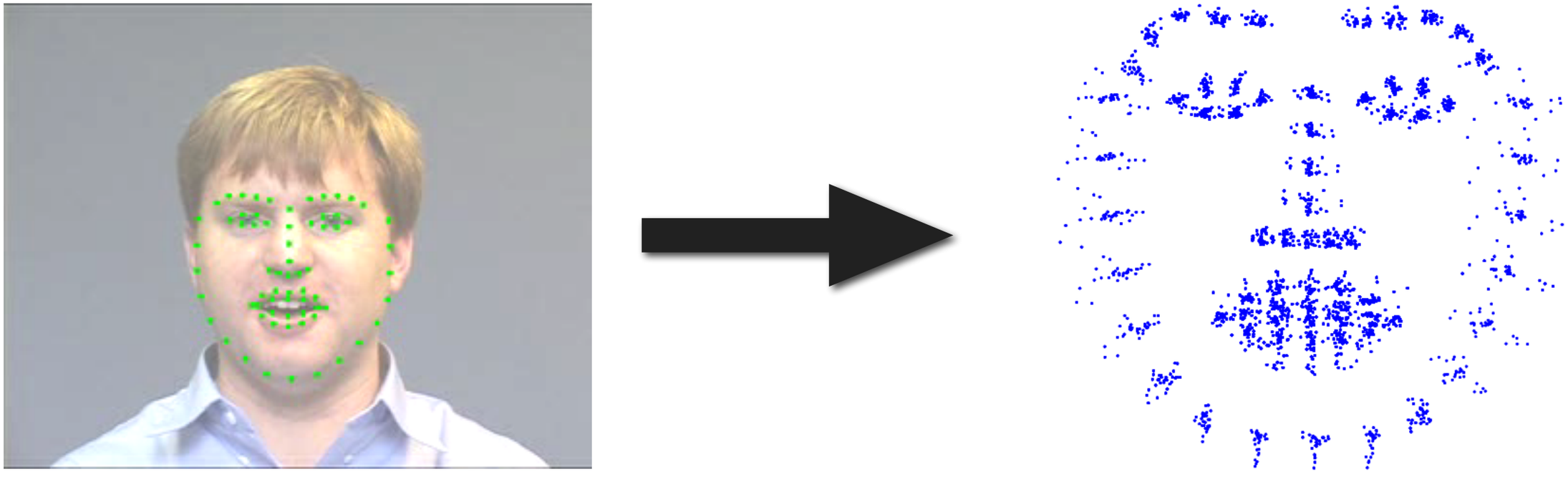
Shape Alignment

- Procrustes analysis
  - Removes variation due to predetermined shape transformation

(Matthews and Gross)

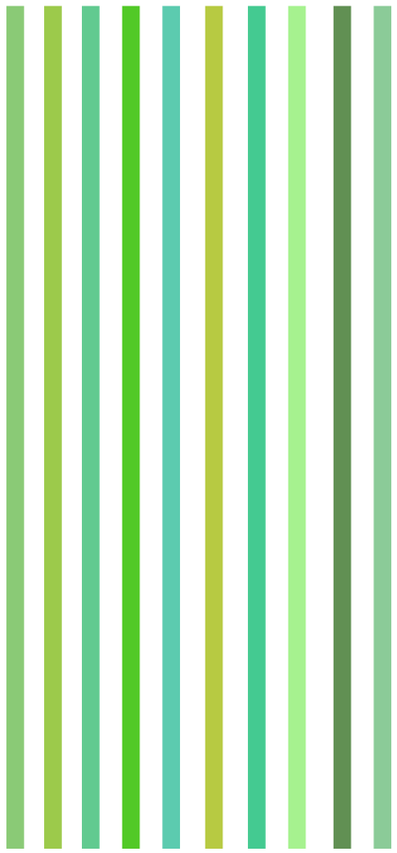
Shape Alignment

- Procrustes analysis
  - Removes variation due to predetermined shape transformation

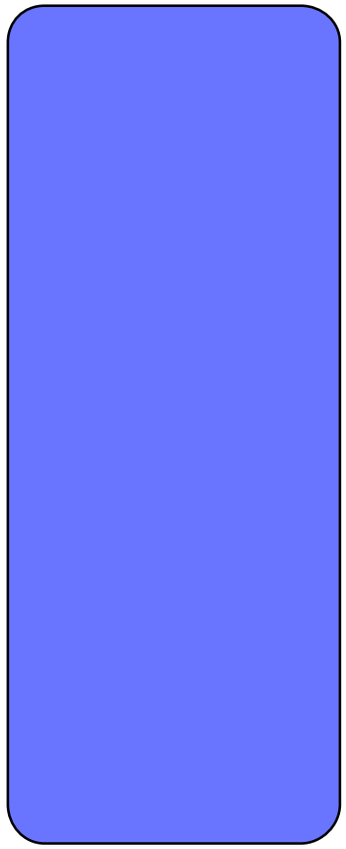


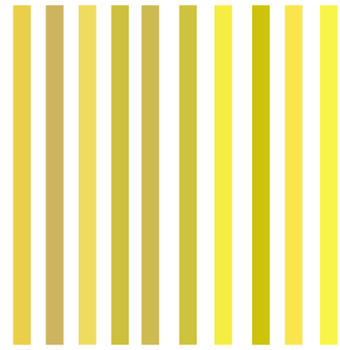
(Matthews and Gross)

Building a Shape Model



=





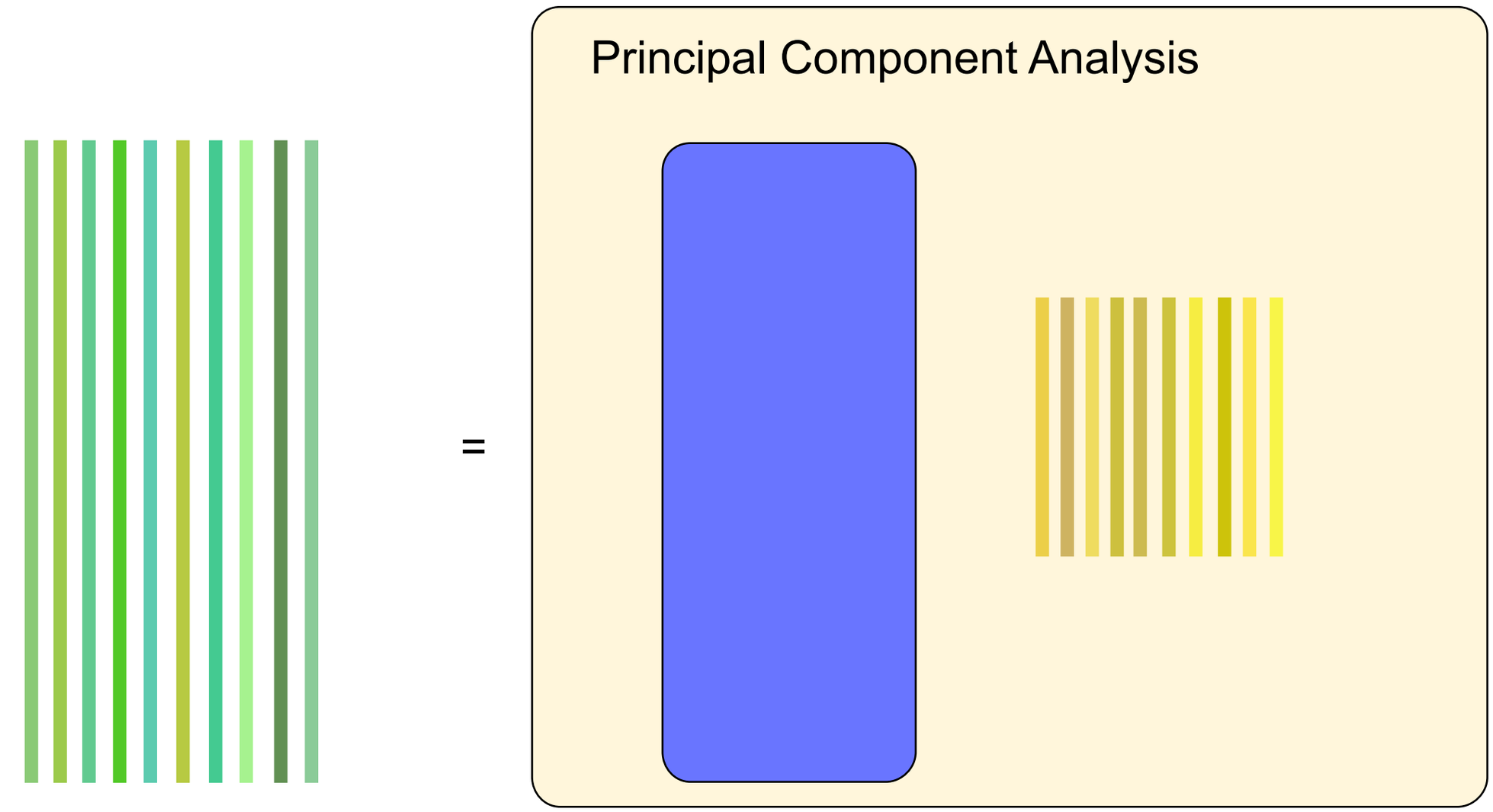
Input Shape

Basis

Parameters

(Matthews and Gross)

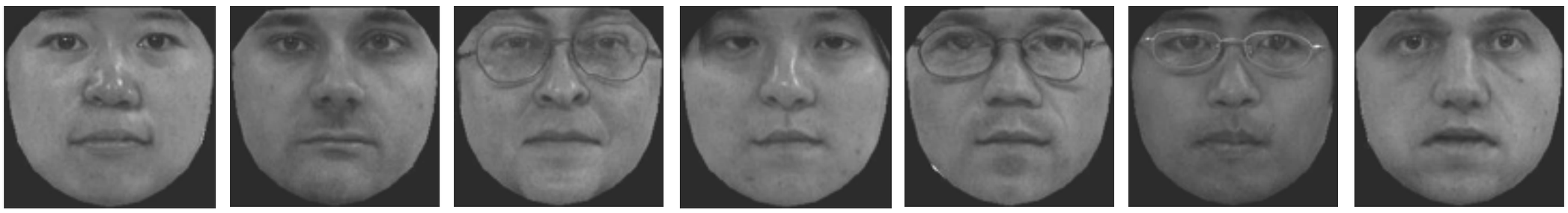
Building a Shape Model



(Matthews and Gross)

PCA on Appearance

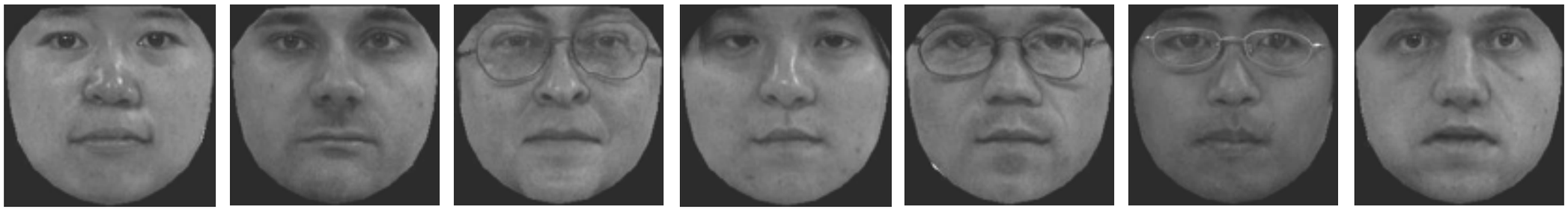
Original



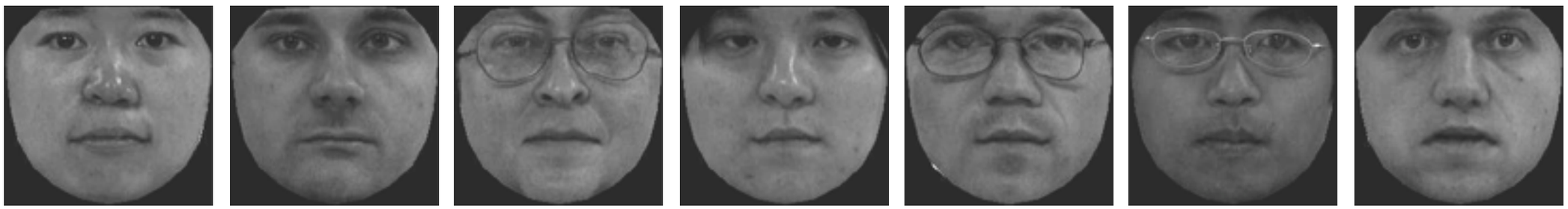
(Matthews and Gross)

PCA on Appearance

Original



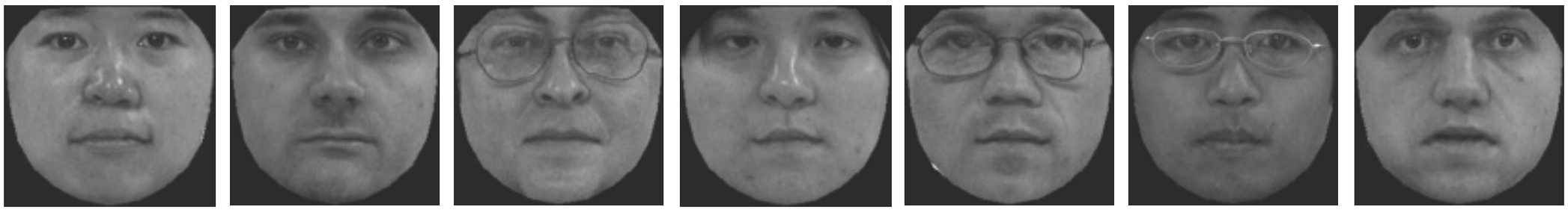
Full



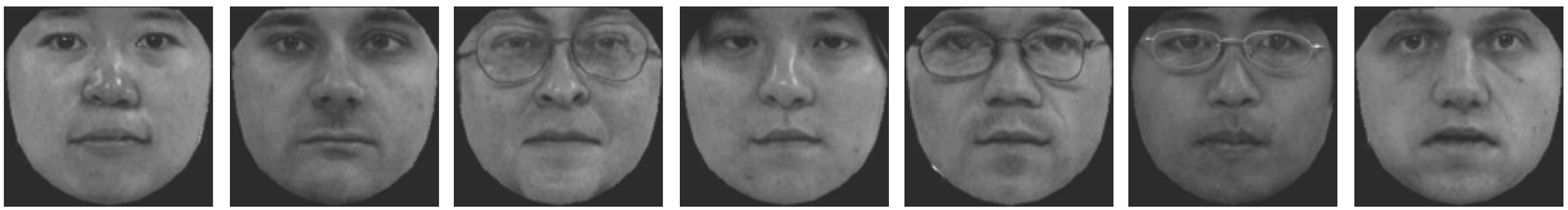
(Matthews and Gross)

PCA on Appearance

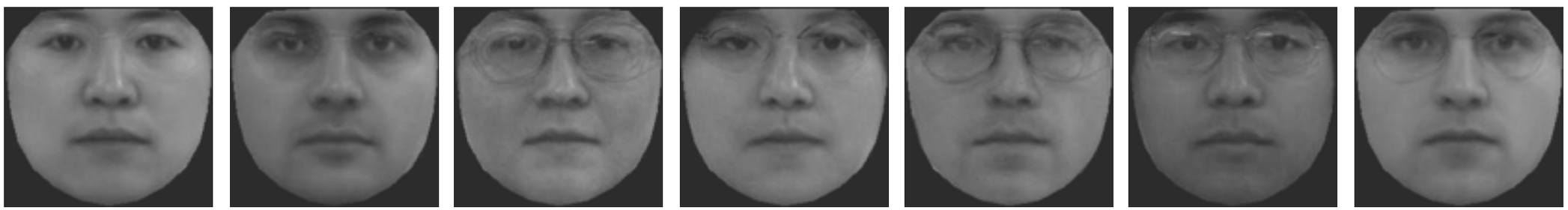
Original



Full



20 dim.



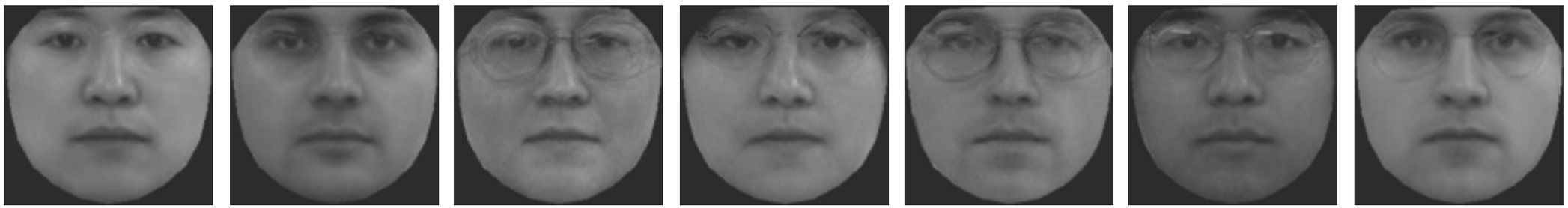
(Matthews and Gross)

PCA on Appearance

Original

Full

20 dim.

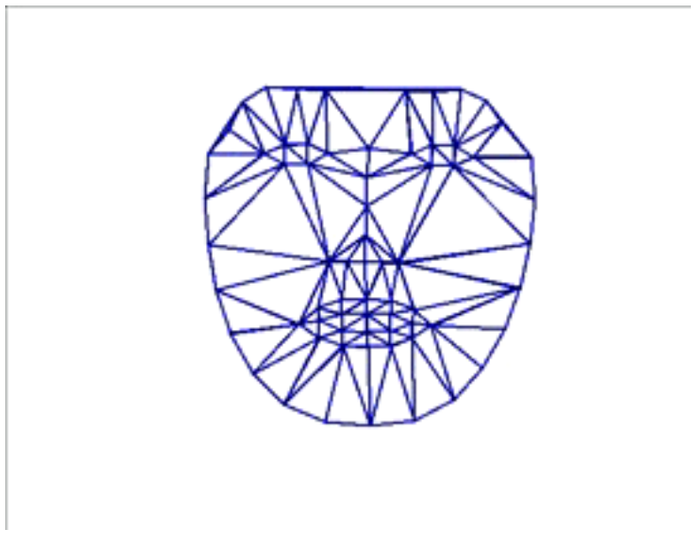


10 dim.

(Matthews and Gross)

PCA Shape Model

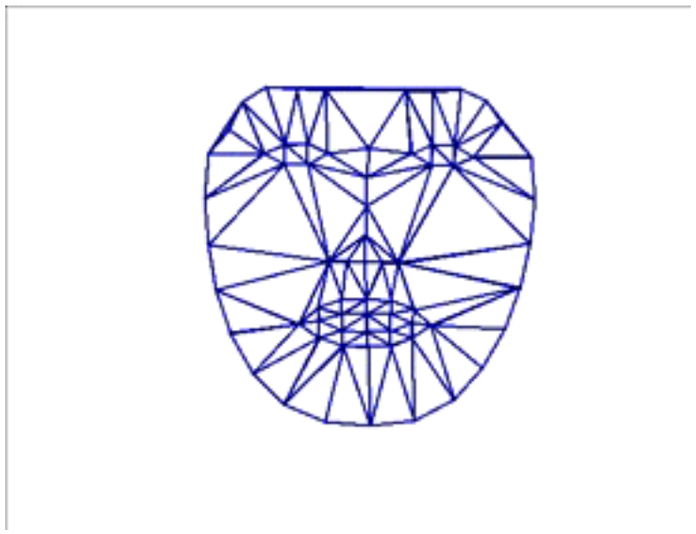
- Input:
  Bunch of aligned face shapes
- Output
  Mean shape and the direction of the largest variations (eigenvectors)



(Matthews and Gross)

PCA Shape Model

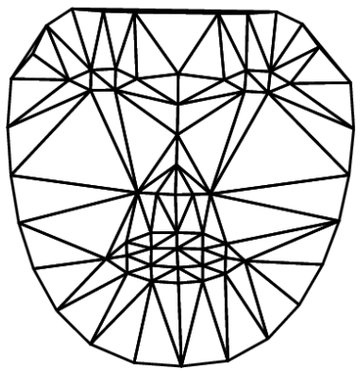
- Input:
  Bunch of aligned face shapes
- Output
  Mean shape and the direction of the largest variations (eigenvectors)



(Matthews and Gross)

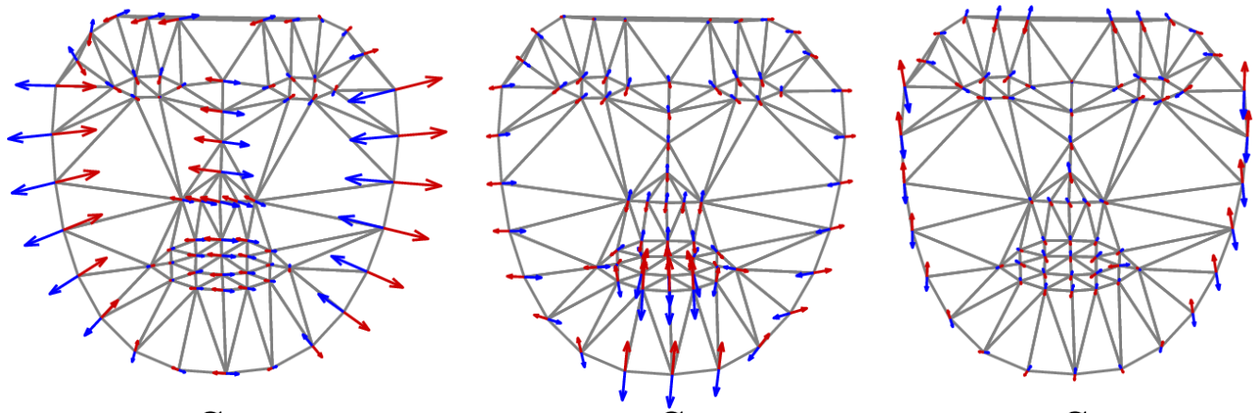
PCA Shape Model

Mean Face



$\mathbf{s}_0$

First 3 Shape Modes



$\mathbf{s}_1$ $\mathbf{s}_2$ $\mathbf{s}_3$

Shape Model

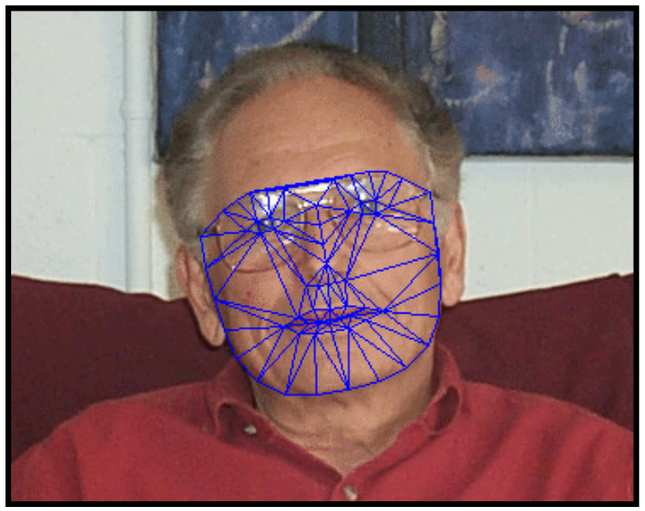
$$\mathcal{W}(\mathbf{z};\mathbf{p}) = \mathbf{s}_0 + \sum_{k=1}^{K} p_k \mathbf{s}_k$$

Shape Parameters

$$\mathbf{p} = [p_1, p_2, \ldots, p_K]^T$$

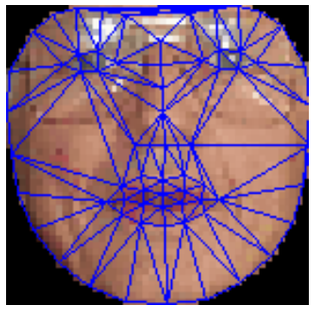
(Matthews and Gross)

Construction of Appearance Model



Warp to mean shape

$\mathbf{s}_0$



PCA on Appearance

(Matthews and Gross)

PCA Appearance Model

- Input:
  Bunch of shape normalized textures
- Output:
  Mean appearance and appearance eigenvectors

(Matthews and Gross)

PCA Appearance Model

- Input:
  Bunch of shape normalized textures
- Output:
  Mean appearance and appearance eigenvectors

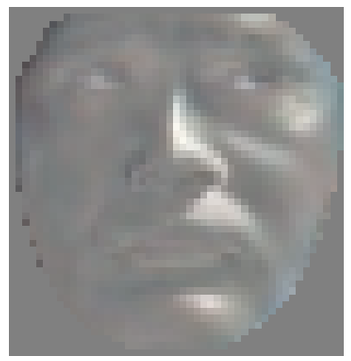
(Matthews and Gross)

AAM Appearance Model

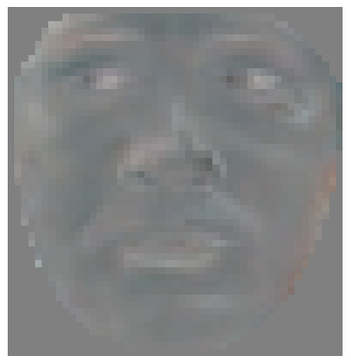
Mean Face

$A_0(\mathbf{z})$

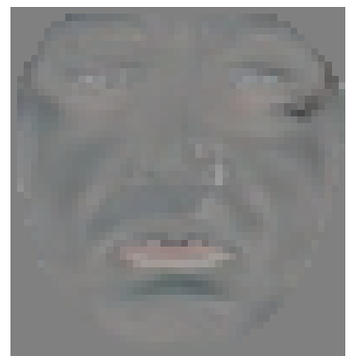
First 3 Appearance Modes



$A_1(\mathbf{z})$



$A_2(\mathbf{z})$



$A_3(\mathbf{z})$

Appearance Model

$$T(\mathbf{z}) = A_0(\mathbf{z}) + \sum_{m=1}^{M} \lambda_i A_m(\mathbf{z})$$

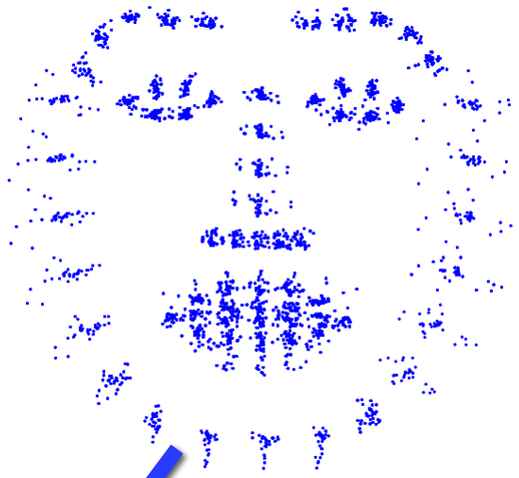
Appearance Parameters

$$(\lambda_1, \lambda_2, \ldots, \lambda_n)^T$$

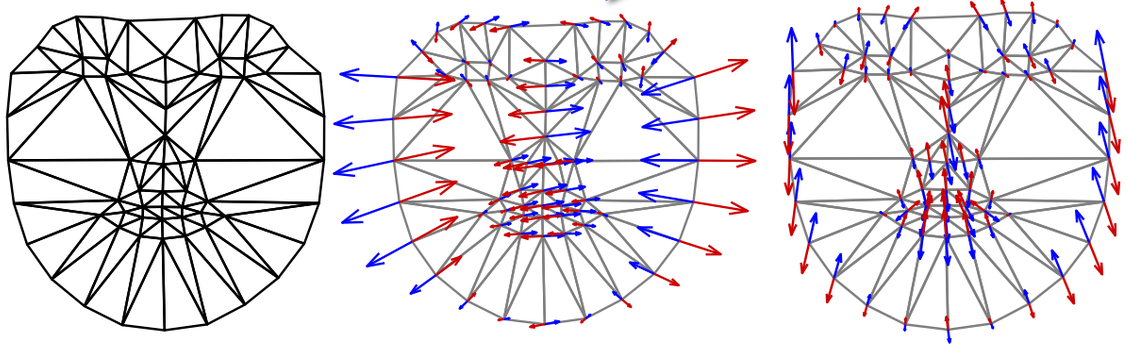
(Matthews and Gross)

AAM Model Building Overview

Procrustes



PCA



s_0 s_1 s_2

Shape Modes

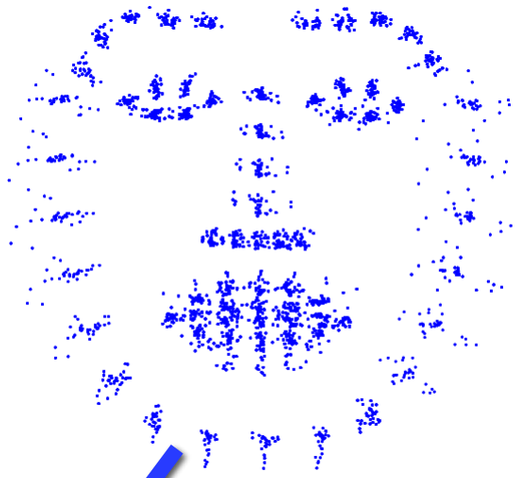
PCA

A_0 A_1 A_2

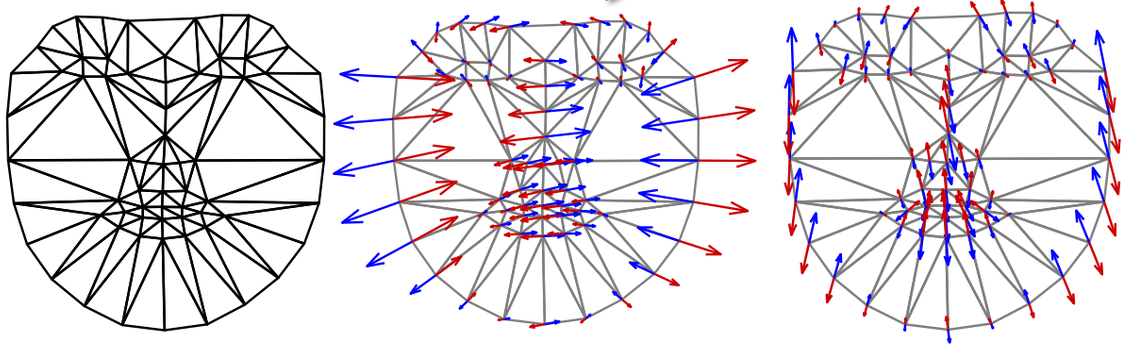
Appearance Modes

AAM Model Building Overview

Procrustes



PCA



s_0 s_1 s_2

Shape Modes

PCA

A_0 A_1 A_2

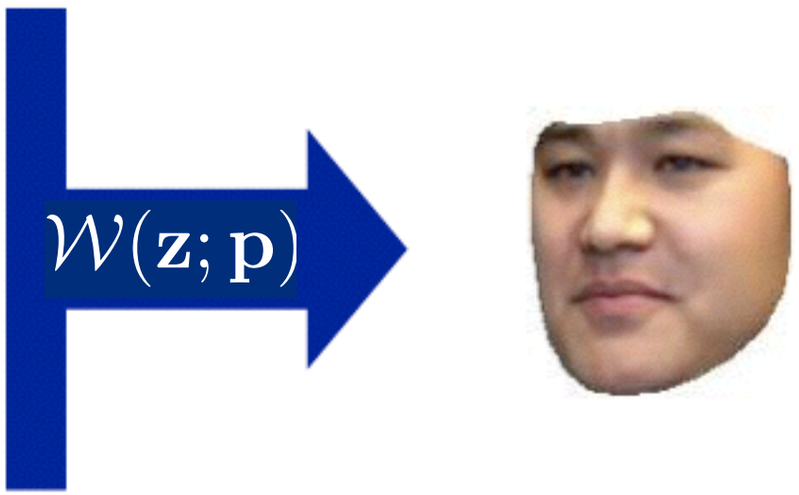
Appearance Modes

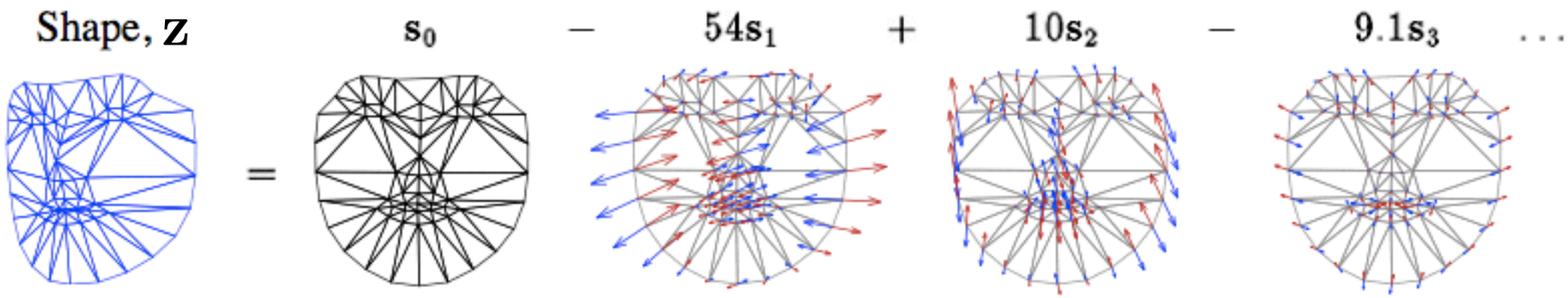
Model Instantiation

Appearance, $T(\mathbf{z})$ $= A_0(\mathbf{z}) + 3559A_1(\mathbf{z}) + 351A_2(\mathbf{z}) - 256A_3(\mathbf{z}) \ldots$

(Matthews and Gross)

Model Instantiation

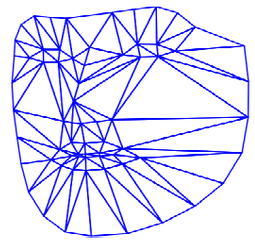




(Matthews and Gross)

Model Instantiation

$$T(\mathbf{z}) = A_0(\mathbf{z}) + \sum_{m=1}^{M} \lambda_i A_m(\mathbf{z}) \qquad \mathcal{W}(\mathbf{z};\mathbf{p}) = \mathbf{s}_0 + \sum_{k=1}^{K} p_k \mathbf{s}_k$$



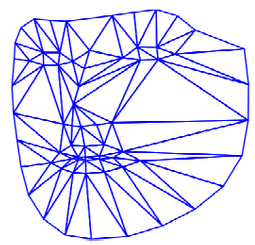
Warp

$T(\mathcal{W}(\mathbf{z};\mathbf{p}))$

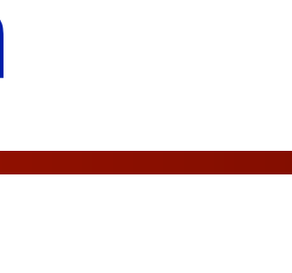
(Matthews and Gross)

Model Instantiation

$$T(\mathbf{z}) = A_0(\mathbf{z}) + \sum_{m=1}^{M} \lambda_i A_m(\mathbf{z}) \qquad \mathcal{W}(\mathbf{z};\mathbf{p}) = \mathbf{s}_0 + \sum_{k=1}^{K} p_k \mathbf{s}_k$$



Warp



$T(\mathcal{W}(\mathbf{z};\mathbf{p}))$

2D Similarity Transform

Source Image

(Matthews and Gross)

A Parameterized Face

(Matthews and Gross)

A Parameterized Face

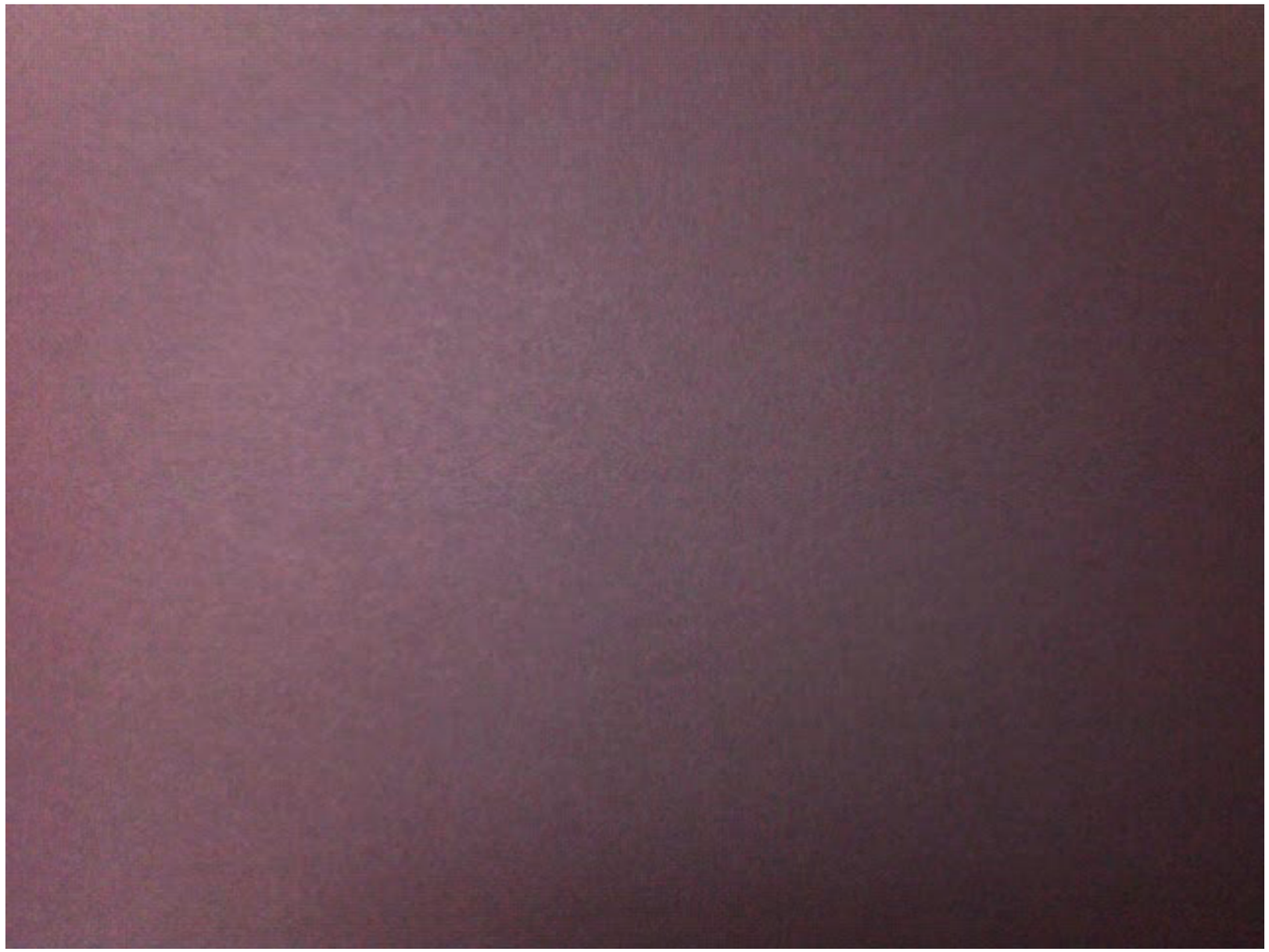
(Matthews and Gross)

Things to Note

- AAMs are **linear** models
  - Efficiently capture non-rigid face deformations
- AAMs are entirely **data-driven**
  - No “face constraints”
- We discussed independent AAMs. “Combined” AAMs add an additional PCA on the combined shape and appearance parameters.

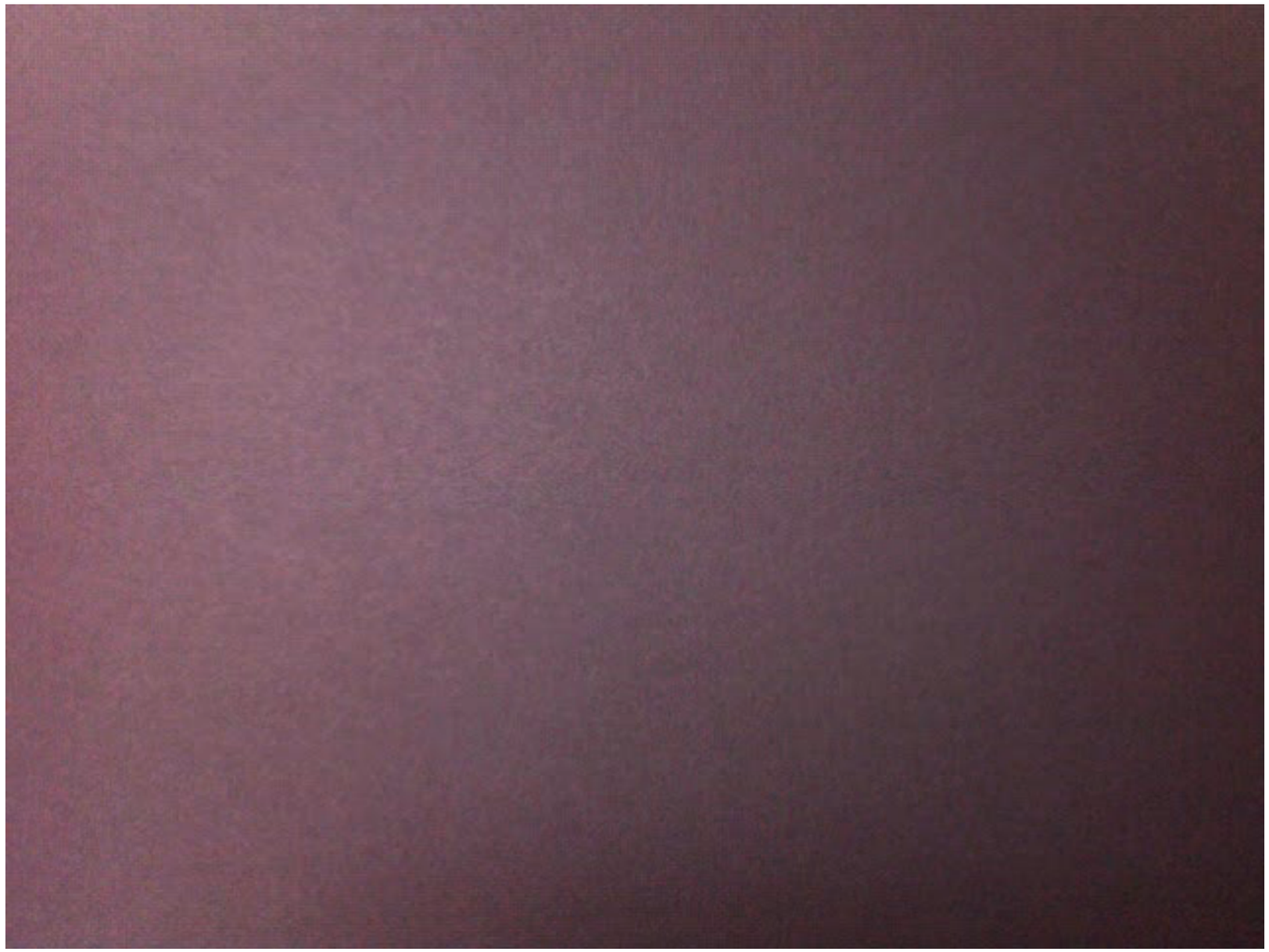
(Matthews and Gross)

Hand AAM



(Matthews and Gross)

Hand AAM



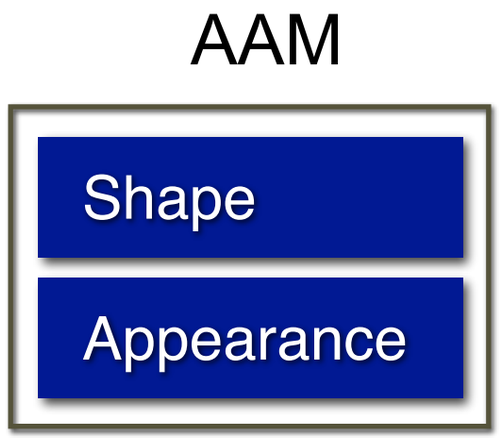
(Matthews and Gross)

Things to Note

- AAMs are **linear** models
  - Efficiently capture non-rigid face deformations
- AAMs are entirely **data-driven**
  - No “face constraints”
- We discussed independent AAMs. “Combined” AAMs add an additional PCA on the combined shape and appearance parameters.

(Matthews and Gross)

Active Appearance Models



Determine the best model parameters to reconstruct the image

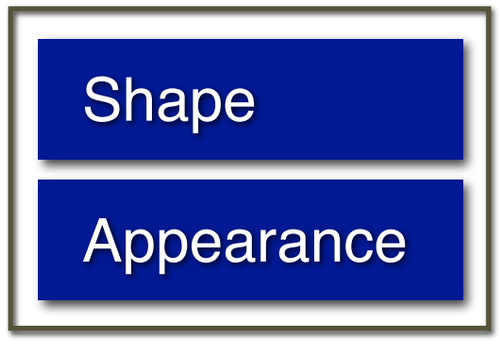
(Matthews and Gross)

Active Appearance Models

Parameters

0.12
-0.34
6.78
-12.2
0.01

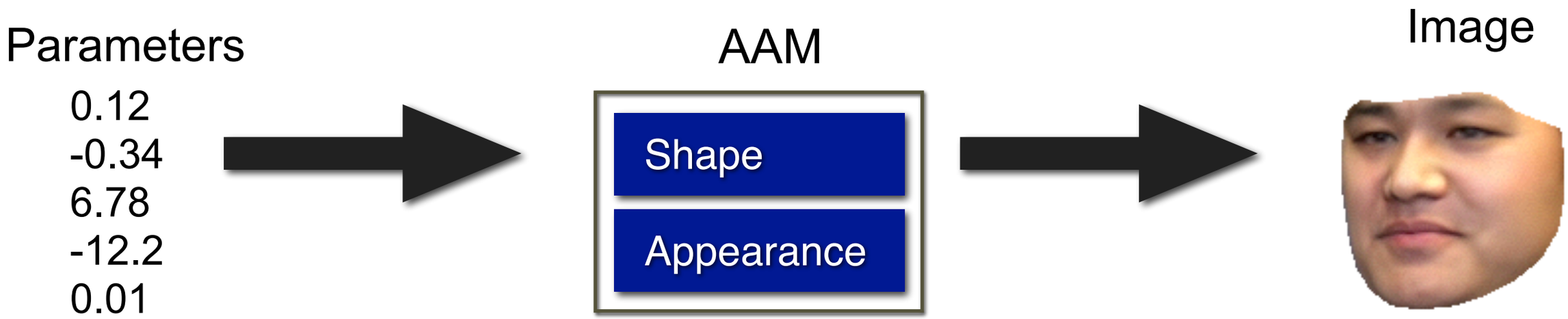
AAM



Determine the best model parameters to reconstruct the image

(Matthews and Gross)

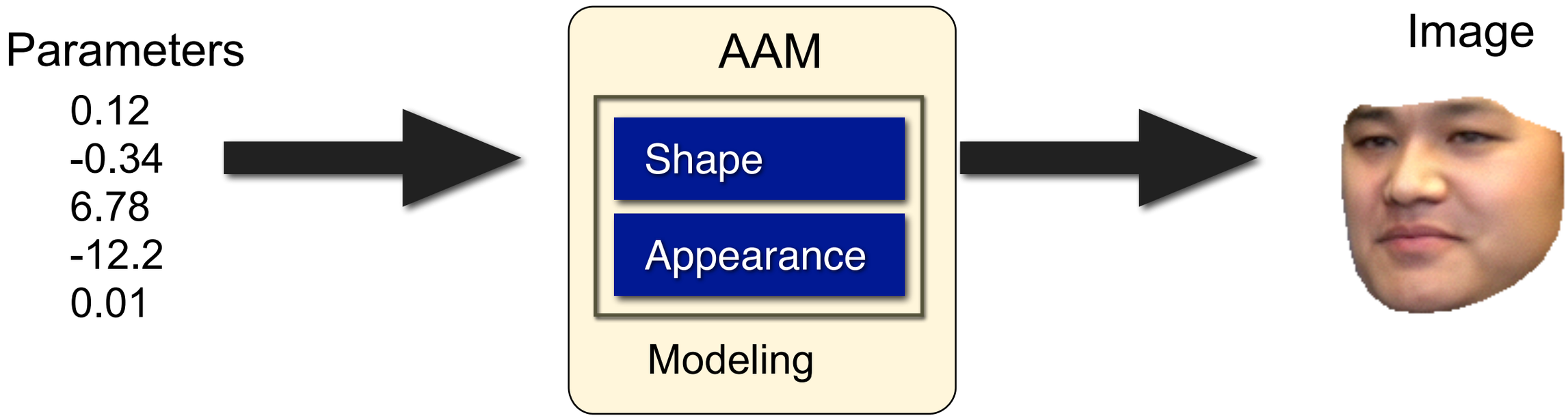
Active Appearance Models



Determine the best model parameters to reconstruct the image

(Matthews and Gross)

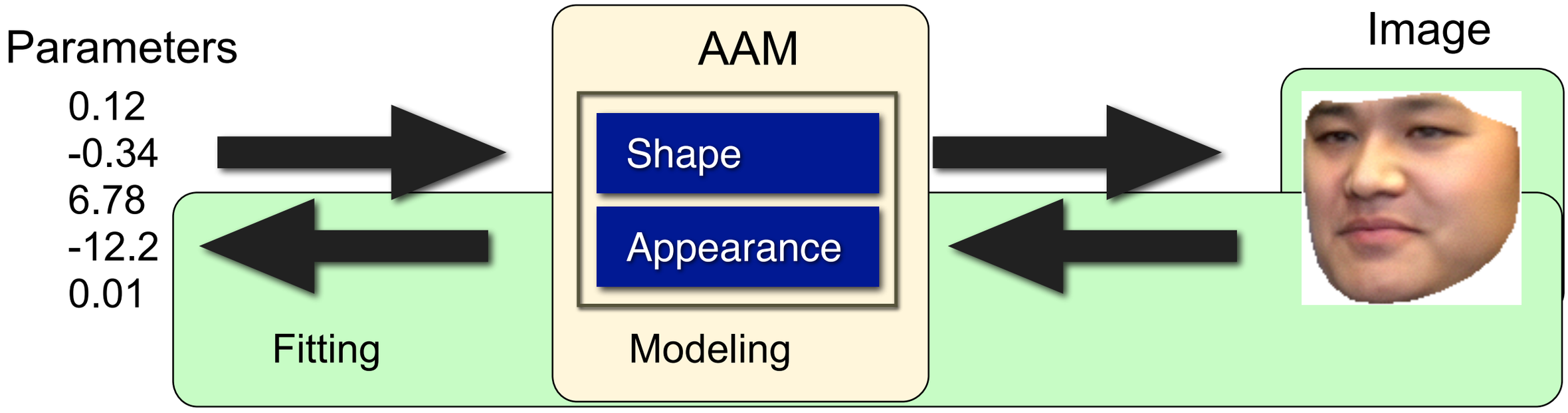
Active Appearance Models



Determine the best model parameters to reconstruct the image

(Matthews and Gross)

Active Appearance Models



Determine the best model parameters to reconstruct the image

(Matthews and Gross)

AAM - Fitting "Simultaneous"

- Now that we have defined an AAM, how should do the fitting?
- Turns out, we can still use the "simultaneous" extension of the LK algorithm,

$$\arg \min_{\Delta\boldsymbol{\lambda},\Delta\mathbf{p}} ||T(\mathbf{z}) + \mathbf{A}\Delta\boldsymbol{\lambda} - I(\mathcal{W}(\mathbf{z};\mathbf{p})) - \mathbf{J}\Delta\mathbf{p}||^2$$

- where,

$$\frac{\partial \mathcal{W}(\mathbf{z};\mathbf{p})}{\partial \mathbf{p}}^T = [\mathbf{s}_1, \ldots, \mathbf{s}_K] \qquad \mathbf{J} = \frac{\partial I(\mathcal{W}(\mathbf{z};\mathbf{p})}{\partial \mathcal{W}(\mathbf{z};\mathbf{p})}^T \frac{\partial \mathcal{W}(\mathbf{z};\mathbf{p})}{\partial \mathbf{p}}^T$$

AAM - Fitting "Inverse Composition"

- Even faster if we use the inverse composition (IC) extension as well as "projecting out" the appearance,

$$\arg\min_{\Delta\mathbf{p}} ||T(\mathbf{z}) + \mathbf{J}_{ic}\Delta\mathbf{p} - I(\mathcal{W}(\mathbf{z};\mathbf{p}))||^2_{\text{null}(\mathbf{A})}$$

- where,

$$\frac{\partial\mathcal{W}(\mathbf{z};\mathbf{0})}{\partial\mathbf{p}}^T = [\mathbf{s}_1,\ldots,\mathbf{s}_K] \qquad \mathbf{J}_{ic} = \frac{\partial T(\mathcal{W}(\mathbf{z};\mathbf{0})}{\partial\mathcal{W}(\mathbf{z};\mathbf{0})}^T \frac{\partial\mathcal{W}(\mathbf{z};\mathbf{0})}{\partial\mathbf{p}}^T$$

IC AAM Fitting

- Runs at 230 Hz on a 3.2GHz PC

Input

Shape Overlaid

Overlaid Model Instance

Rendered Model Instance

(Matthews and Gross)

IC AAM Fitting

- Runs at 230 Hz on a 3.2GHz PC

Input

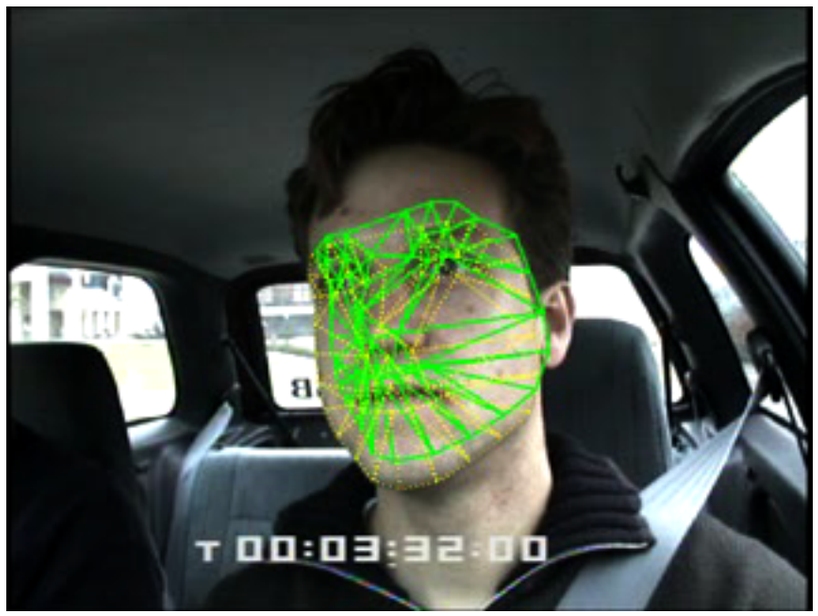
Shape Overlaid

Overlaid Model Instance

Rendered Model Instance

(Matthews and Gross)

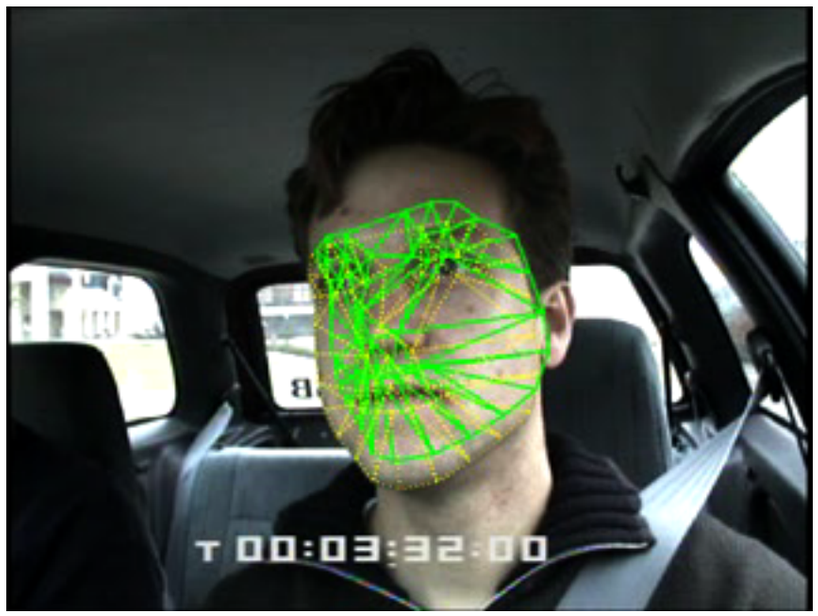
Why We Need High Speed?



Re-initialize model multiple times if tracking fails and still track in real time

(Matthews and Gross)

Why We Need High Speed?



Re-initialize model multiple times if tracking fails and still track in real time

(Matthews and Gross)

Not All Is Peachy

Original model does not handle occlusion well

(Matthews and Gross)

Not All Is Peachy

Original model does not handle occlusion well

(Matthews and Gross)

AAM - Fitting “Robust Error Function”

- To handle occlusions we can employ the robust error function,

$$\arg \min_{\Delta\boldsymbol{\lambda},\Delta\mathbf{p}} \eta(T(\mathbf{z}) + \mathbf{A}\Delta\boldsymbol{\lambda} - I(\mathcal{W}(\mathbf{z};\mathbf{p})) - \mathbf{J}\Delta\mathbf{p})$$

- where,

$$\frac{\partial \mathcal{W}(\mathbf{z};\mathbf{p})}{\partial \mathbf{p}}^T = [\mathbf{s}_1, \ldots, \mathbf{s}_K] \qquad \mathbf{J} = \frac{\partial I(\mathcal{W}(\mathbf{z};\mathbf{p})}{\partial \mathcal{W}(\mathbf{z};\mathbf{p})}^T \frac{\partial \mathcal{W}(\mathbf{z};\mathbf{p})}{\partial \mathbf{p}}^T$$

AAMs with Occlusion Modeling

(Matthews and Gross)

AAMs with Occlusion Modeling

(Matthews and Gross)

Applications

- **User Interfaces:**
  - Mouse replacement
  - Automotive: Windshield Displays, Smart Airbags
- **Face Recognition:**
  - Pose Normalization
- **Lipreading/Audio-Visual Speech Recognition**
- **Rendering and Animation:**
  - Low-Bandwidth Video Conferencing
  - Audio-Visual Speech Synthesis

(Matthews and Gross)

User Interfaces: Head Pose

- Mouse replacement

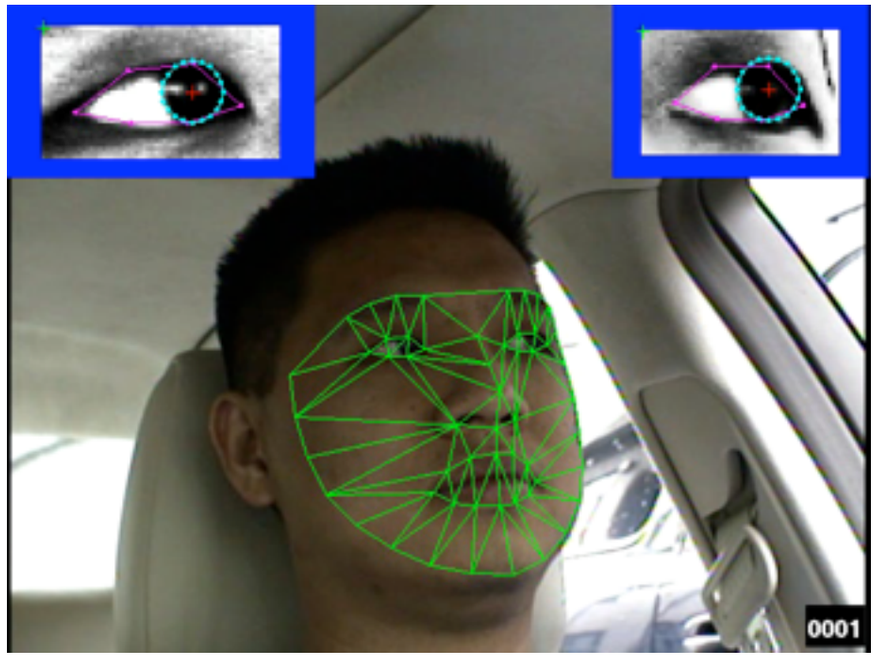
(Matthews and Gross)

User Interfaces: Head Pose

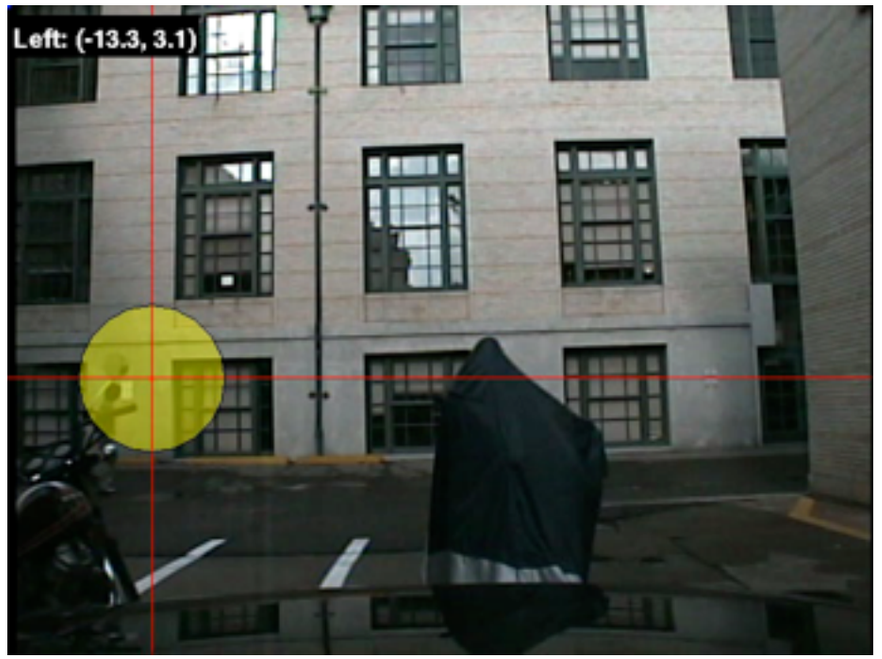
- Mouse replacement

(Matthews and Gross)

User Interfaces: Gaze Tracking



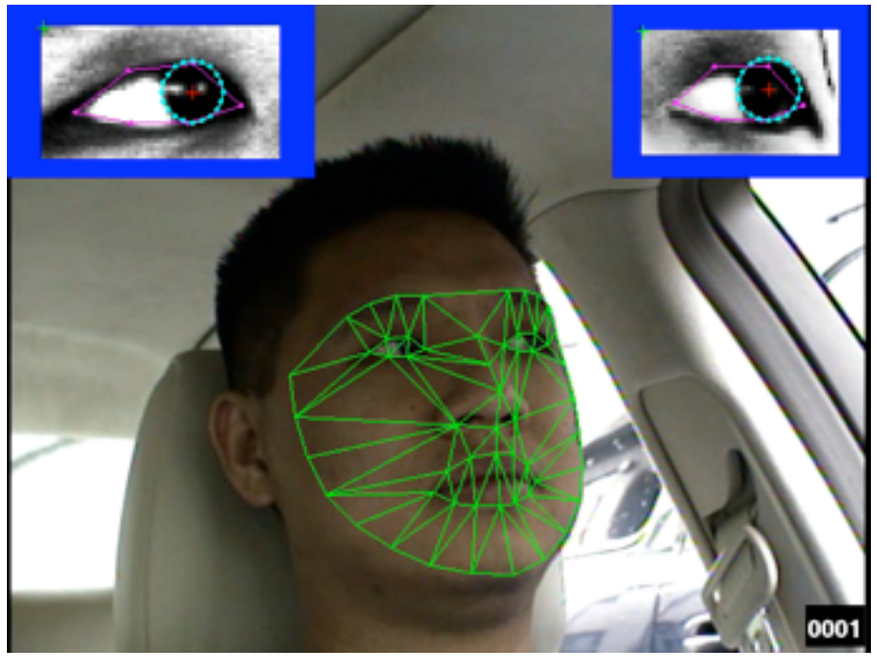
Driver Camera



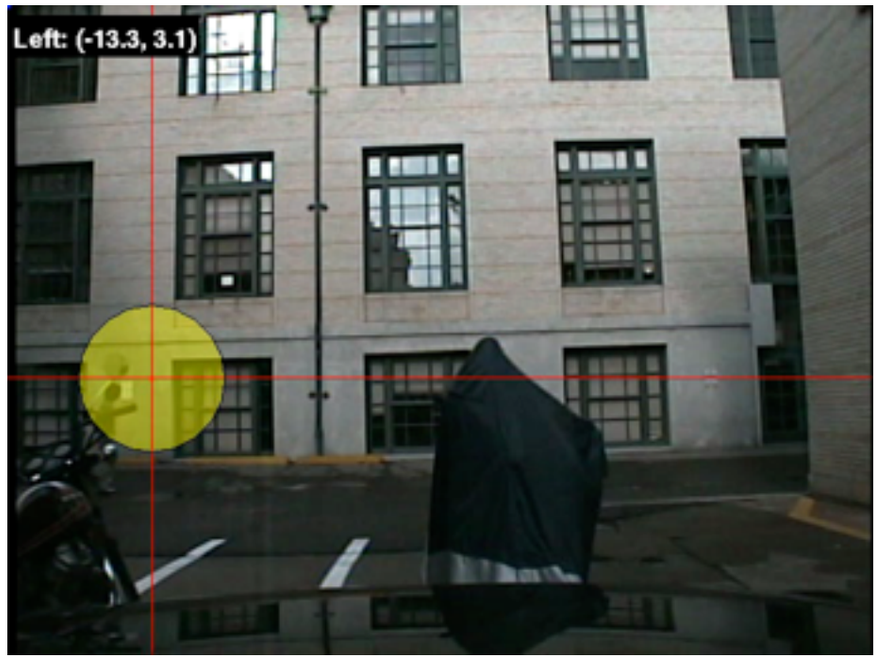
Exterior View Camera

(Matthews and Gross)

User Interfaces: Gaze Tracking



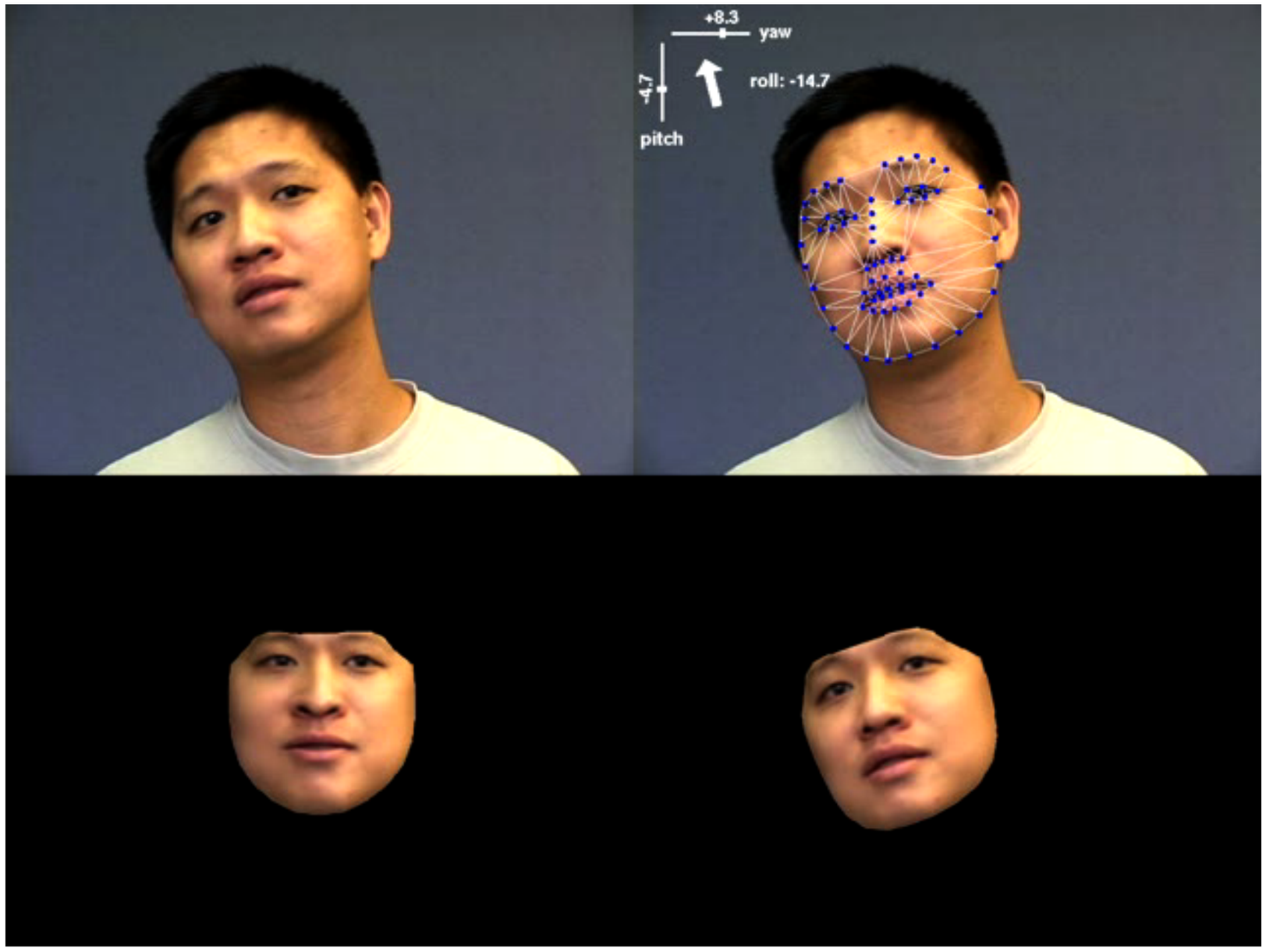
Driver Camera



Exterior View Camera

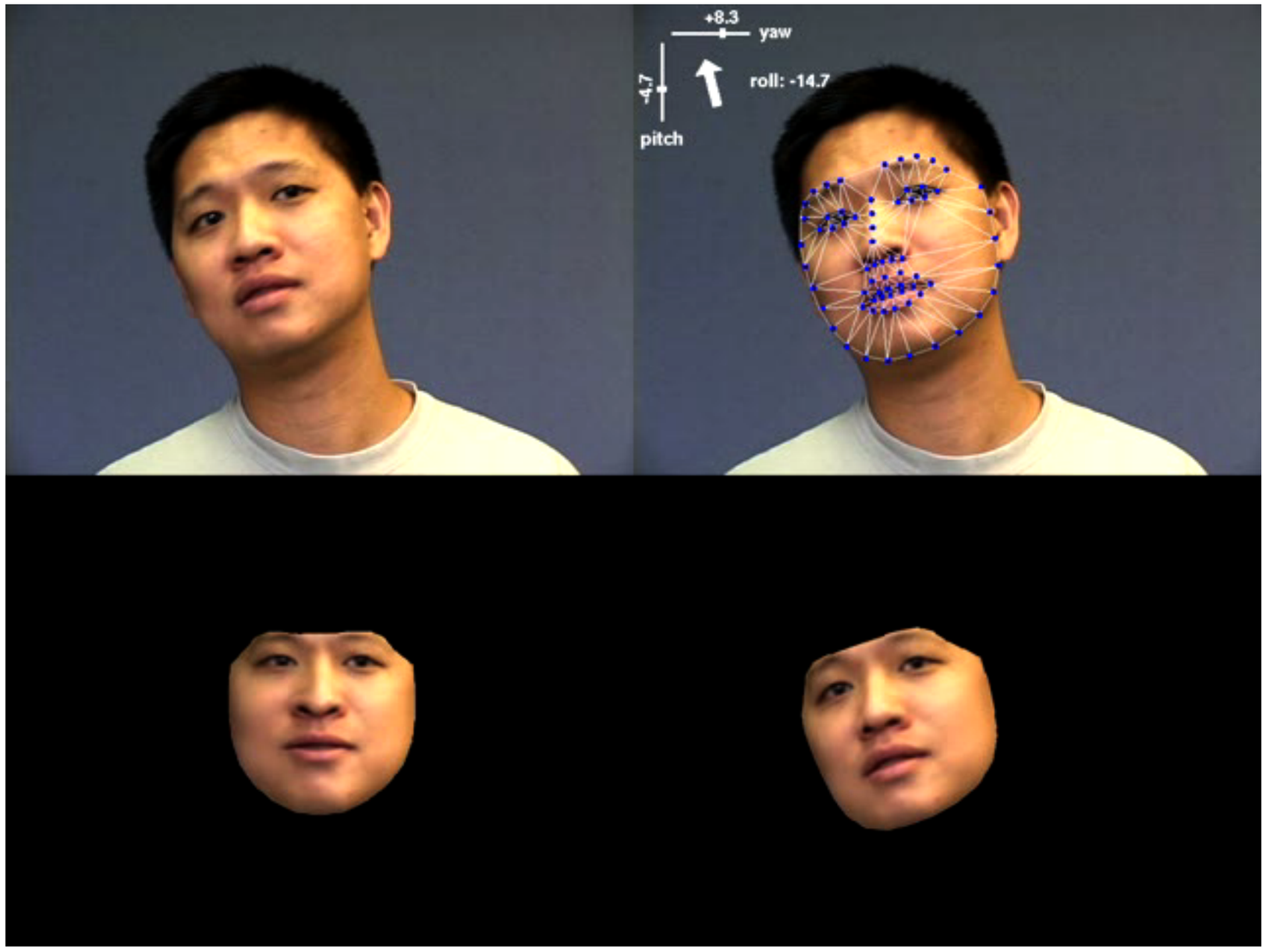
(Matthews and Gross)

Face Recognition: Pose Normalization



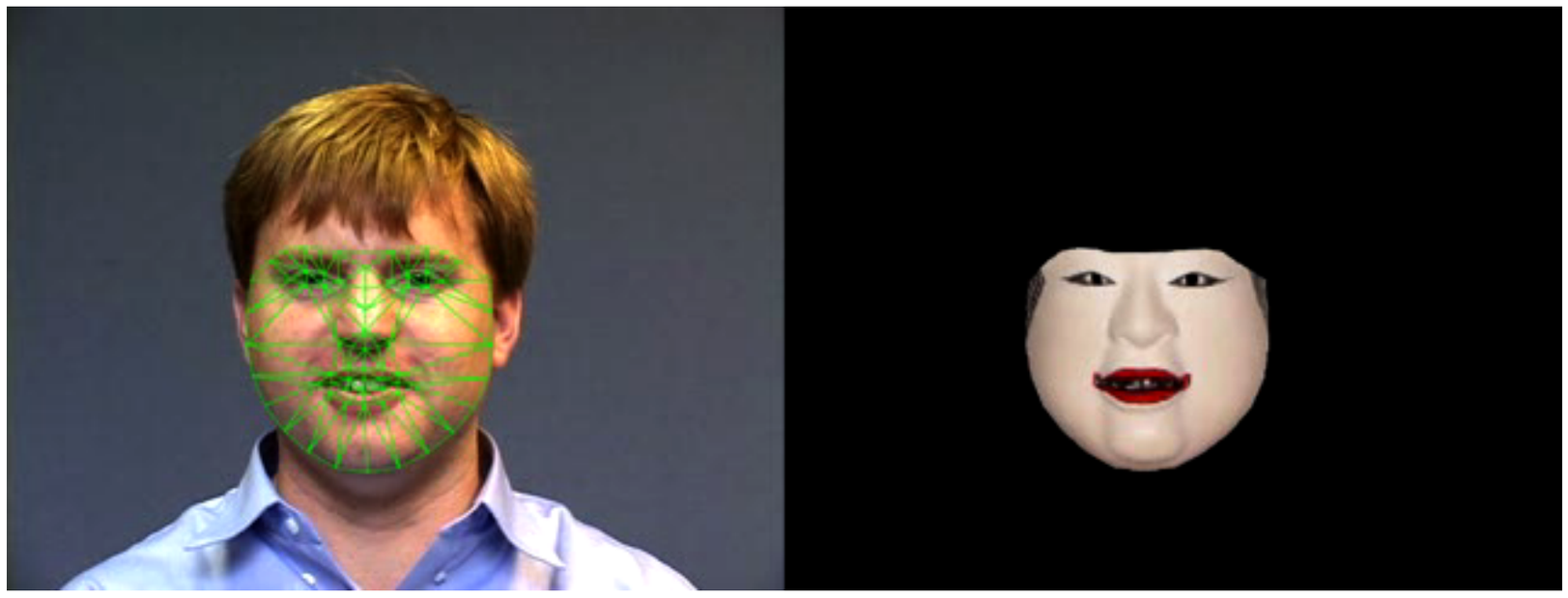
(Matthews and Gross)

Face Recognition: Pose Normalization



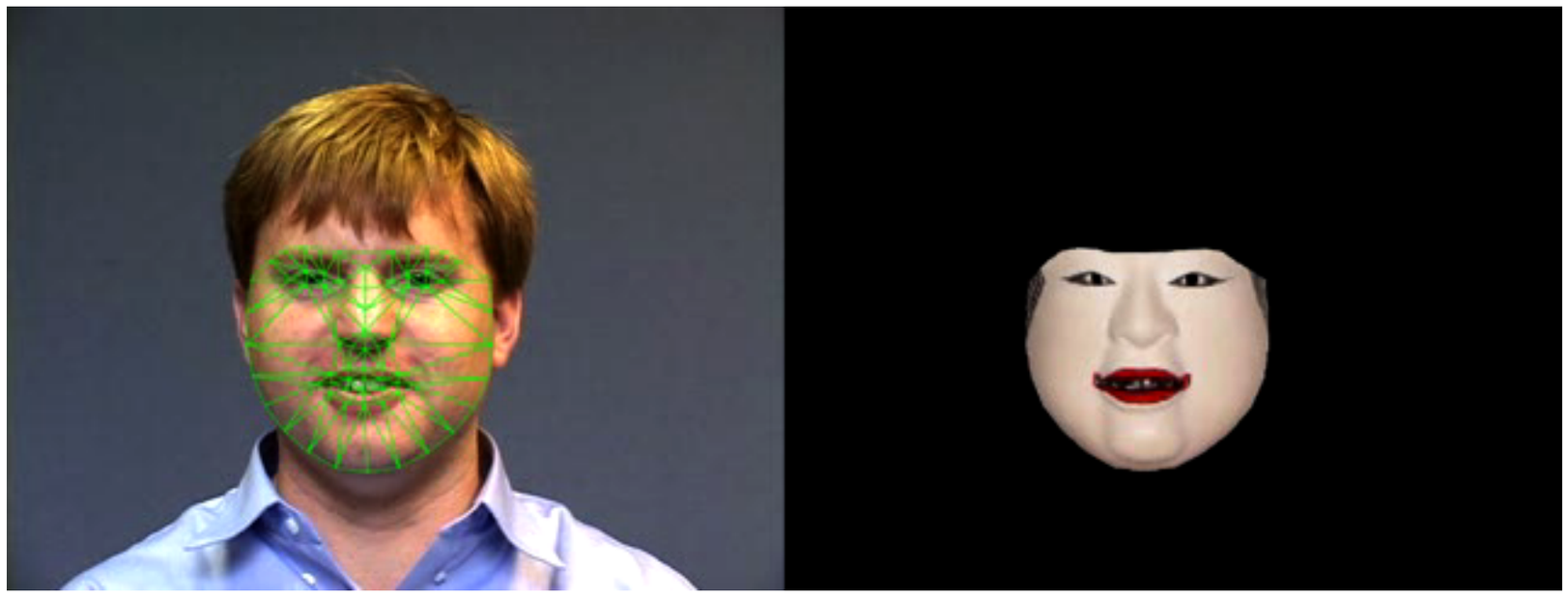
(Matthews and Gross)

Animation Generation



(Matthews and Gross)

Animation Generation



(Matthews and Gross)

Audio-Visual Speech Synthesis

Jingle Bells, Jingle Bells,
Jingle All the Way, ...

(Matthews and Gross)

Audio-Visual Speech Synthesis

Jingle Bells, Jingle Bells,
Jingle All the Way, ...

(Matthews and Gross)