Appeared in *Image and Vision Computing*, 24 (2006), 593-604

Active Appearance Models with Occlusion

Ralph Gross, Iain Matthews, and Simon Baker

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

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 head pose and gaze 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 track faces efficiently 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 and types of occlusions.

1 Introduction

Active Appearance Models (AAMs) [6] (and the closely related concepts of Active Blobs [16] and Morphable Models [5]) are generative parametric models commonly used to track faces non-rigidly 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 [6]. AAMs are then fit frame-by-frame to input videos to track the face through the video [6, 15]. The best fit model parameters are then used in whatever the chosen application is. A variety of video applications are possible, including dynamic head pose and gaze 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. Other causes of occlusion include sunglasses or any 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 [17, 18] to compute the shape and appearance variation. We compare models computed from unoccluded and occluded data and empirically show a high degree of similarity for up to 45% occlusion of the face region.

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 [4] requiring minutes per frame. Efficient robust fitting algorithms have been proposed, for example by Hager and Belhumeur in [13]. However, as we will show in Section 4, these algorithms make approximations that adversely affect their robustness.

We begin Section 3 by first describing our previously introduced (efficient, but non-robust) project-out inverse compositional AAM fitting algorithm [15]. 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 exten-

sion to the normalization algorithm and show in Section 3.5 how to implement the robust normalization algorithm efficiently. For completeness in Section 3.6 we describe the robust Gauss-Newton inverse compositional algorithm applied simultaneously to the shape and appearance variation. While being considerably slower this algorithm performs better than the other robust algorithms and should therefore be considered for applications with less stringent real-time demands. See Figure 16 for an overview of the algorithms discussed in this paper.

In Section 4 we quantitativly evaluate all of the fitting algorithms on synthetic data. In particular we show that the efficient robust normalization algorithm outperforms the Hager-Belhumeur algorithm [13]. We furthermore 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 at around an estimated 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 faces to be modeled with the location of all of the *visible* mesh vertices in each of the images marked. Due to self-occlusion, e.g. when generating a model across large changes in pose, or due to occlusion by an object, only a subset of the vertices may be visible in any given training image. In the following we use the definition of an *independent AAM* which omits the combined PCA across shape and appearance [15].

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:

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

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 $\mathbf{s}_i$:

$$\mathbf{s} \;=\; \mathbf{s}_0 + \sum_{i=1}^{n} p_i \mathbf{s}_i \tag{2}$$

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 $\mathbf{s}_0$ with Occlusion

In traditional AAMs [6, 15] 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 [8]. 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. The *Procrustes* algorithm with occlusion is then:

1. Initialize the base mesh $\mathbf{s}_0$ to be the visible vertices of the mesh $\mathbf{s}$ in 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 2D similarity transform (rotation, translation, and scale) 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 $\mathbf{s}_i$ with Occlusion

In traditional AAMs [6, 15] 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 [6]. The mean shape (i.e., the base mesh $\mathbf{s}_0$) is subtracted from each shape vector. Principal Components Analysis [11] 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 [17, 18] 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 [6] we

retain enough shape modes to explain 95% of the observed variation in the training set.

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 $A_i(\mathbf{x})$:

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

where the coefficients λ_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 with Occlusion

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 $\mathbf{s}_0$ [6]. 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 [17, 18] to compute the appearance vectors A_i. The appearance vectors A_i are then set to be the orthonormalized eigenvectors with the largest eigenvalues. As in the computation of the shape model, we retain enough appearance modes to explain 95% of the observed variation in the training set.

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. Using artificially occluded data in this way allows for a more systematic evaluation of how the algorithms perform with varying degrees of occlusion. In Section 2.3.4 we include experiments with natural occlusion. In total

(a) (b) (c)

Figure 1: Artificially occluded data. (a) Original images with all mesh vertices $\mathbf{s}$ visible. (b) Images with 10% of the *face region* occluded. (c) Images with 50% of the *face region* occluded. Only non-occluded vertices are used in the AAM construction.

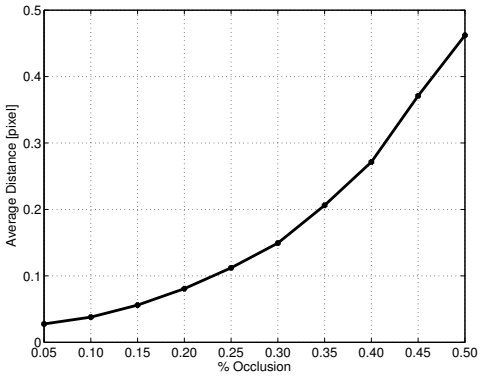
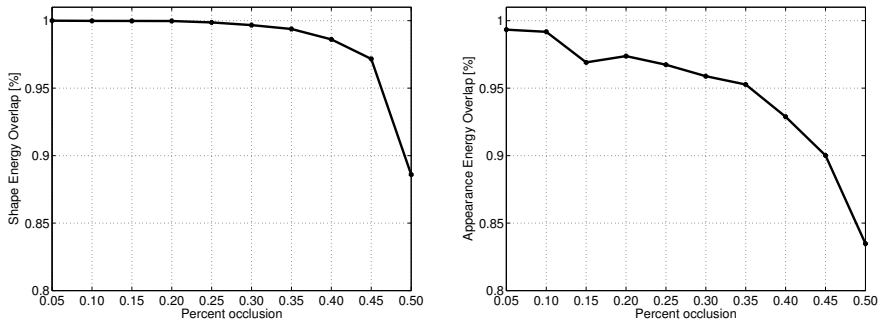


Figure 2: Base mesh distance. The graph shows average pixel distances between base meshes $\mathbf{s_0}$ computed from unoccluded and occluded training data. While the pixel distance increases for higher levels of occlusion it stays below 0.5 pixels even for the maximal occlusion of 50%.

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



(a) Shape energy overlap $\mathbf{SE}$ (b) Appearance energy overlap $\mathbf{AE}$

Figure 3: Comparison of the AAM model components shape variation and appearance variation computed from unoccluded and occluded data. (a) Shape energy overlap $\mathbf{SE}$. (b) Appearance energy overlap $\mathbf{AE}$. For both components a high degree of similarity is evident. At around 50% occlusion, however, the performance drops off rapidly.

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 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 4 shows the base mesh $\mathbf{s_0}$ and shape variations $\mathbf{s_1} - \mathbf{s_3}$ computed from unoccluded data and data containing 50% occlusion. The resulting shape modes are very similar. See the accompanying movie `shape_modes.mpg` which shows the shape modes computed from unoccluded and occluded data[1]. In order to quantify the similarity of the shape modes we measure the shape energy overlap $\mathbf{SE}$ between shape variations $\mathbf{s}_i^u$ and $\mathbf{s}_i^o$ computed from unoccluded and occluded data, respectively. The energy overlap is the fraction of one shape subspace contained

[1]Movies are available at http://www.ri.cmu.edu/projects/project_562.html

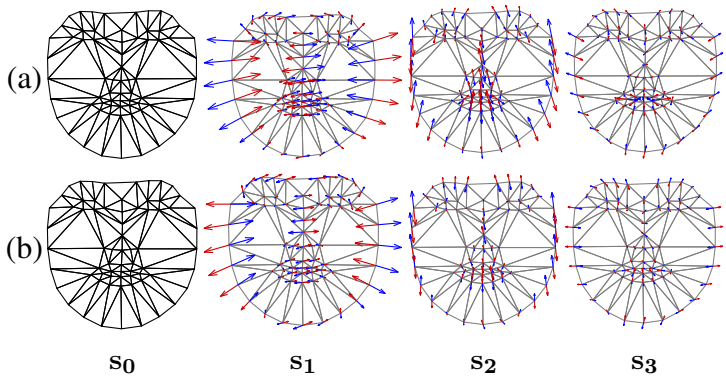


Figure 4: Mean shape $\mathbf{s_0}$ and shape variations $\mathbf{s_1} - \mathbf{s_3}$ overlaid on the base mesh. (a) Shape images computed from unoccluded data. (b) Shape images computed from data with 50% occlusion. The resulting shape modes are very similar.

in the other and is computed by projecting all of the occluded shape vectors into the unoccluded shape subspace and computing the fraction of the energy retained. The exact definition is as follows:

$$\mathbf{SE} = \frac{1}{n} \sum_i \sqrt{\sum_j ((s_i^u)^T s_j^o)^2} \tag{4}$$

for $i, j = 0, \ldots, n$, where n refers to the number of shape modes. $\mathbf{SE}$ ranges in value from 0 to 1. Figure 3(a) plots $\mathbf{SE}$ 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 5 shows the mean appearance $\mathbf{A_0}$ and appearance variations $\mathbf{A_1} - \mathbf{A_3}$ computed from unoccluded data and data containing 50% occlusion. The resulting mean appearance images look very similar. The accompanying movie `app_modes.mpg` which shows the appearance modes computed from unoccluded and occluded data. Since it is hard to interpret the appearance eigenvectors we again quantify the similarity of the appearance models with the appearance energy overlap $\mathbf{AE}$ which is defined analogously to $\mathbf{SE}$ (see Eqn. (4)).

$$\mathbf{AE} = \frac{1}{n} \sum_i \sqrt{\sum_j ((A_i^u)^T A_j^o)^2} \tag{5}$$

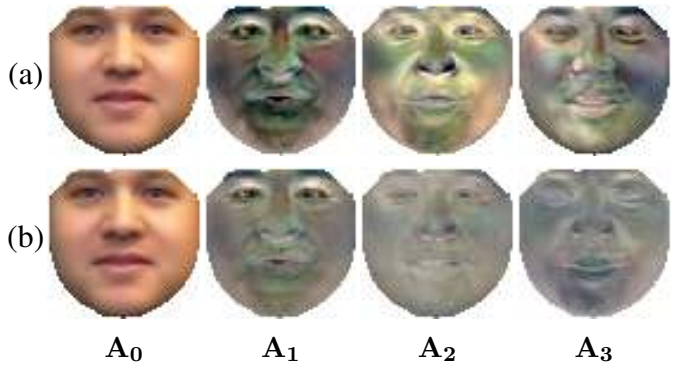


Figure 5: Mean appearance $\mathbf{A_0}$ and appearance variations $\mathbf{A_1} - \mathbf{A_3}$. (a) Appearance images computed from unoccluded data. (b) Appearance images computed from data with 50% occlusion.

Figure 3(b) plots $\mathbf{AE}$ values for different occlusion sizes. The $\mathbf{AE}$ values decline slightly faster than the $\mathbf{SE}$ 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 right profile views) and occlusion by an object to build the AAM. See Figure 6 for example images. In the training set on average 18% of the feature points are occluded. The AAM successfully tracks a face in an independent test sequence. Figure 7 shows example frames with the fitted mesh overlaid on the input image. The accompanying movie `fit.mpg` includes the full sequence of 457 frames.

2.3.5 Summary

In this section we showed how to construct an AAM from training data with occlusion. We empirically showed that AAMs computed from data containing up to 45% occlusion are very similar to AAMs computed from unoccluded data. We furthermore demonstrated good tracking results using an AAM constructed from training data containing both self-occlusion and occlusion by an object.

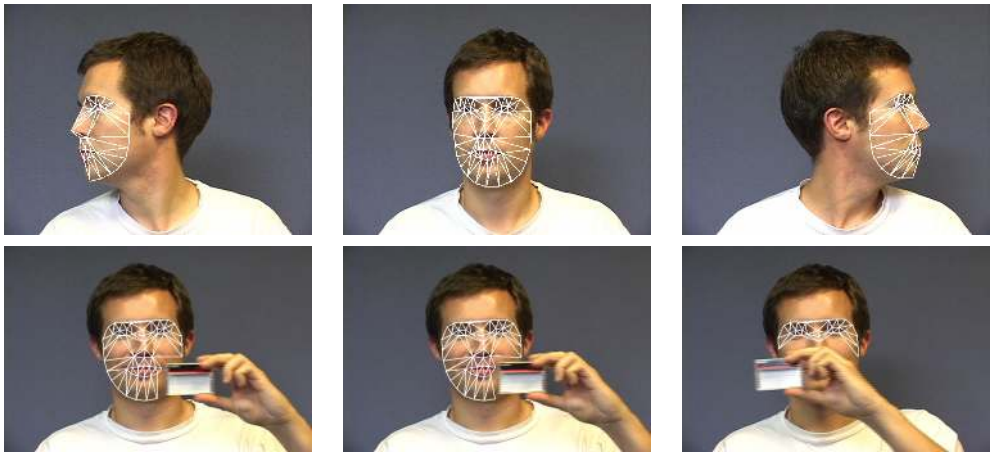


Figure 6: Training images with and without occlusion. We show 6 of the 120 hand-marked images used in the training of the AAM for the tracking task of Figure 7.

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, the Project-Out Algorithm [15] 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, the Normalization Algorithm, which can be implemented efficiently and empirically demonstrate its ability to track occluded faces, robustly and in real-time. For completeness we then describe the inefficient, but better performing robust Gauss-Netwon inverse compositional algorithm applied simultaneously to the shape and appearance parameters.

3.1 Background: Efficient Project-Out Algorithm

Fitting a AAM is usually formulated [15] 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 $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$:

$$\sum_{\mathbf{x} \in \mathbf{s}_0} \left[A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2 \tag{6}$$

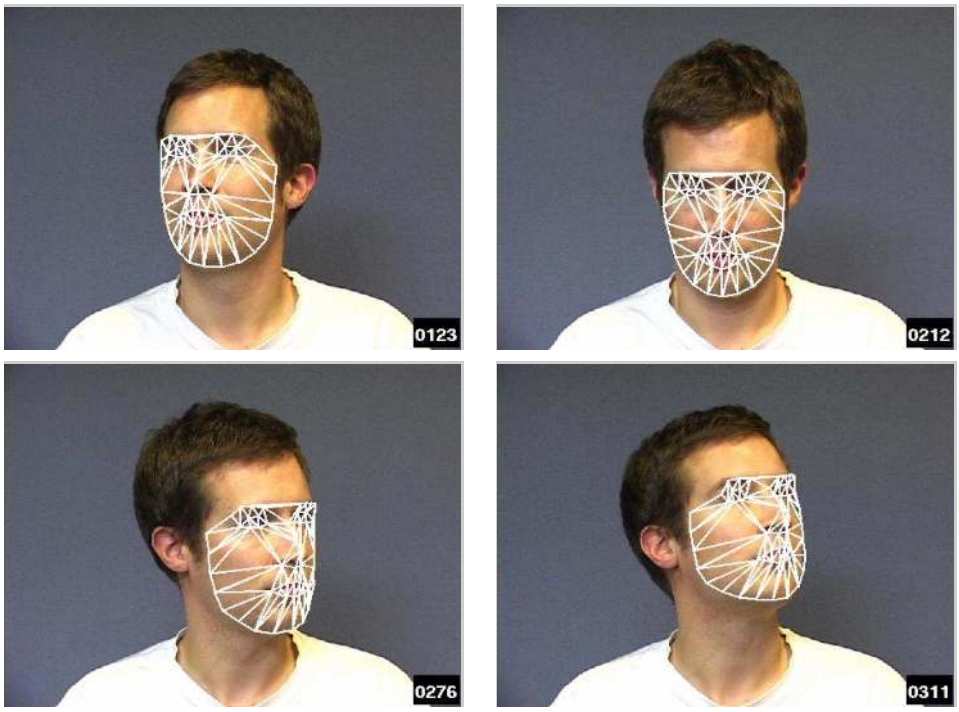


Figure 7: Example frames of a test sequence showing accurate tracking with an AAM constructed with occlusion. See the accompanying movie `fit.mpg` for the full sequence of 457 frames.

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 [15] we showed how to include this warp into $\mathbf{W}$. The goal of AAM fitting is to minimize the expression in Equation (6) simultaneously with respect to the shape $\mathbf{p}$ and appearance $\boldsymbol{\lambda}$ parameters. The "project-out" inverse compositional algorithm [3] and its extension to 2D AAMs was proposed in [15]. See Figure 8 for a summary. The algorithm performs the non-linear optimization of Equation 6 in two steps (similar to Hager and Belhumeur [13]). 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

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. (7))

(P4) Project out appearance from $\mathbf{SD}_{\mathrm{ic}}(\mathbf{x})$ (Eqn. (8))

(P5) Compute the Hessian matrix H_{po} (Eqn. (10))

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 $\sum_{\mathbf{x}} \mathbf{SD}^{\mathrm{T}}_{\mathrm{po}}(\mathbf{x}) E(\mathbf{x})$

(I4) Compute $\Delta \mathbf{p} = -H^{-1}_{\mathrm{po}} \sum_{\mathbf{x}} \mathbf{SD}^{\mathrm{T}}_{\mathrm{po}}(\mathbf{x}) E(\mathbf{x})$

(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 8: The project-out inverse compositional algorithm [15].

variation from the *steepest-descent images*:

$$\mathbf{SD}_{\mathrm{ic}}(\mathbf{x}) = \boldsymbol{\nabla} A_0 \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \tag{7}$$

by computing:

$$\mathbf{SD}_{\mathrm{po}}(\mathbf{x}) = \mathbf{SD}_{\mathrm{ic}} - \sum_{i=1}^{m} \left[\sum_{\mathbf{x} \in \mathbf{s}_0} A_i(\mathbf{x}) \mathbf{SD}_{\mathrm{ic}}(\mathbf{x}) \right] A_i(\mathbf{x}). \tag{8}$$

Equation (8) 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:

$$\Delta \mathbf{p} \;=\; -H_{\mathrm{po}}^{-1} \sum_{\mathbf{x} \in \mathbf{s}_0} \mathbf{SD}_{\mathrm{po}}(\mathbf{x}) [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - A_0(\mathbf{x})] \tag{9}$$

using the *Project-Out Hessian*:

$$H_{\mathrm{po}} \;=\; \sum_{\mathbf{x} \in \mathbf{s}_0} \mathbf{SD}_{\mathrm{po}}(\mathbf{x})^{\mathrm{T}} \mathbf{SD}_{\mathrm{po}}(\mathbf{x}). \tag{10}$$

The incremental warp $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ is then *inverted* and *composed* with the current estimate to give the new estimate $\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$. In the second step, the appearance parameters $\boldsymbol{\lambda}$ can then be computed as:

$$\lambda_i \;=\; \sum_{\mathbf{x} \in \mathbf{s}_0} A_i(\mathbf{x}) \cdot [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - A_0(\mathbf{x})] \,. \tag{11}$$

If there are n shape parameters, m appearance parameters, and N pixels in the base appearance A_0, the pre-computation 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 until convergence or for 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 [15].

3.2 Robust Fitting: Inefficient Algorithm

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 [4, 13, 14]. The goal of *robustly* fitting a AAM is then to minimize

$$\sum_{\mathbf{x} \in \mathbf{s}_0} \varrho \left(\left[A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2 ; \boldsymbol{\sigma} \right) \tag{12}$$

with respect to the shape $\mathbf{p}$ and appearance $\boldsymbol{\lambda}$ parameters where $\varrho(t;\boldsymbol{\sigma})$ is a symmetric *robust error function* [14] 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 9) and the Hessian $H_{\rm po}$ (Equation 10) have to be weighted by the error function $\varrho'(E_{\rm app}(\mathbf{x})^2)$, where:

$$E_{\rm app}(\mathbf{x}) = I(\mathbf{W}(\mathbf{x};\mathbf{p})) - \left[A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x})\right] \tag{13}$$

Equation (9) then becomes:

$$\Delta\mathbf{p} \;=\; -H_{\rho}^{-1} \sum_{\mathbf{x}\in\mathbf{s}_0} \varrho'(E_{\rm app}(\mathbf{x})^2)\mathbf{SD}_{\rm po}(\mathbf{x})E_{\rm app}(\mathbf{x}) \tag{14}$$

with:

$$H_{\rho} \;=\; \sum_{\mathbf{x}\in\mathbf{s}_0} \varrho'(E_{\rm app}(\mathbf{x})^2)\mathbf{SD}_{\rm po}(\mathbf{x})^{\rm T}\mathbf{SD}_{\rm po}(\mathbf{x}). \tag{15}$$

The steepest descent images $\mathbf{SD}_{\rm po}$ also have to be re-computed using Equation (8) because the appearance images are no longer orthonormal. The appearance images A_i must be re-orthonormalized with respect to the new inner product:

$$\sum_{\mathbf{x}} \varrho'\left(E_{\rm app}(\mathbf{x})^2\right)\, A_i(\mathbf{x})\, A_j(\mathbf{x}) \;=\; \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{16}$$

Steps (P3)-(P5) in Figure 8 can therefore no longer be pre-computed and have to be moved inside the iteration. As a result the robust project-out inverse compositional algorithm is very inefficient. See [2] for more 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 [13], where the *H*-Algorithm [9] is used to keep the Hessian constant to yield an efficient algorithm. As we will show in Section 4 this approximation while fast, leads to poor performance.

3.3 Project-out vs. Normalization

We now describe a slightly different algorithm to minimize the expression in Equation (6), 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 (6) is to project out the appearance images A_i from the *single* error image $E_{\rm app}$ rather than the large number of steepest descent images $\mathbf{SD}_{\rm ic}$. 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:

$$\begin{aligned} \lambda_i \;&=\; \sum_{\mathbf{x}} A_i(\mathbf{x})\, E(\mathbf{x}) \;\text{ for } i = 1, \ldots, m \\ E_{\text{app}}(\mathbf{x}) \;&\leftarrow\; E(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i\, A_i(\mathbf{x}). \end{aligned} \tag{17}$$

As indicated, in the process of normalizing E_{app} in this way the appearance parameters λ_i are estimated. In comparison to the project-out algorithm in Figure 8 steps (P4) and (A1) are removed and the normalization step of Equation (17) 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 4.

3.4 Robust Normalization Algorithm

The goal of the normalization step in Equation (17) is to make the component of the error image in the direction A_i to be zero, whilst computing λ_i at the same time. We now reformulate this step 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:

$$\sum_{\mathbf{x}} \varrho'\left(E_{\text{app}}(\mathbf{x})^2 \right) \left[E_{\text{app}}(\mathbf{x}) - \sum_{i=1}^{m} \Delta\lambda_i A_i(\mathbf{x}) \right]^2. \tag{18}$$

The least squares minimum of this expression is:

$$\Delta\boldsymbol{\lambda} \;=\; H_{\mathbf{A}}^{-1} \sum_{\mathbf{x}} \varrho'\left(E_{\text{app}}(\mathbf{x})^2 \right) \mathbf{A}^{\mathrm{T}}(\mathbf{x}) E_{\text{app}}(\mathbf{x}) \tag{19}$$

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

$$H_{\mathbf{A}} \;=\; \sum_{\mathbf{x}} \varrho'\left(E_{\text{app}}(\mathbf{x})^2 \right) \mathbf{A}(\mathbf{x})^{\mathbf{T}} \mathbf{A}(\mathbf{x}). \tag{20}$$

The steepest descent parameter updates and the Hessian are computed as in Equations (14) and (15). Note that we avoid re-orthonormalization of the appearance images A_i in every iteration as is required in the robust algorithm of Section 3.2.

3.5 Efficient Robust Fitting

Due to the computation of the appearance Hessian H_A and the Hessian H_ρ 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^{th} triangle. Equation (15) can then be rewritten:

$$H_\rho \;=\; \sum_{i=1}^{K} \sum_{\mathbf{x} \in T_i} \varrho' \left(E_{\text{app}}(\mathbf{x})^2 \right) \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x}) \, \mathbf{SD}_{\text{ic}}(\mathbf{x}). \tag{21}$$

Based on the spatial coherence of the outliers [1], assume that $\varrho'(E_{\text{app}}(\mathbf{x})^2)$ is constant in each triangle; i.e. assume $\varrho'(E_{\text{app}}(\mathbf{x})^2) = \varrho'_i$, say, for all $\mathbf{x} \in T_i$. In practice this assumption only holds approximately and so ϱ'_i must be estimated from $\varrho'(E_{\text{app}}(\mathbf{x})^2)$, for example by setting it to be the mean value computed over the triangle [1]. Equation (21) can then be rearranged to:

$$H_\rho \;=\; \sum_{i=1}^{K} \varrho'_i \sum_{\mathbf{x} \in T_i} \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x}) \, \mathbf{SD}_{\text{ic}}(\mathbf{x}). \tag{22}$$

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

$$H_\rho^i \;=\; \sum_{\mathbf{x} \in T_i} \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x}) \, \mathbf{SD}_{\text{ic}}(\mathbf{x}). \tag{23}$$

The Hessian H_ρ^i is the Hessian for the triangle T_i and can be precomputed. Equation (22) then simplifies to:

$$H_\rho \;=\; \sum_{i=1}^{K} \varrho'_i \cdot H_\rho^i. \tag{24}$$

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 (20). The efficient robust normalization inverse compositional algorithm is summarized in Figure 9.

3.6 Robust Simultaneous Fitting Algorithm

In this paper we have described a variety of efficient gradient descent algorithms. All of these algorithms are approximations to the simultaneous Gauss-Newton

Efficient Robust Normalization 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. (7))

(P4) Compute Hessian H_ρ^i for each triangle (Eqn. (23))

(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. (13))

(I3) Compute $H_A = \sum_i \varrho_i' \cdot H_A^i$

(I4) Compute $\Delta \boldsymbol{\lambda}$ and update λ and $E_{\mathrm{app}}(\mathbf{x})$ (Eqn. (19))

(I5) Compute the Hessian H_ρ and invert it (Eqn. (24))

(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 9: The efficient robust normalization inverse compositional image alignment algorithm.

inverse compositional gradient descent algorithm over both the shape and appearance parameters. In this section we describe the full robust simultaneous inverse compositional algorithm. The algorithm operates by iteratively minimizing:

$$\sum_{\mathbf{x}} \varrho \left(\left[A_0(\mathbf{W}(\mathbf{x}; \Delta \mathbf{p})) + \sum_{i=1}^{m} (\lambda_i + \Delta \lambda_i) A_i(\mathbf{W}(\mathbf{x}; \Delta \mathbf{p})) - I(\mathbf{W}(\mathbf{x}; \mathbf{p})) \right]^2 \right) \tag{25}$$

simultaneously with respect to $\Delta \mathbf{p}$ and $\Delta \boldsymbol{\lambda} = (\Delta \lambda_1, \ldots, \Delta \lambda_m)^{\mathrm{T}}$, and then updating the warp $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta \mathbf{p})^{-1}$ and the appearance parameters $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \Delta \boldsymbol{\lambda}$.

To simplify the notation, denote:

$$\mathbf{q} = \begin{pmatrix} \mathbf{p} \\ \boldsymbol{\lambda} \end{pmatrix} \quad \text{and similarly} \quad \Delta\mathbf{q} = \begin{pmatrix} \Delta\mathbf{p} \\ \Delta\boldsymbol{\lambda} \end{pmatrix}; \tag{26}$$

i.e. $\mathbf{q}$ is an $n+m$ dimensional column vector containing the warp parameters $\mathbf{p}$ concatenated with the appearance parameters $\boldsymbol{\lambda}$. Denote the $n+m$ dimensional steepest-descent images as follows:

$$\mathbf{SD}_{\text{sim}}(\mathbf{x}) = \left(\boldsymbol{\nabla} A \frac{\partial \mathbf{W}}{\partial p_1}, \ldots, \boldsymbol{\nabla} A \frac{\partial \mathbf{W}}{\partial p_n}, A_1(\mathbf{x}), \ldots, A_m(\mathbf{x}) \right) \tag{27}$$

where $\boldsymbol{\nabla}\mathbf{A}$ is defined as

$$\boldsymbol{\nabla}\mathbf{A} = \boldsymbol{\nabla} A_0 + \sum_{i=1}^{m} \lambda_i \boldsymbol{\nabla} A_i. \tag{28}$$

We can then compute the parameter update $\Delta\mathbf{q}$ as

$$\Delta\mathbf{q} = -H_{\text{sim},\varrho}^{-1} \sum_{\mathbf{x}} \varrho' \left(E_{\text{app}}(\mathbf{x})^2 \right) \mathbf{SD}_{\text{sim}}^{\text{T}}(\mathbf{x}) E_{\text{app}}(\mathbf{x}) \tag{29}$$

where:

$$H_{\text{sim},\varrho} = \sum_{\mathbf{x}} \varrho' \left(E_{\text{app}}(\mathbf{x})^2 \right) \mathbf{SD}_{\text{sim}}^{\text{T}}(\mathbf{x}) \, \mathbf{SD}_{\text{sim}}(\mathbf{x}) \tag{30}$$

and E_{app} is defined as in Equation (13). See [2] for more details. Since the steepest descent images $\mathbf{SD}_{\text{sim}}$ depend on the appearance parameters $\boldsymbol{\lambda}$ through Equation (28) they have to be re-computed in every iteration. The algorithm is therefore inefficient. The robust simultaneous algorithm is summarized in Figure 10.

4 Evaluation

We evaluate all of the fitting algorithms described in Section 3 on synthetic data. We then demonstrate successful face tracking using the robust normalization algorithm on a number of image sequences containing occlusion.

4.1 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

Robust Simultaneous Inverse Compositional Algorithm

Pre-Computation:

(P1) Evaluate the gradients of ∇A_0 and ∇A_i for $i = 1, \ldots, m$

(P2) Evaluate the Jacobian of the warp $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x}; \mathbf{0})$

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 $E_{\text{app}}(\mathbf{x})$ (Eqn. (13))

(I3) Compute the steepest descent images $\mathbf{SD}_{\text{sim}}(\mathbf{x})$ (Eqn. (27))

(I4) Compute the Hessian $H_{\text{sim},\varrho}$ using Equation (30) and invert it

(I5) Compute $\sum_{\mathbf{x}} \varrho' \left(E_{\text{app}}(\mathbf{x})^2\right) \mathbf{SD}_{\text{sim}}^{\text{T}}(\mathbf{x}) E_{\text{app}}(\mathbf{x})$

(I6) Compute $\Delta\mathbf{q} = -H_{\text{sim},\varrho}^{-1} \sum_{\mathbf{x}} \varrho' \left(E_{\text{app}}(\mathbf{x})^2\right) \mathbf{SD}_{\text{sim}}^{\text{T}}(\mathbf{x}) E_{\text{app}}(\mathbf{x})$

(I7) Update $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta\mathbf{p})^{-1}$ and $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \Delta\boldsymbol{\lambda}$

Figure 10: The robust simultaneous algorithm. Because the steepest descent images depend on the appearance parameters, Steps (I3) and (I4) must be performed in every iteration.

W to be a global affine warp because it is far easier to generate a large number of synthetic test cases. 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 4.1.1 the equivalence of the project-out and normalization inverse compositional algorithms. In Section 4.1.2 we evaluate the different robust fitting algorithms. We show that the approximation proposed in [13] 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.

4.1.1 Project-out vs. Normalization

Following the procedure in [3], 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 $\sum_{i=1}^{m} \lambda_i A_i(\mathbf{x})$ to $I(\mathbf{x})$. In the first experiment we

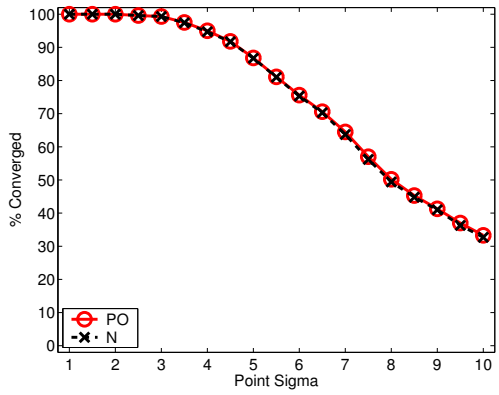


Figure 11: Average frequency of convergence for the project-out (PO) and normalization (N) algorithms for 10 appearance images A_i. The two algorithms perform identically, showing empirically that they are equivalent. For more results see [2].

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 λ_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 used the bottom left corner $(0, 0)$, the bottom right corner $(99, 0)$, and the center top pixel $(49, 99)$ as the canonical points. 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 11 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.

4.1.2 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-regions occupy between 10% and 50% 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 12 plots the frequency of convergence

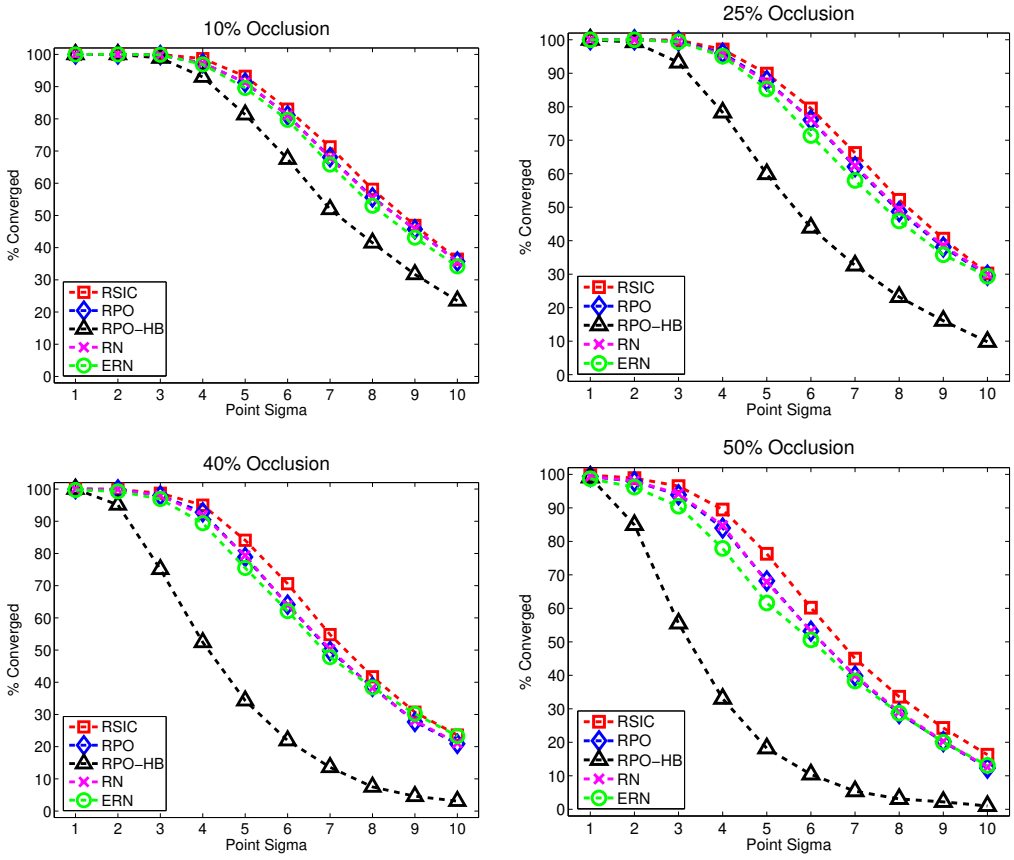


Figure 12: Average frequency of convergence for the robust fitting algorithms for different levels of occlusion. The robust project-out (RPO) and the robust normalization (RN) algorithm again perform identically. The efficient robust normalization algorithm (ERN) only performs slightly worse than the non-efficient variants. The robust project-out algorithm with Hager-Belhumeur approximation (RPO-HB) performs far worse than any of the other algorithms, especially for higher levels of occlusion. Across all four conditions the robust simultaneous algorithm (RSIC) performs best.

for the different robust fitting algorithms for different levels of occlusion, 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 [9] to keep the Hessian constant) performs far worse than the other algorithms, especially for higher levels of occlusion. Across all conditions, the robust simultaneous algorithm performs best.

4.2 Efficiency Comparison

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.)

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 [15]. 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.

Figure 13: Comparison of using the (non-robust) project-out (top row) [15] and the efficient robust normalization algorithm (bottom row) on an image sequence with occlusion by a black box. The project-out algorithm fails to track once the face is covered by the box (top center and top right) and is unable to recover (see `box.mpg`). The efficient robust normalization algorithm accurately tracks the face (bottom row).

4.3 Qualitative Evaluation

Figures 13, 14, and 15 show 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 13, movie `box.mpg`) a black box is moved in front of the face. In the second sequence (Figure 14, movie `hand.mpg`) the hand covers the chin while the head rotates. Finally in the third sequence (Figure 15, 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. Note that in Figure 15 we achieve accurate tracking of a face across wide pose changes with a *single* model. In [7] 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.

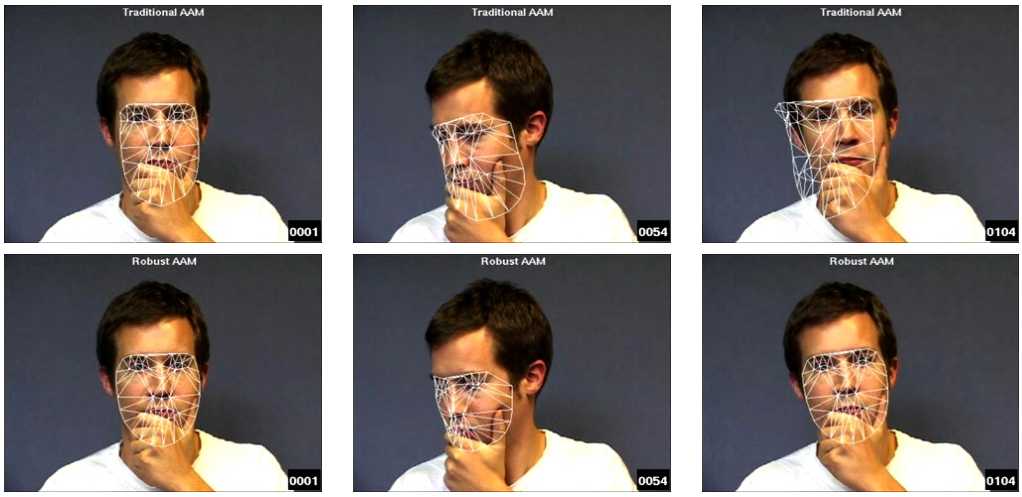


Figure 14: Comparison of using the (non-robust) project-out (top row) [15] and the efficient robust normalization algorithm (bottom row) on an image sequence with occlusion by a hand. The chin is covered by the hand while the face rotates. The project-out algorithm fails to track once the face starts to rotate (top center) and again is unable to recover (see `hand.mpg`). The efficient robust normalization algorithm accurately tracks the face throughout the sequence (bottom row).

5 Discussion

In this paper we proposed algorithms to construct and robustly fit AAMs with occlusion. We empirically showed that AAMs computed from data containing up to 45% occlusion are very similiar to AAMs computed from unoccluded data. In comparison to previously introduced robust fitting and tracking algorithms [10, 13, 16] which make use of ad hoc approximations, we analytically derived a gradient descent algorithm, the robust normalization algorithm. We empirically showed that the Hager-Belhumeur algorithm introduced in [13] 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. See Figure 16 for an overview of all algorithms. We finally demonstrated successful tracking using our algorithm on videos with varying degrees and types of occlusion.

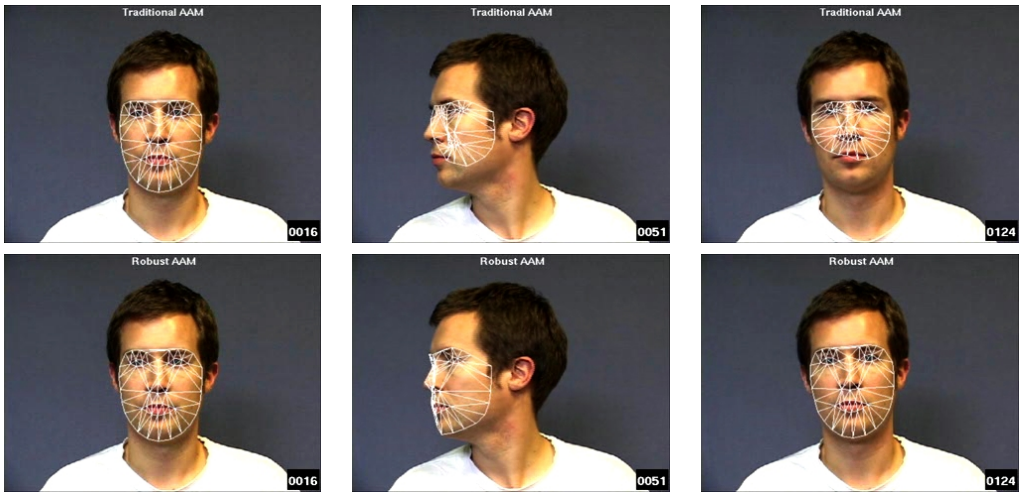


Figure 15: Comparison of using the (non-robust) project-out (top row) and the efficient robust normalization algorithm (bottom row) on an image sequence with self-occlusion. The face rotates from frontal to full left profile and back to frontal again. The project-out algorithm fails to track once the face nears the profile locatation (top center). Again, the efficient robust normalization algorithm accurately tracks the face throughout the entire sequence (see `rotate.mpg`).

6 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. An earlier version of this paper appeared in [12].

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] S. Baker and I. Matthews. Lucas-Kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, February 2004.

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

[5] V. Blanz and T. Vetter. A morphable model for the synthesis of 3D faces. In *Computer Graphics, Annual Conference Series (SIGGRAPH)*, pages 187–194, 1999.

[6] T. Cootes, G. Edwards, and C. Taylor. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):681–685, 2001.

[7] T. Cootes, G. Wheeler, K. Walker, and C. Taylor. View-based active appearance models. *Image and Vision Computing*, 20:657–664, 2002.

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

[9] R. Dutter and P.J. Huber. Numerical methods for the nonlinear robust regression problem. *Journal of Statistical and Computational Simulation*, 13:79–113, 1981.

[10] 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.

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

[12] R. Gross, I. Matthews, and S. Baker. Constructing and fitting active appearance models with occlusion. In *First IEEE Workshop on Face Processing in Video (FPiV)*, 2004.

[13] G. Hager and P. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10):1025–1039, 1998.

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

[15] I. Matthews and S. Baker. Active Appearance Models revisited. *International Journal of Computer Vision*, 60(2):135–164, 2004.

[16] S. Sclaroff and J. Isidoro. Active blobs. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1146–1153, 1998.

[17] H. Shum, K. Ikeuchi, and R. Reddy. Principal component analysis with missing data and its application to polyhedral object modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(9):855–867, 1995.

[18] F. de la Torre and M. Black. A framework for robust subspace learning. *International Journal of Computer Vision*, 54(1):117–142, 2003.

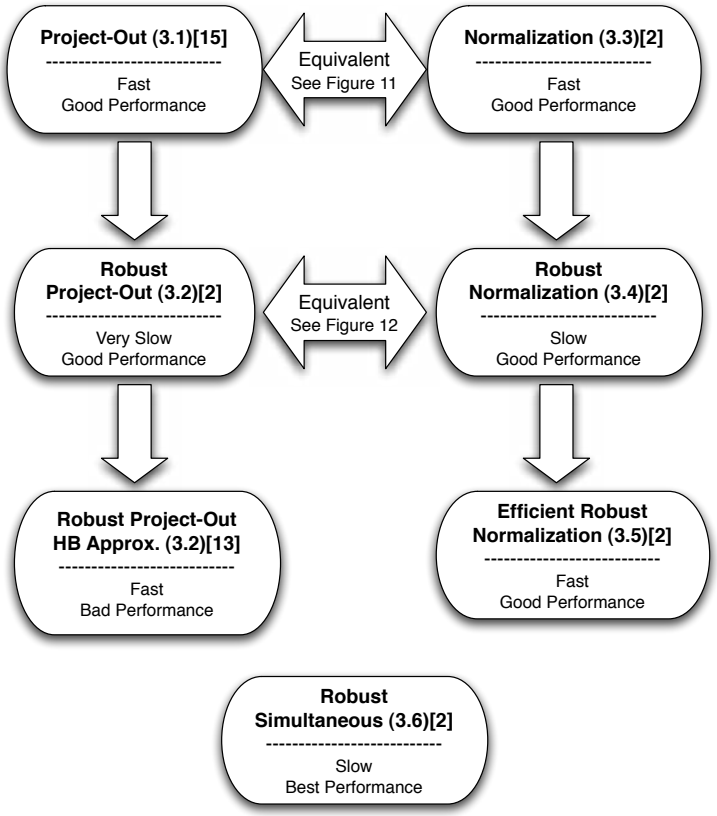


Figure 16: Overview of the algorithms discussed in this paper. The numbers in parenthesis refer to the sections in which the respective algorithm is described. The project-out algorithm was introduced in [15]. The Hager-Belhumeur approximation to the robust project-out algorithm was proposed in [13]. All other algorithms were introduced in [2].