# Constructing and Fitting Active Appearance Models With Occlusion

**Ralph Gross, Iain Matthews, and Simon Baker**

The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA
{rgross, iainm, simonb}@cs.cmu.edu

## Abstract

Active Appearance Models (AAMs) are generative parametric models that have been successfully used in the past to track faces in video. A variety of video applications are possible, including dynamic pose estimation for real-time user interfaces, lip-reading, and expression recognition. To construct an AAM, a number of training images of faces with a mesh of canonical feature points (usually hand-marked) are needed. All feature points have to be visible in all training images. However, in many scenarios parts of the face may be occluded. Perhaps the most common cause of occlusion is 3D pose variation, which can cause self-occlusion of the face. Furthermore, tracking using standard AAM fitting algorithms often fails in the presence of even small occlusions. In this paper we propose algorithms to construct AAMs from occluded training images and to efficiently track faces in videos containing occlusion. We evaluate our algorithms both quantitatively and qualitatively and show successful real-time face tracking on a number of image sequences containing varying degrees of occlusions.

## 1 Introduction

Active Appearance Models (AAMs) [5] (and the closely related concepts of Active Blobs [14] and Morphable Models [4]) are generative parametric models commonly used to track faces in video. AAMs are normally constructed by applying Procrustes analysis followed by Principal Components Analysis (PCA) to a collection of training images of faces with a mesh of canonical feature points (usually hand-marked) on them [5]. AAMs are then fit frame-by-frame to input videos to track the face through the video [5, 13]. The best fit model parameters are then used in whatever the chosen application is. A variety of video applications are possible, including dynamic pose estimation for real-time user interfaces, expression recognition, and lip-reading.

In many scenarios there is the opportunity for occlusion. The occlusion may occur in the training data used to construct the AAM, and/or in the input videos to which the AAM is fit. Perhaps the most common cause of occlusion is 3D pose variation, which often causes self-occlusion. For faces, other causes of occlusion include sunglasses and other objects placed in front of the face. Since occlusion is so common, it is important to be able to: (1) construct AAMs from occluded training images, and (2) efficiently fit AAMs to novel videos containing occlusion.

In Section 2 we describe how to construct AAMs with training data containing occlusion. We first generalize the Procrustes alignment algorithm. We then show how to apply Principal Component Analysis with missing data [15, 16] to compute the shape and appearance variation.

In Section 3 we show how to *efficiently* track an AAM with occlusion. While it may seem that fitting with occlusion is simply a matter of adding a robust error function, if we wish to retain both high efficiency and robust performance, the task is more difficult. The naïve Gauss-Newton algorithm is very slow [3] requiring minutes per frame. Efficient robust fitting algorithms have been proposed [10], however, as we will show, these algorithms make approximations that adversely affect their robustness. See Figure 9.

We begin Section 3 by first describing our previously introduced (non-robust) project-out inverse compositional AAM fitting algorithm [13]. In Section 3.2 we show that the naïve robust extension to this algorithm is very inefficient. We then propose a novel (non-robust) fitting algorithm, the normalization inverse compositional algorithm in Section 3.3 and empirically show its equivalence to the project-out algorithm. In Section 3.4 we describe the robust extension to the normalization algorithm and show in Section 3.5 how to implement the robust normalization algorithm efficiently. In Section 3.6 we demonstrate successful face tracking using the robust normalization algorithm on a number of image sequences containing occlusion. The overall tracking algorithm runs at around 8 frames-per-second in Matlab and around 50 frames-per-second in C.

## 2 Construction With Occlusion

We first define AAMs and then describe how they are constructed from training data with occlusion. The input consists of a collection of training images of the phenomenon to be modeled with the location of all of the *visible* mesh vertices in each of the images marked. Due to self-occlusion or occlusion by an object, only a subset of the vertices may be visible in any given training image.

## 2.1 Shape

The *shape* of an AAM is defined by a triangulated mesh and in particular the vertex locations of the mesh. Mathematically, we define the shape $\mathbf{s}$ of an AAM as the $xy$-coordinates of the $v$ vertices that make up the mesh:

$$\mathbf{s} = (x_1, y_1, x_2, y_2, \ldots, x_v, y_v)^{\mathrm{T}}. \quad (1)$$

AAMs allow linear shape variation; i.e. the shape $\mathbf{s}$ can be expressed as a base shape $\mathbf{s}_0$ plus a linear combination of $n$ shape vectors $\mathbf{s}_i$:

$$\mathbf{s} = \mathbf{s}_0 + \sum_{i=1}^{n} p_i \mathbf{s}_i \quad (2)$$

where the coefficients $p_i$ are the shape parameters. Since we can perform a linear re-parameterization, wherever necessary we assume that the shape vectors $\mathbf{s}_i$ are orthonormal.

### 2.1.1 Computing the Base Mesh: $\mathbf{s}_0$

In traditional AAMs all of the mesh vertices $\mathbf{s}$ are marked in every training image. The base mesh $\mathbf{s}_0$ is then constructed using the *Procrustes* algorithm [12]. In the presence of occlusion the situation is complicated by the fact that not all of the mesh vertices are marked in every training image. The outline of the *Procrustes* algorithm stays the same, however only vertices visible in a given training image are used.

1. Initialize the base mesh $\mathbf{s}_0$ to be the visible vertices of the mesh $\mathbf{s}$ of any one of the training images.
2. Repeat until the estimate of $\mathbf{s}_0$ converges:
   (a) For each training image, align $\mathbf{s}$ to the current $\mathbf{s}_0$ with a similarity transform using the vertices common to $\mathbf{s}$ and $\mathbf{s}_0$.
   (b) Update $\mathbf{s}_0$ as the mean of all of the aligned meshes $\mathbf{s}$.

In Step 2. only images are used where there is substantial overlap between their visible $\mathbf{s}$ and the current estimate of $\mathbf{s}_0$. In our implementation, substantial overlap means over 50% of the vertices in $\mathbf{s}$ are in $\mathbf{s}_0$. In Step 2b. only the vertices that appear in at least one of the $\mathbf{s}$ are updated. The mean for each vertex is computed across the images in which it is visible.

### 2.1.2 Computing the Shape Variation: $\mathbf{s}_i$

In traditional AAMs the shape vectors $\mathbf{s}_i$ are computed by first aligning every training shape vector $\mathbf{s}$ with the base mesh $\mathbf{s}_0$ using a similarity transform [5]. The mean shape (i.e. the base mesh $\mathbf{s}_0$) is subtracted from each shape vector. Principal Components Analysis [9] is then performed on the aligned shape vectors $\mathbf{s}$. In the case of occlusion only the visible vertices are aligned to the base mesh. Principal Components Analysis with missing data [15,16] is then performed on the aligned shape vectors $\mathbf{s}$. The shape vectors $\mathbf{s}_i$ are then set to be the orthonormalized eigenvectors with the largest eigenvalues. As is common practice [5] we retain enough shape modes to explain $95\%$ of the observed variation in the training set.

(a) (b)

Figure 1: Occluded data. (a) Original images with all mesh vertices $\mathbf{s}$ visible. (b) Images with 40% of the face region occluded. Only non-occluded vertices are used in the AAM construction.

## 2.2 Appearance

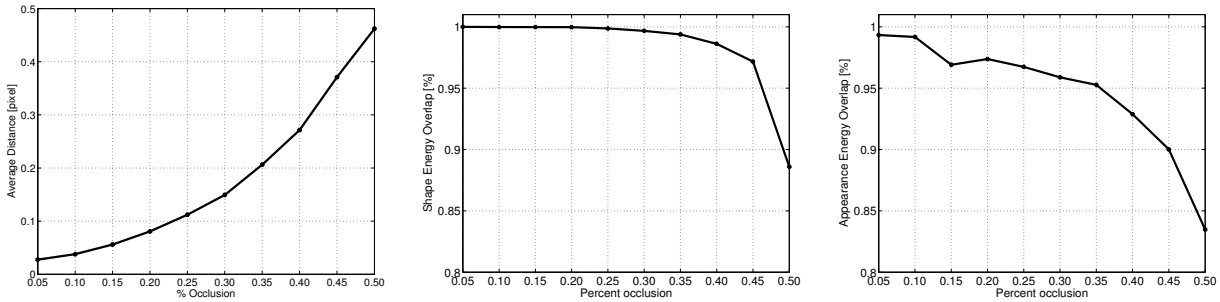
As a convenient abuse of terminology, let $\mathbf{s}_0$ also denote the pixels $\mathbf{x} = (x, y)^{\mathrm{T}}$ that lie inside the base mesh $\mathbf{s}_0$. The *appearance* of a AAM is then an image $A(\mathbf{x})$ defined over the pixels $\mathbf{x} \in \mathbf{s}_0$. AAMs allow linear appearance variation. This means that the appearance $A(\mathbf{x})$ can be expressed as a base appearance $A_0(\mathbf{x})$, plus a linear combination of $m$ appearance images $A_i(\mathbf{x})$:

$$A(\mathbf{x}) = A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) \qquad \forall\, \mathbf{x} \in \mathbf{s}_0 \quad (3)$$

where the coefficients $\lambda_i$ are the appearance parameters. As in Section 2.1, wherever necessary we assume that the images $A_i$ are orthonormal.

### 2.2.1 Computing the Appearance Variation $A_i$

In traditional AAMs the appearance vectors $A_i$ are computed by warping all of the input images onto the base mesh using the piecewise affine warps defined between the training shape vector $\mathbf{s}$ and the base mesh $\mathbf{s}_0$ [5]. Principal Components Analysis is then applied to the resulting images. In the case of occlusion the shape normalized input images are incomplete. If any of the vertices of a triangle are not visible in the training image, that triangle will be missing in the training image. Again, we use Principal Components Analysis with missing data [15, 16] to compute the appearance vectors $A_i$. The appearance vectors $A_i$ are then set to be the orthonormalized eigenvectors with the largest eigenvalues. We again retain enough appearance modes to explain $95\%$ of the observed variation in the training set.




(a) Base mesh distance (b) Shape energy overlap **SE** (c) Appearance energy overlap **AE**

Figure 2: Comparison of the AAM model components base mesh, shape variation and appearance variation computed from unoccluded and occluded data. (a) Average pixel distance between base meshes $\mathbf{s_0}$. (b) Shape energy overlap **SE**. (c) Appearance energy overlap **AE**. For all three components a high degree of similarity is evident. At around $50\%$ occlusion, however, the performance drops off rapidly.

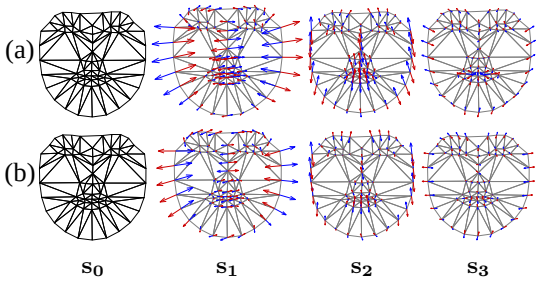



Figure 3: Mean shape $\mathbf{s_0}$ and shape variations $\mathbf{s_1} - \mathbf{s_3}$ overlaid on the base mesh. (a) Shape images computed from unoccluded data. (b) Shape images computed from data with $50\%$ occlusion. The resulting shape modes are very similar.

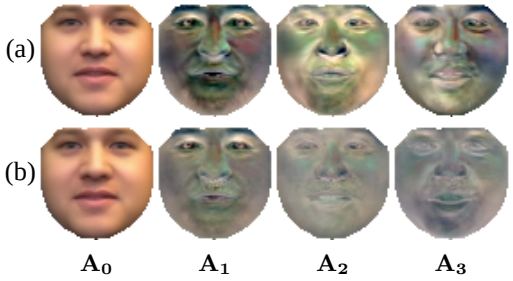



Figure 4: Mean appearance $\mathbf{A_0}$ and appearance variations $\mathbf{A_1} - \mathbf{A_3}$. (a) Appearance images computed from unoccluded data. (b) Appearance images computed from data with $50\%$ occlusion.

## 2.3 Experiments

In order to evaluate AAMs constructed with occlusion we start with fully labeled image sequences of five subjects in which randomly selected regions are artificially occluded. In total 900 training images were used. See Figure 1 for examples. Results are reported for occluding regions ranging in size from $5 - 50\%$ of the total face region. We compare the base mesh $\mathbf{s}_0$, shape and appearance models for unoccluded and occluded training data.

### 2.3.1 Base Mesh

The base mesh for this dataset contains 68 vertices. In Figure 2(a) we compare the pixel distance between base meshes computed from unoccluded and occluded training data averaged over the 68 vertices. While the average pixel distance increases with higher levels of occlusion, it stays below 0.5 pixels even for the maximal occlusion of $50\%$.

### 2.3.2 Shape Variation

Figure 3 shows the base mesh $\mathbf{s_0}$ and shape variations $\mathbf{s_1} - \mathbf{s_3}$ computed from unoccluded and occluded data. The resulting shape models are very similar. In order to quantify the similarity of the shape modes we measure the shape energy overlap **SE** between shape variations $\mathbf{s}_i^u$ and $\mathbf{s}_i^o$ computed from unoccluded and occluded data, respectively. The energy overlap computes dot products between the shape vectors and is defined as $\mathbf{SE} = \frac{1}{n}\sum_i \sqrt{\sum_j ((s_i^u)^T s_j^o)^2}$ for $i, j = 0, \ldots, n$, where $\mathbf{n}$ refers to the number of shape modes. **SE** ranges in value from $0$ to $1$. Figure 2(b) plots **SE** values for different occlusion sizes. Overall the energy overlap declines slowly. It stays above $95\%$ for up to $45\%$ occlusion and then drops off rapidly.

### 2.3.3 Appearance Variation

Figure 4 shows the mean appearance $\mathbf{A_0}$ and appearance variations $\mathbf{A_1} - \mathbf{A_3}$ computed from unoccluded and occluded data. The resulting mean appearance images look very similar. Since it is hard to interpret the appearance eigenvectors we again quantify the similarity of the appearance models with the appearance energy overlap **AE** which is defined analogously to **SE**. Figure 2(c) plots **AE** values for different occlusion sizes. The **AE** values decline slightly faster than the **SE** values, possibly due to the much higher dimensionality of the appearance images. However, the appearance energy overlap still stays above $90\%$ for up to $45\%$ occlusion.

### 2.3.4 Face Tracking

Finally we validate that an AAM constructed with occlusion can still successfully be used to track a face. We use 120 training images containing self-occlusion (full left and

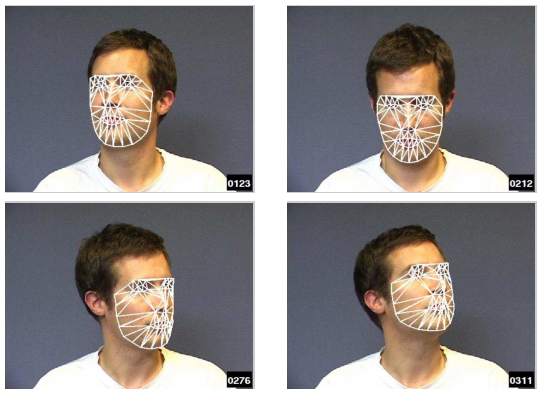



Figure 5: Example frames of a test sequence showing accurate tracking with an AAM constructed with occlusion. See the accompanying movie `fit.mpg` for the full sequence of 457 frames.

right profile views) and occlusion by an object to build the AAM. In the training set $18\%$ of the feature points are occluded. The AAM successfully tracks a face in an independent test sequence. Figure 5 shows example frames with the fitted mesh overlaid on the input image. The accompanying movie[1] `fit.mpg` includes the full sequence of 457 frames.

# 3 Fitting AAMs With Occlusion

We now describe how to track an occluded face in a video with an AAM, both efficiently and robustly. We first describe our previously proposed (non-robust) AAM fitting algorithm [13] and show how it can be modified to robustly fit AAMs. The resulting algorithm is robust, but inefficient. We then propose a different robust fitting algorithm which can be implemented efficiently and empirically demonstrate its ability track occluded faces, robustly and in real-time.

## 3.1 Background

Fitting a AAM is usually formulated [13] as minimizing the sum of squares difference between the model instance $A(\mathbf{x}) = A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x})$ and the input image warped back onto the base mesh $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$:

$$\sum_{\mathbf{x}\in\mathbf{s}_0} \left[ A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2 \quad (4)$$

where the sum is performed over all of the pixels $\mathbf{x}$ in the base mesh $\mathbf{s}_0$. In this equation, the warp $\mathbf{W}$ is the piecewise affine warp from the base mesh $\mathbf{s}_0$ to the current AAM shape $\mathbf{s}$ defined by the vertices. Hence, $\mathbf{W}$ is a function of the shape parameters $\mathbf{p}$. For ease of notation, in this paper we have omitted mention of the 2D similarity transformation that is used to normalize the shape of an AAM. In [13] we showed how to include this warp into $\mathbf{W}$. The

[1] Movies are available at http://www.ri.cmu.edu/project/project_448.html

**The Project-Out Inverse Compositional Algorithm**

**Pre-Computation:**

(P1) Evaluate the gradient of the base appearance $\boldsymbol{\nabla} A_0$
(P2) Evaluate the Jacobian of the warp $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x};\mathbf{0})$
(P3) Compute the steepest descent images $\mathbf{SD}_{\mathrm{ic}}(\mathbf{x})$ (Eqn. (5))
(P4) Project out appearance from $\mathbf{SD}_{\mathrm{ic}}(\mathbf{x})$ (Eqn. (6))
(P5) Compute the Hessian matrix $H_{\mathrm{po}}$ (Eqn. (8))

**Iterate:**

(I1) Warp $I$ with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
(I2) Compute the error image $E(\mathbf{x}) = I(\mathbf{W}(\mathbf{x};\mathbf{p})) - \mathbf{A_0}(\mathbf{x})$
(I3) Compute $\sum_{\mathbf{x}} \mathbf{SD}_{\mathrm{po}}^{\mathrm{T}}(\mathbf{x}) E(\mathbf{x})$
(I4) Compute $\Delta\mathbf{p} = -H_{\mathrm{po}}^{-1} \sum_{\mathbf{x}} \mathbf{SD}_{\mathrm{po}}^{\mathrm{T}}(\mathbf{x}) E(\mathbf{x})$
(I5) Update the warp $\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$

**Compute appearance parameters:**

(A1) Compute $\lambda_i = \sum_{\mathbf{x}\in\mathbf{s}_0} A_i(\mathbf{x}) \cdot [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - A_0(\mathbf{x})]$

Figure 6: The project-out inverse compositional algorithm [13].

goal of AAM fitting is to minimize the expression in Equation (4) simultaneously with respect to the shape $\mathbf{p}$ and appearance $\boldsymbol{\lambda}$ parameters. The "project-out" inverse compositional algorithm and its extension to 2D AAMs was proposed in [13]. See Figure 6 for a summary. The algorithm performs the non-linear optimization of Equation 4 in two steps (similar to Hager and Belhumeur [10]). The shape parameters $\mathbf{p}$ are found through non-linear optimization in a subspace in which the appearance variation can be ignored. This is achieved by "projecting out" the appearance variation from the *steepest-descent images*:

$$\mathbf{SD}_{\mathrm{ic}}(\mathbf{x}) = \boldsymbol{\nabla} A_0 \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \quad (5)$$

by computing:

$$\mathbf{SD}_{\mathrm{po}}(\mathbf{x}) = \mathbf{SD}_{\mathrm{ic}} - \sum_{i=1}^{m} \left[ \sum_{\mathbf{x}\in\mathbf{s}_0} A_i(\mathbf{x}) \mathbf{SD}_{\mathrm{ic}}(\mathbf{x}) \right] A_i(\mathbf{x}). \quad (6)$$

Equation (6) requires the appearance images $A_i$ to be orthonormal. In each iteration of the algorithm, the input image is warped with the current estimate of the warp to estimate $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$, the base appearance subtracted to give the error image $E(\mathbf{x}) = I(\mathbf{W}(\mathbf{x};\mathbf{p})) - A_0(\mathbf{x})$, and the incremental parameter updates computed:

$$\Delta\mathbf{p} = -H_{\mathrm{po}}^{-1} \sum_{\mathbf{x}\in\mathbf{s}_0} \mathbf{SD}_{\mathrm{po}}(\mathbf{x}) [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - A_0(\mathbf{x})] \quad (7)$$

using the *Gauss-Newton Hessian*:

$$H_{\mathrm{po}} = \sum_{\mathbf{x}\in\mathbf{s}_0} \mathbf{SD}_{\mathrm{po}}(\mathbf{x})^{\mathrm{T}} \mathbf{SD}_{\mathrm{po}}(\mathbf{x}). \quad (8)$$

The incremental warp $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ is then *inverted* and *composed* with the current estimate to give the new estimate

$\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$. The appearance parameters $\boldsymbol{\lambda}$ can then be computed as:

$$\lambda_i \;=\; \sum_{\mathbf{x}\in\mathbf{s}_0} A_i(\mathbf{x}) \cdot [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - A_0(\mathbf{x})] \,. \qquad (9)$$

If there are $n$ shape parameters, $m$ appearance parameters, and $N$ pixels in the base appearance $A_0$, the precomputation takes time $\mathrm{O}(n^2 \cdot N + m \cdot N)$ where the slowest step is the computation of the Hessian in Step P5 which alone takes time $\mathrm{O}(n^2 \cdot N)$. The online cost per iteration is just $\mathrm{O}(n \cdot N + n^3)$ and the post-computation cost is $\mathrm{O}(m \cdot N)$. In all cases we iterate the algorithm a sufficient (fixed) number of times. A implementation of this algorithm in "C" runs at 230 frames per second on a dual 3GHz Pentium 4 Xeon for typical values of $n$, $m$ and $N$ [13].

## 3.2 Robust Fitting

Occluded pixels in the input image can be viewed as "outliers". In order to deal with outliers in a least-squares optimization framework a robust error function can be used. The goal for *robustly* fitting a AAM is then to minimize

$$\sum_{\mathbf{x}\in\mathbf{s}_0} \varrho\left(\left[A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\right]^2 ; \sigma\right) \qquad (10)$$

with respect to the shape $\mathbf{p}$ and appearance $\boldsymbol{\lambda}$ parameters where $\varrho(t;\boldsymbol{\sigma})$ is a symmetric *robust error function* [11] and $\boldsymbol{\sigma}$ is a vector of *scale parameters*. For ease of explanation we treat the scale parameters as known constants and drop them in the following. In comparison to the project-out algorithm the expressions for the incremental parameter update $\Delta\mathbf{p}$ (Equation 7) and the Hessian $H_{\mathrm{po}}$ (Equation 8) have to be weighted by the error function $\varrho'(E_{\mathrm{app}}(\mathbf{x})^2)$, where:

$$E_{\mathrm{app}}(\mathbf{x}) = A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \qquad (11)$$

Equation (7) then becomes:

$$\Delta\mathbf{p} \;=\; -H_\rho^{-1} \sum_{\mathbf{x}\in\mathbf{s}_0} \varrho'(E_{\mathrm{app}}(\mathbf{x})^2)\mathbf{SD}_{\mathrm{po}}(\mathbf{x})E_{\mathrm{app}}(\mathbf{x}) \qquad (12)$$

with:

$$H_\rho \;=\; \sum_{\mathbf{x}\in\mathbf{s}_0} \varrho'(E_{\mathrm{app}}(\mathbf{x})^2)\mathbf{SD}_{\mathrm{po}}(\mathbf{x})^{\mathrm{T}}\mathbf{SD}_{\mathrm{po}}(\mathbf{x}). \qquad (13)$$

The steepest descent images $\mathbf{SD}_{\mathrm{po}}$ also have to be recomputed because the appearance images are no longer orthonormal. The appearance images $A_i$ must be reorthonormalized with respect to the inner product:

$$\sum_{\mathbf{x}} \varrho'\left(E_{\mathrm{app}}(\mathbf{x})^2\right)\, A_i(\mathbf{x})\, A_j(\mathbf{x}) \;=\; \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \qquad (14)$$

Steps (P3)-(P5) in Figure 6 can therefore no longer be precomputed and have to be moved inside the iteration. As a result the robust project-out inverse compositional algorithm is very inefficient. See [2] for details. An approximation is to ignore the lack of orthogonality and just continue to use the Euclidean project out steepest descent images. This approach is taken in [10], where the *H*-Algorithm [7] is used to keep the Hessian constant to yield an efficient algorithm. As we will show in Section 3.6 this approximation leads to poor performance.

## 3.3 Project-out vs. Normalization

We now describe a slightly different algorithm to minimize the expression in Equation (4), the normalization inverse compositional algorithm [2]. As we will show, the robust extension of the normalization algorithm can be implemented very efficiently. An alternative way of dealing with the linear appearance variation in Equation (4) is to project out the appearance images $A_i$ from the *single* error image $E_{\mathrm{app}}$ rather than the large number of steepest descent images $\mathbf{SD}_{\mathrm{ic}}$. This normalization can be achieved by normalizing the error image so that the component of the error image in the direction $A_i$ is zero. In particular, the normalization step consists of:

$$\begin{aligned} \lambda_i \;&=\; -\sum_{\mathbf{x}} A_i(\mathbf{x})\, E(\mathbf{x}) \;\; \text{for } i = 1, \ldots, m \\ E_{\mathrm{app}}(\mathbf{x}) \;&\leftarrow\; E(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i\, A_i(\mathbf{x}). \end{aligned} \qquad (15)$$

As indicated, in the process of normalizing $E_{app}$ in this way the appearance parameters $\lambda_i$ are estimated. In comparison to the project-out algorithm in Figure 6 steps (P4) and (A1) are removed and the normalization step of Equation (15) is added after the computation of the error image $E(\mathbf{x})$ in step (I2). The equivalence of the project-out and normalization algorithms is shown empirically in Section 3.6.

## 3.4 Robust Normalization Algorithm

The goal of the normalization step in Equation (15) is to make the component of the error image in the direction $A_i$ to be zero, whilst computing $\lambda_i$ at the same time. We now need to formulate this problem using the robust error function. We wish to compute updates to the appearance parameters $\Delta\boldsymbol{\lambda} = (\Delta\lambda_1, \ldots, \Delta\lambda_m)^{\mathrm{T}}$ that minimize:

$$\sum_{\mathbf{x}} \varrho'\left(E_{\mathrm{app}}(\mathbf{x})^2\right) \left[E_{\mathrm{app}}(\mathbf{x}) + \sum_{i=1}^{m} \Delta\lambda_i A_i(\mathbf{x})\right]^2 . \qquad (16)$$

The least squares minimum of this expression is:

$$\Delta\boldsymbol{\lambda} \;=\; -H_{\mathbf{A}}^{-1} \sum_{\mathbf{x}} \varrho'\left(E_{\mathrm{app}}(\mathbf{x})^2\right) \mathbf{A}^{\mathrm{T}}(\mathbf{x}) E_{\mathrm{app}}(\mathbf{x}) \qquad (17)$$

**Efficient Robust Normalization Algorithm**

**Pre-Computation:**

(P1) Evaluate the gradient of the base appearance $\nabla A_0$
(P2) Evaluate the Jacobian of the warp $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x}; \mathbf{0})$
(P3) Compute the steepest descent images $\mathbf{SD}_{\mathrm{ic}}(\mathbf{x})$ (Eqn. (5))
(P4) Compute Hessian $H_\rho^i$ for each triangle (Eqn. (21))
(P5) Compute appearance Hessian $H_A^i$ for each triangle

**Iterate:**

(I1) Warp $I$ with $\mathbf{W}(\mathbf{x}; \mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$
(I2) Compute the error image $E_{\mathrm{app}}(\mathbf{x})$ (Eqn. (11))
(I3) Compute $H_A = \sum_i \varrho_i' \cdot H_A^i$
(I4) Compute $\Delta\boldsymbol{\lambda}$ and update $\lambda$ and $E_{\mathrm{app}}(\mathbf{x})$ (Eqn. (17))
(I5) Compute the Hessian $H_\rho$ and invert it (Eqn. (22))
(I6) Compute $\sum_{\mathbf{x}} \varrho' \left(E_{\mathrm{app}}(\mathbf{x})^2\right) \mathbf{SD}_{\mathrm{ic}}^{\mathrm{T}}(\mathbf{x}) E_{\mathrm{app}}(\mathbf{x})$
(I7) Compute $\Delta\mathbf{p} = -H_\varrho^{-1} \sum_{\mathbf{x}} \varrho' \left(E_{\mathrm{app}}(\mathbf{x})^2\right) \mathbf{SD}_{\mathrm{ic}}^{\mathrm{T}}(\mathbf{x}) E_{\mathrm{app}}(\mathbf{x})$
(I8) Update the warp $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta\mathbf{p})^{-1}$

Figure 7: The efficient robust normalization inverse compositional image alignment algorithm.

where $\mathbf{A}(\mathbf{x}) = (A_1(\mathbf{x}), \ldots, A_m(\mathbf{x}))$ and $H_{\mathbf{A}}$ is the appearance Hessian:

$$H_{\mathbf{A}} = \sum_{\mathbf{x}} \varrho' \left( E_{\mathrm{app}}(\mathbf{x})^2 \right) \mathbf{A}(\mathbf{x})^{\mathbf{T}} \mathbf{A}(\mathbf{x}). \quad (18)$$

The steepest descent parameter updates and the Hessian are computed as in Equations (12) and (13). Note that we avoid re-orthonormalization of the appearance images $A_i$ in every iteration as is required in the robust fitting algorithm of Section 3.2.

## 3.5 Efficient Robust Fitting

Due to the computation of the appearance Hessian $H_A$ and the Hessian $H_\rho$ in every iteration the robust normalization algorithm is also inefficient. However, most of this computation can be moved outside of the iteration if we assume that the outliers are spatially coherent. To make use of this assumption we subdivide the base appearance $\mathbf{A_0}$ into triangles according to the triangulation of the base mesh $\mathbf{s_0}$. Suppose there are $K$ triangles $T_1, T_2, \ldots, T_K$ with $N_i$ pixels in the $i^{\mathrm{th}}_K$ triangle. Equation (13) can then be rewritten:

$$H_\rho = \sum_{i=1}^{K} \sum_{\mathbf{x} \in T_i} \varrho' \left(E_{\mathrm{app}}(\mathbf{x})^2\right) \mathbf{SD}_{\mathrm{ic}}^{\mathrm{T}}(\mathbf{x}) \, \mathbf{SD}_{\mathrm{ic}}(\mathbf{x}). \quad (19)$$

Based on the spatial coherence of the outliers [1], assume that $\varrho'(E_{\mathrm{app}}(\mathbf{x})^2)$ is constant in each triangle; i.e. assume $\varrho'(E_{\mathrm{app}}(\mathbf{x})^2) = \varrho_i'$, say, for all $\mathbf{x} \in T_i$. In practice this assumption only holds approximately and so $\varrho_i'$ must be estimated from $\varrho'(E_{\mathrm{app}}(\mathbf{x})^2)$, for example by setting it to be the mean value computed over the triangle [1]. Equation (19) can then be rearranged to:

$$H_\rho = \sum_{i=1}^{K} \varrho_i' \sum_{\mathbf{x} \in T_i} \mathbf{SD}_{\mathrm{ic}}^{\mathrm{T}}(\mathbf{x}) \, \mathbf{SD}_{\mathrm{ic}}(\mathbf{x}). \quad (20)$$

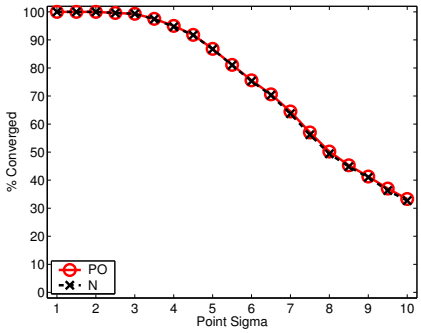



Figure 8: Average frequency of convergence for the project-out and normalization algorithms for 10 appearance images $A_i$. The two algorithms perform identically, showing empirically that they are equivalent. For more results see [2].

The internal part of this expression does not depend on the robust function $\varrho'$ and so is constant across iterations. Denote:

$$H_\rho^i = \sum_{\mathbf{x} \in T_i} \mathbf{SD}_{\mathrm{ic}}^{\mathrm{T}}(\mathbf{x}) \, \mathbf{SD}_{\mathrm{ic}}(\mathbf{x}). \quad (21)$$

The Hessian $H_\rho^i$ is the Hessian for the triangle $T_i$ and can be precomputed. Equation (20) then simplifies to:

$$H_\rho = \sum_{i=1}^{K} \varrho_i' \cdot H_\rho^i. \quad (22)$$

Although this Hessian does vary from iteration to iteration, the cost of computing it is minimal. The same spatial coherence approximation can be made for the appearance Hessian of Equation (18). The efficient robust normalization inverse compositional algorithm is summarized in Figure 7.

## 3.6 Quantitative Comparison

We first compare the performance of the various non-robust and robust fitting algorithms described earlier on synthetic data. In these experiments, we restrict W to be a global affine warp because it is far easier to generate the large number of synthetic test cases which we use. We are only interested in the relative performance of the algorithms and the relative performance should be the same whatever the choice of W. We empirically show in Section 3.6.1 the equivalence of the project-out and normalization inverse compositional algorithms. In Section 3.6.2 we evaluate the different robust fitting algorithms. We show that the approximation proposed in [10] performs far worse than the robust normalization algorithm and that the spatial coherence approximation to the robust normalization algorithm does not significantly reduce the performance.

### 3.6.1 Project-out vs. Normalization

We start with a 225x150 pixel face image $I(\mathbf{x})$ and manually select a 100x100 pixel template $T(\mathbf{x})$ in the center of the face. We then add the appearance variation

Table 1: Fitting speed comparison on a 3GHz Pentium 4 in milliseconds. We measure the average fitting speed per frame of the project-out (PO), robust normalization (RN) and efficient robust normalization (ERN) algorithms over an image sequence of 457 frames. These results are for an AAM with 11 shape parameters, 20 appearance parameters, and 9981 color pixels.

| | PO | RN | ERN |
|---|---|---|---|
| Matlab | 27 ms | 1280 ms | 129 ms |
| C | 4.3 ms | 203.9 ms (est.) | 20.5 ms (est.) |

$\sum_{i=1}^{m} \lambda_i A_i(\mathbf{x})$ to $I(\mathbf{x})$. In the first experiment we randomly select $m = 10$ sub-images of a large image of a natural scene. The images are orthonormalized and used as appearance images $A_i(\mathbf{x})$. The appearance parameters $\lambda_i$ are set to 0.11. We then randomly generate affine warps $\mathbf{W}(\mathbf{x};\mathbf{p})$ in the following manner. We selected 3 canonical points in the template. We then randomly perturb these points with additive white Gaussian noise of a certain variance and fit for the affine warp parameters $\mathbf{p}$ that these 3 perturbed points define. We then warp $I(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x})$ with the affine warp $\mathbf{W}(\mathbf{x};\mathbf{p})$ and run the different algorithms starting from the identity warp. Where appropriate, the appearance parameters are initialized to 0. In Figure 8 we show the average frequency of convergence over 1000 randomly generated inputs for the project-out and normalization inverse compositional algorithm. The two algorithms perform identically for all point sigma values, showing empirically that they are equivalent.

#### 3.6.2 Robust Fitting Algorithms

Using the same image $I(\mathbf{x})$ and template $T(\mathbf{x})$ as in the previous section we randomly occlude a sub-region of $I(\mathbf{x})$ with another image (a sub-image of a natural scene) to evaluate the robust fitting algorithms. The occluding sub-region occupies 30% of the size of the template $T(\mathbf{x})$. We add one appearance image $A_1$ to $I(\mathbf{x})$ with $\lambda_1 = 0.35$. Figure 9 plots the frequency of convergence for the different robust fitting algorithms, again averaged over 1000 randomly generated inputs. The robust project-out algorithm (described in Section 3.2) and the robust normalization algorithm (introduced in Section 3.4) perform identically, showing empirically their equivalence as was already demonstrated in the last section for the non-robust case. The efficient robust normalization algorithm (described in Section 3.5) trails the robust normalization algorithm only slightly in performance, therefore justifying its use. Finally the robust project-out algorithm with Hager-Belhumeur approximation (no re-orthonormalization of the appearance images and use of the *H*-Algorithm [7] to keep the Hessian constant) performs far worse than the other algorithms.

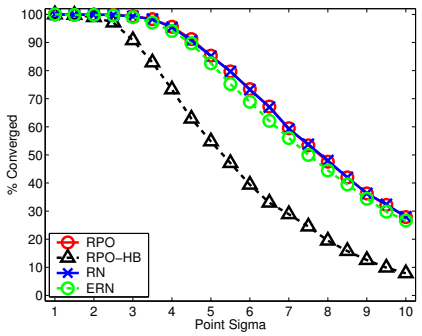



Figure 9: Average frequency of convergence for the robust fitting algorithms in the case of 30% occlusion. The robust project-out (RPO) and the robust normalization (RN) algorithm again perform identically. The efficient robust normalization algorithm (ERN) only performs slightly worse than the non-efficient variants. The robust project-out algorithm with Hager-Belhumeur approximation (RPO-HB) performs far worse then any of the other algorithms. For more results with different sizes of occlusion see [2].

### 3.7 Efficiency Comparison

We now evaluate the efficiency of the robust normalization algorithm. Table 1 compares the average fitting speed per frame of the project-out algorithm (PO) with the robust normalization (RN) and efficient robust normalization (ERN) algorithms. We implemented all three algorithms in Matlab and measured the fitting speed over an image sequence of 457 frames. The Matlab implementation of the efficient robust normalization algorithm provides a 10-fold speed up over the non-efficient robust normalization algorithm. We previously measured the fitting speed of an implementation of the project-out algorithm in C at 230 frames per second [13]. Due to the structure of the algorithms it is reasonable to assume that we can achieve similar speed up rates between Matlab and C implementations of the robust normalization and efficient robust normalization algorithms. Based on this estimate the efficient robust normalization algorithm would run at 48.8 frames per second.

### 3.8 Qualitative Evaluation

Figure 10 shows example frames from tracking experiments comparing the fitted meshes of the (non-robust) project-out and the efficient robust normalization algorithm. The three image sequences show different kinds of occlusion. In the first sequence (Figure 10(a), movie `box.mpg` ) a black box is moved in front of the face. In the second sequence (Figure 10(b), movie `hand.mpg`) the hand covers the chin while the head rotates. Finally in the third sequence (Figure 10(c), movie `rotate.mpg`) the face rotates from frontal to full left profile and back to frontal again. In all three cases the efficient robust normalization algorithm accurately tracks the face while the (non-robust) project-out algorithm fails. The AAM used in all cases was trained on images that do not appear in the test sequences. It is im-

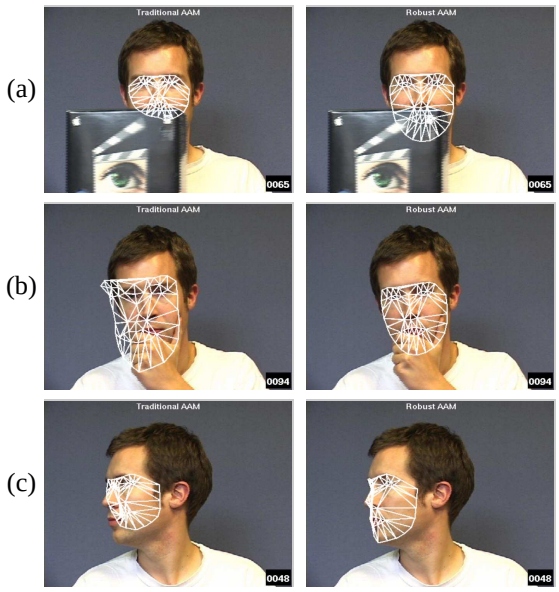



Figure 10: Comparison of using the (non-robust) project-out and the efficient robust normalization algorithm on image sequences with occlusion. (a) A black box is moved in front of the face (see `box.mpg`). (b) The chin is covered by the hand while the face rotates (`hand.mpg`). (c) The face rotates from frontal to full left profile and back to frontal again (movie `rotate.mpg`). In all cases the efficient robust normalization algorithm accurately tracks the face while the project-out algorithm fails.

portant to note that we achieve accurate tracking of a face across wide pose changes with a *single* model. In [6] the same task was achieved using multiple AAMs and a heuristic for switching between them. One major advantage of using only a single model is that the model parameters have the same "meaning" for all poses.

# 4 Discussion

In this paper we proposed algorithms to construct and robustly fit AAMs with occlusion. In comparison to previously introduced robust fitting and tracking algorithms [8, 10, 14] which make use of ad hoc approximations, we analytically derived a gradient descent algorithm, the robust normalization algorithm. We empirically showed that the algorithm introduced in [10] performs far worse than the robust normalization algorithm. Furthermore, we proposed an efficient approximation to the robust normalization algorithm which can run in real-time at approximately 50 frames-per-second. We finally demonstrated successful tracking using our algorithm on videos with varying degrees and types of occlusion.

# 5 Acknowledgments

The research described in this paper was supported by ONR contract N00014-00-1-0915 and in part by U.S. Department of Defense contract N41756-03-C4024.

# References

[1] S. Baker, R. Gross, T. Ishikawa, and I. Matthews. Lucas-Kanade 20 years on: A unifying framework: Part 2. Technical Report CMU-RI-TR-03-01, Carnegie Mellon University Robotics Institute, 2003.

[2] S. Baker, R. Gross, and I. Matthews. Lucas-Kanade 20 years on: A unifying framework: Part 3. Technical Report CMU-RI-TR-03-35, Carnegie Mellon University Robotics Institute, 2003.

[3] M. Black and A. Jepson. Eigen-tracking: Robust matching and tracking of articulated objects using a view-based representation. *IJCV*, 36(2):101–130, 1998.

[4] V. Blanz and T. Vetter. A morphable model for the synthesis of 3D faces. In *Proc. of SIGGRAPH*, 1999.

[5] T. Cootes, G. Edwards, and C. Taylor. Active appearance models. *IEEE PAMI*, 23(6), 2001.

[6] T. Cootes, G. Wheeler, K. Walker, and C. Taylor. View-based active appearance models. *Image and Vision Computing*, 20, 2002.

[7] R. Dutter and P.J. Huber. Numerical methods for the nonlinear robust regression problem. *Journal of Statistical and Computational Simulation*, 13, 1981.

[8] G.J. Edwards, T.J. Cootes, and C.J. Taylor. Advances in active appearance models. In *International Conference on Computer Vision*, pages 137–142, 1999.

[9] K. Fukunaga. *Introduction to statistical pattern recognition*. Academic Press, 1990.

[10] G. Hager and P. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE PAMI*, 20(10), 1998.

[11] P.J. Huber. *Robust Statistics*. Wiley & Sons, 1981.

[12] Dryden. I.L. and K.V. Mardia. *Statistical Shape Analysis*. Wiley & Sons, 1998.

[13] I. Matthews and S. Baker. Active Appearance Models revisited. *International Journal of Computer Vision*, 2004.

[14] S. Sclaroff and J. Isidoro. Active blobs. In *Proc. ICCV*, pages 1146–1153, 1998.

[15] H. Shum, K. Ikeuchi, and R. Reddy. Principal component analysis with missing data and its application to polyhedral object modeling. *IEEE PAMI*, 17(9), 1995.

[16] F. de la Torre and M. Black. A framework for robust subspace learning. *IJCV*, 54(1), 2003.