Active Appearance Models Revisited

Iain Matthews and Simon Baker

The Robotics Institute
Carnegie Mellon University

Abstract

Active Appearance Models (AAMs) and the closely related concepts of Morphable Models and Active Blobs are generative models of a certain visual phenomenon. Although linear in both shape and appearance, overall, AAMs are nonlinear parametric models in terms of the pixel intensities. Fitting an AAM to an image consists of minimising the error between the input image and the closest model instance; i.e. solving a nonlinear optimisation problem. We propose an efficient fitting algorithm for AAMs based on the *inverse compositional* image alignment algorithm. We show that the effects of appearance variation during fitting can be precomputed ("projected out") using this algorithm and how it can be extended to include a global shape normalising warp, typically a 2D similarity transformation. We evaluate our algorithm to determine which of its novel aspects improve AAM fitting performance.

Keywords: Active Appearance Models, AAMs, Active Blobs, Morphable Models, fitting, efficiency, Gauss-Newton gradient descent, inverse compositional image alignment.

1 Introduction

Active Appearance Models (AAMs) [7–11, 13], first proposed in [14], and the closely related concepts of Active Blobs [21,22] and Morphable Models [6,18,24], are non-linear, generative, and parametric models of a certain visual phenomenon. The most frequent application of AAMs to date has been face modelling [19]. However, AAMs may be useful for other phenomena too [18, 22]. In a typical application, the first step is to fit the AAM to an input image, i.e. model parameters are found to maximise the "match" between the model instance and the input image. The model parameters are then used in whatever the application is. For example, the parameters could be passed to a classifier to yield a face recognition algorithm. Many different classification tasks are possible. In [19], for example, the same model was used for face recognition, pose estimation, and expression recognition.

Fitting an AAM to an image is a non-linear optimisation problem. The usual approach [7, 10, 11] is to iteratively solve for incremental *additive* updates to the parameters (the shape and appearance coefficients.) Given the current estimates of the shape parameters, it is possible to warp the input image onto the model coordinate frame and then compute an error image between the current model instance and the image that the AAM is being fit to. In most previous algorithms, it is simply assumed that there is a *constant* linear relationship between this error image and the additive incremental updates to the parameters. The constant coefficients in this linear relationship can then be found either by linear regression [7, 13, 14] or by other numerical methods [10, 11].

Unfortunately the assumption that there is such a simple relationship between the error image and the appropriate update to the model parameters is in general incorrect. See Section 2.3.3 for a counterexample. The result is that existing AAM fitting algorithms perform poorly, both in terms of the number of iterations required to converge, and in terms of the accuracy of the final fit. In this paper we propose a new analytical AAM fitting algorithm that does not make this simplifying assumption. Our algorithm is based on an extension to the *inverse compositional* image alignment algorithm [2, 3]. The inverse compositional algorithm is only applicable to sets of warps that form a group. Unfortunately, the set of piecewise affine warps typically used in AAMs does not form a

group. Hence, to use the inverse compositional algorithm, we derive first order approximations to the group operators of *composition* and *inversion.*

The inverse compositional algorithm also allows a different treatment of the appearance variation. Using the approach proposed in [16], we are able to "project out" the appearance variation in a precomputation step and thereby eliminate a great deal of online computation. Another step that we implement differently is shape normalisation. The linear shape variation of AAMs is often augmented by combining it with a 2D similarity transformation to "normalise" the shape. This separates the global transformation into the image from the local variation due to non-rigid shape deformation. We show how the inverse compositional algorithm can be used to simultaneously fit the combination of the two warps: the linear AAM shape variation and a following global shape transformation.

2 Linear Shape and Appearance Models: AAMs

Although they are perhaps the most well-known example, Active Appearance Models are just one instance in a large class of closely related *linear shape and appearance models* and their associated fitting algorithms. This class includes Active Appearance Models (AAMs) [7,11,13,14,19], Shape AAMs [8–10], Direct Appearance Models [17], Active Blobs [22], and Morphable Models [6, 18, 24], as well as possibly others. Many of these models were proposed independently in 1997–1998 [7, 18, 19, 21, 24]. In this paper we use the term *Active Appearance Model* to refer generically to the entire class of linear shape and appearance models. We chose the term Active Appearance Model rather than Active Blob or Morphable Model only because it seems to have stuck better in the vision literature, not because the term was introduced earlier or because AAMs have any particular technical advantage. We also wanted to avoid introducing any new, and potentially confusing, terminology.

Unfortunately the previous literature is already somewhat confusing. The terminology often refers to the combination of a model and a fitting algorithm. For example, Active Appearance

Model [7] strictly only refers to a specific model and an algorithm for fitting that model. Similarly, Direct Appearance Models [17] refers to a different model–fitting algorithm pair. In order to simplify the terminology we make a clear distinction between models and algorithms, *even though this sometimes means we will have to abuse previous terminology.* In particular, we use the term AAM to refer to the model, independent of the fitting algorithm. We also use AAM to refer to a slightly larger class of models than that described in [7].

In essence there are just two types of linear shape and appearance models, those which model shape and appearance independently, and those which parameterize shape and appearance with a single set of linear parameters. We refer to the first set as *independent shape and appearance models* and the second as *combined shape and appearance models*. We will also refer to the first set as *independent AAMs* and the second as *combined AAMs*.

2.1 Independent AAMs

2.1.1 Shape

As the name suggests, independent AAMs model shape and appearance separately. The *shape* of an independent AAM is defined by a mesh and in particular the vertex locations of the mesh. Mathematically, we define the shape $\mathbf{s}$ of an AAM as the coordinates of the v vertices that make up the mesh:

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

See Figure 1 for an example mesh that contains 68 vertices. AAMs allow linear shape variation. This means that the shape $\mathbf{s}$ can be expressed as a base shape $\mathbf{s}_0$ plus a linear combination of n shape vectors $\mathbf{s}_i$:

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

In this expression the coefficients p_i are the shape parameters. Since we can always perform a linear reparameterization, wherever necessary we assume that the vectors $\mathbf{s}_i$ are orthonormal.

AAMs are normally computed from hand labelled training images. The standard approach is to

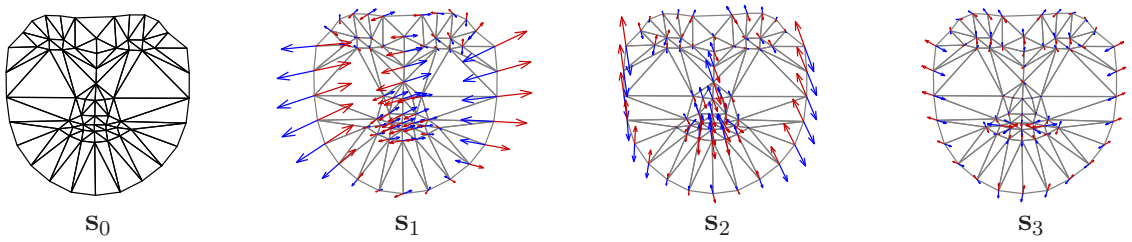


Figure 1: The linear shape model of an independent AAM. The model consists of a triangulated base mesh $\mathbf{s}_0$ plus a linear combination of n shape vectors $\mathbf{s}_i$. The base mesh is shown on the left, and to the right are the first three shape vectors $\mathbf{s}_1$, $\mathbf{s}_2$, and $\mathbf{s}_3$ overlaid on the base mesh.

apply Principal Component Analysis (PCA) to the training meshes [11]. The base shape $\mathbf{s}_0$ is the mean shape and the vectors $\mathbf{s}_0$ are the n eigenvectors corresponding to the n largest eigenvalues. Usually, the training meshes are first *normalised* using a Procrustes analysis [10] before PCA is applied. This step removes variation due to a chosen *global shape normalising transformation* so that the resulting PCA is only concerned with local, non-rigid shape deformation. See Section 4.2 for the details of how such a normalisation affects the AAM fitting algorithm described in this paper.

An example shape model is shown in Figure 1. On the left of the figure, we plot the triangulated base mesh $\mathbf{s}_0$. In the remainder of the figure, the base mesh $\mathbf{s}_0$ is overlaid with arrows corresponding to each of the first three shape vectors $\mathbf{s}_1$, $\mathbf{s}_2$, and $\mathbf{s}_3$.

2.1.2 Appearance

The *appearance* of an independent AAM is defined within the base mesh $\mathbf{s}_0$. Let $\mathbf{s}_0$ also denote the set of pixels $\mathbf{x} = (x, y)^{\mathrm{T}}$ that lie inside the base mesh $\mathbf{s}_0$, a convenient abuse of terminology. The appearance of an 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 $A_i(\mathbf{x})$:

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

$A_0(\mathbf{x})$ $A_1(\mathbf{x})$ $A_2(\mathbf{x})$ $A_3(\mathbf{x})$

Figure 2: The linear appearance variation of an independent AAM. The model consists of a base appearance image A_0 defined on the pixels inside the base mesh $\mathbf{s}_0$ plus a linear combination of m appearance images A_i also defined on the same set of pixels.

In this expression the coefficients λ_i are the appearance parameters. Since we can always perform a linear reparameterization, wherever necessary we assume that the images A_i are orthonormal.

As with the shape component, the base appearance A_0 and the appearance images A_i are normally computed by applying PCA to a set of *shape normalised* training images. Each training image is shape normalised by warping the (hand labelled) training mesh onto the base mesh $\mathbf{s}_0$. Usually the mesh is triangulated and a piecewise affine warp is defined between corresponding triangles in the training and base meshes [11] (although there are ways to avoid triangulating the mesh using, for example, thin plate splines rather than piecewise affine warping [10].) The base appearance is set to be the mean image and the images A_i to be the m eigenimages corresponding to the m largest eigenvalues. The fact that the training images are shape normalised before PCA is applied normally results in a far more compact appearance eigenspace than would otherwise be obtained.

The appearance of an example independent AAM is shown in Figure 2. On the left of the figure we plot the base appearance A_0. On the right we plot the first three appearance images A_1–A_3.

2.1.3 Model Instantiation

Equations (2) and (3) describe the AAM shape and appearance variation. However, they do not describe how to generate a model instance. Given the AAM shape parameters $\mathbf{p} = (p_1, p_2, \ldots, p_n)^{\mathrm{T}}$ we can use Equation (2) to generate the shape of the AAM $\mathbf{s}$. Similarly, given the AAM appearance parameters $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_m)^{\mathrm{T}}$, we can generate the AAM appearance $A(\mathbf{x})$ defined in the interior of the base mesh $\mathbf{s}_0$. The AAM model instance with shape parameters $\mathbf{p}$ and appearance

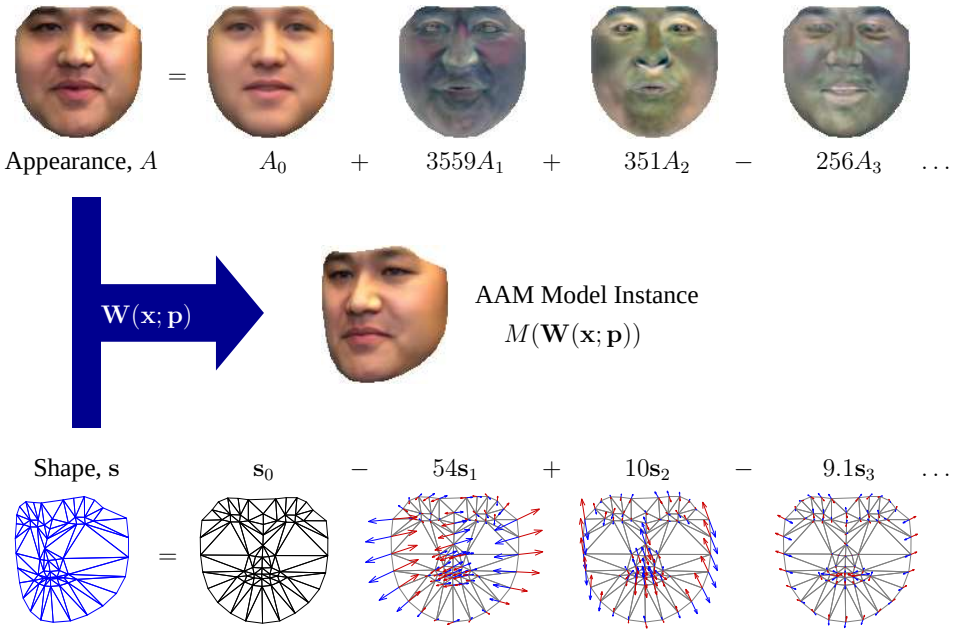


Figure 3: An example of AAM instantiation. The shape parameters $\mathbf{p} = (p_1, p_2, \ldots, p_n)^{\mathrm{T}}$ are used to compute the model shape $\mathbf{s}$ and the appearance parameters $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_m)^{\mathrm{T}}$ are used to compute the model appearance A. The model appearance is defined in the base mesh $\mathbf{s}_0$. The pair of meshes $\mathbf{s}_0$ and $\mathbf{s}$ define a (piecewise affine) warp from $\mathbf{s}_0$ to $\mathbf{s}$ which we denote $\mathbf{W}(\mathbf{x};\mathbf{p})$. The final AAM model instance, denoted $M(\mathbf{W}(\mathbf{x};\mathbf{p}))$, is computed by forwards warping the appearance A from $\mathbf{s}_0$ to $\mathbf{s}$ using $\mathbf{W}(\mathbf{x};\mathbf{p})$.

parameters $\boldsymbol{\lambda}$ is then created by warping the appearance A from the base mesh $\mathbf{s}_0$ to the model shape $\mathbf{s}$. This process is illustrated in Figure 3 for concrete values of $\mathbf{p}$ and $\boldsymbol{\lambda}$.

In particular, the pair of meshes $\mathbf{s}_0$ and $\mathbf{s}$ define a piecewise affine warp from $\mathbf{s}_0$ to $\mathbf{s}$. For each triangle in $\mathbf{s}_0$ there is a corresponding triangle in $\mathbf{s}$. Any pair of triangles define a unique affine warp from one to the other such that the vertices of the first triangle map to the vertices of the second triangle. See Section 4.1.1 for more details. The complete warp is then computed: (1) for any pixel $\mathbf{x}$ in $\mathbf{s}_0$ find out which triangle it lies in, and then (2) warp $\mathbf{x}$ with the affine warp for that triangle. We denote this piecewise affine warp $\mathbf{W}(\mathbf{x};\mathbf{p})$. The final AAM model instance is then computed by warping the appearance A from $\mathbf{s}_0$ to $\mathbf{s}$ with warp $\mathbf{W}(\mathbf{x};\mathbf{p})$. This process is defined by the following equation:

$$M(\mathbf{W}(\mathbf{x};\mathbf{p})) \;=\; A(\mathbf{x}) \tag{4}$$

where M is a 2D image of the appropriate size and shape that contains the model instance. This equation, describes a forwards warping that should be interpreted as follows. Given a pixel $\mathbf{x}$ in

$\mathbf{s}_0$, the destination of this pixel under the warp is $\mathbf{W}(\mathbf{x};\mathbf{p})$. The AAM model M at pixel $\mathbf{W}(\mathbf{x};\mathbf{p})$ in $\mathbf{s}$ is set to the appearance $A(\mathbf{x})$. Implementing this forwards warping to generate the model instance M without holes (see Figure 3) is actually somewhat tricky and is best performed by backwards warping with the inverse warp from $\mathbf{s}$ to $\mathbf{s}_0$. Fortunately, in the AAM fitting algorithms, only backwards warping from $\mathbf{s}$ onto the base mesh $\mathbf{s}_0$ is needed. Finally, note that the piecewise affine warping described in this section could be replaced with any other method of interpolating between the mesh vertices. For example, thin plate splines could be used instead [10].

2.2 Combined AAMs

While independent AAMs have separate shape $\mathbf{p}$ and appearance $\boldsymbol{\lambda}$ parameters, *combined* AAMs just use a single set of parameters $\mathbf{c} = (c_1, c_2, \ldots, c_l)^{\mathrm{T}}$ to parameterize shape:

$$\mathbf{s} \;=\; \mathbf{s}_0 + \sum_{i=1}^{l} c_i \mathbf{s}_i \tag{5}$$

and appearance:

$$A(\mathbf{x}) \;=\; A_0(\mathbf{x}) + \sum_{i=1}^{l} c_i A_i(\mathbf{x}). \tag{6}$$

The shape and appearance parts of the model are therefore coupled. This coupling has a number of disadvantages. For example, it means that we can no longer assume the vectors $\mathbf{s}_i$ and $A_i(\mathbf{x})$ are respectively orthonormal. It also restricts the choice of fitting algorithm. See the discussion at the end of this paper. On the other hand, combined AAMs have a number of advantages. First, the combined formulation is more general and is a strict superset of the independent formulation. To see this, set $\mathbf{c} = (p_1, p_2, \ldots, p_n, \lambda_1, \lambda_2, \ldots, \lambda_m)^{\mathrm{T}}$ and choose $\mathbf{s}_i$ and A_i appropriately. Second, combined AAMs often need less parameters to represent the same visual phenomenon to the same degree of accuracy; i.e. in practice $l \leq m + n$. Therefore fitting may be more efficient.

This second advantage is actually not very significant. Since we will "project out" the appearance variation, as discussed in Section 4.1.5, the computational cost of our new algorithm is mainly dependent on the number of shape parameters n and does not depend significantly on the number of

appearance parameters m. The computational reduction by using l parameters rather than $n+m$ parameters is therefore non-existent. For the same representational accuracy, $l \geq \max(n, m)$. Hence, our algorithm which uses independent AAMs and runs in time $\mathrm{O}(n)$ is actually more efficient than any for combined AAMs and which runs in time $\mathrm{O}(l)$.

Combined AAMs are normally computed by taking an independent AAM and performing (a third) Principal Component Analysis on the appropriately weighted training shape $\mathbf{p}$ and appearance $\boldsymbol{\lambda}$ parameters. The shape and appearance parameters are then linearly reparameterized in terms of the new eigenvectors of the combined PCA. See [11] for the details, although note that the presentation there is somewhat different from the essentially equivalent presentation here.

2.3 Fitting AAMs

2.3.1 Fitting Goal

Suppose we are given an input image $I(\mathbf{x})$ that we wish to fit an AAM to and that we know the optimal shape $\mathbf{p}$ and appearance $\boldsymbol{\lambda}$ parameters for the fit. This means that the image $I(\mathbf{x})$ and the model instance $M(\mathbf{W}(\mathbf{x};\mathbf{p})) = A(\mathbf{x})$ must be similar. In order to define the fitting process, we must formally define the criterion to be optimised. Naturally, we want to minimise the error between $I(\mathbf{x})$ and $M(\mathbf{W}(\mathbf{x};\mathbf{p})) = A(\mathbf{x})$. There are two coordinate frames in which this error can be computed, the coordinate frame of the image I and the coordinate frame of the AAM. The better choice in terms of deriving an efficient fitting algorithm is to use the coordinate frame of the AAM; i.e. the base mesh $\mathbf{s}_0$. If $\mathbf{x}$ is a pixel in $\mathbf{s}_0$, then the corresponding pixel in the input image I is $\mathbf{W}(\mathbf{x};\mathbf{p})$. At pixel $\mathbf{x}$ the AAM has the appearance $A(\mathbf{x}) = A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x})$. At pixel $\mathbf{W}(\mathbf{x};\mathbf{p})$, the input image has the intensity $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$. We want to minimise the sum of squares of the difference between these two quantities:

$$\sum_{\mathbf{x}\in\mathbf{s}_0} \left[A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2 \tag{7}$$

where the sum is performed over all pixels $\mathbf{x}$ in the base mesh $\mathbf{s}_0$. The goal of AAM fitting is then to minimise the expression in Equation (7) simultaneously with respect to the shape parameters $\mathbf{p}$ and the appearance parameters $\boldsymbol{\lambda}$. In general the optimisation is nonlinear in the shape parameters $\mathbf{p}$, although linear in the appearance parameters $\boldsymbol{\lambda}$.

For notational convenience, we define the *error image* in the coordinate frame of the AAM and denote it as:

$$E(\mathbf{x}) \;=\; A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})). \tag{8}$$

The error image can be computed as follows. For each pixel $\mathbf{x}$ in the base mesh $\mathbf{s}_0$, we compute the corresponding pixel $\mathbf{W}(\mathbf{x};\mathbf{p})$ in the input image by warping $\mathbf{x}$ with the piecewise affine warp $\mathbf{W}$. The input image I is then sampled at the pixel $\mathbf{W}(\mathbf{x};\mathbf{p})$; typically it is bilinearly interpolated at this pixel. The resulting value is then subtracted from the appearance $A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x})$ at that pixel and the result stored in E. In other words, the input image I is backwards warped onto the base mesh $\mathbf{s}_0$ with warp $\mathbf{W}$ and then subtracted from the current AAM appearance.

2.3.2 Inefficient Gradient Descent Algorithms

Perhaps the most natural way of minimising the expression in Equation (7) is to use a standard gradient descent optimisation algorithm. Various researchers have tried this. For example, Levenberg-Marquardt was used in [21] and a stochastic gradient descent algorithm was used in [6, 18]. The advantage of these algorithms is that they use a principled, analytical algorithm, the convergence properties of which are well understood. The disadvantage of these gradient descent algorithms is that they are very slow. The partial derivatives, Hessian, and gradient direction all need to be recomputed for each iteration.

2.3.3 Efficient Ad-Hoc Fitting Algorithms

Because standard gradient descent algorithms are so slow, a considerable amount of effort has been devoted to developing alternative fitting algorithms that are more efficient [7,11,21]. In all of these algorithms, the approach is to assume that there is a *constant* linear relationship between the error

image $E(\mathbf{x})$ and *additive* increments to the shape and appearance parameters:

$$\Delta p_i \;=\; \sum_{\mathbf{x}\in\mathbf{s}_0} R_i(\mathbf{x})E(\mathbf{x}) \qquad \text{and} \qquad \Delta\lambda_i \;=\; \sum_{\mathbf{x}\in\mathbf{s}_0} S_i(\mathbf{x})E(\mathbf{x}) \tag{9}$$

where $R_i(\mathbf{x})$ and $S_i(\mathbf{x})$ are constant images defined on the base mesh $\mathbf{s}_0$. Here, constant means that $R_i(\mathbf{x})$ and $S_i(\mathbf{x})$ do not depend on p_i or λ_i. This assumption is motivated by the fact that almost all previous gradient descent algorithms boil down to computing Δp_i and $\Delta\lambda_i$ as linear functions of the error image and then updating $p_i \leftarrow p_i + \Delta p_i$ and $\lambda_i \leftarrow \lambda_i + \Delta\lambda_i$. However, in previous gradient descent algorithms the equivalent of $R_i(\mathbf{x})$ and $S_i(\mathbf{x})$ are not constant, but instead depend on the AAM model parameters $\mathbf{p}$ and $\boldsymbol{\lambda}$. It is because they depend on the current model parameters that they have to be recomputed. In essence, this is why the gradient descent algorithms are so inefficient.

To improve the efficiency, previous AAM fitting algorithms such as [7, 11, 21] have either explicitly or implicitly simply assumed that $R_i(\mathbf{x})$ and $S_i(\mathbf{x})$ *do not* depend on the model parameters. This assumption is an approximation. To provide a counterexample showing that $R_i(\mathbf{x})$ and $S_i(\mathbf{x})$ are not constant, it is sufficient to exhibit two cases where the error image is the same and for which different increments to the parameters should be applied. We demonstrate this in Figure 4.

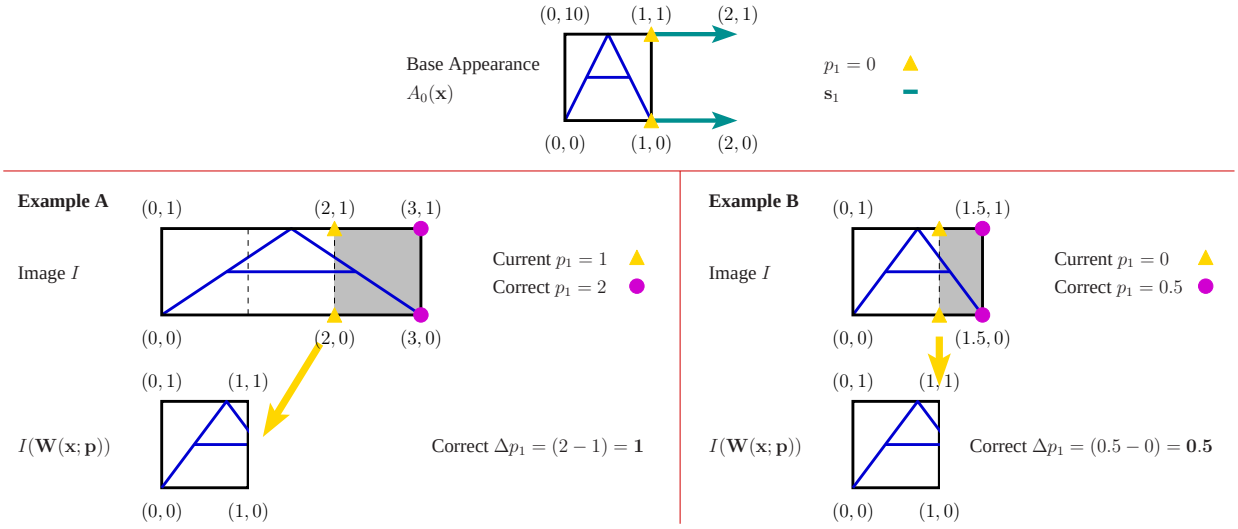


Figure 4: A demonstration that the linear relationship between Δp_i and the error image $E(\mathbf{x})$ does not have constant coefficients. A simple AAM is applied to two input images, both of which yeild the same error image. However, the correct update Δp_1 to the parameter p_1 is different in the two cases.

The base appearance $A_0(\mathbf{x})$ is shown at the top of the figure. The single shape vector $\mathbf{s}_1$ moves the two vertices on the right hand side of the square one unit to the right. Consider Example A: the input image I consists of a stretched version of the base template. In this example the correct value of $p_1 = 2$, and the current estimate of $p_1 = 1$. When I is warped back to the template, $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ is as shown in the lower left of the figure. The correct estimate of Δp_1 should be equal to $2-1 = 1$. Example B is similar: the input image is the base template stretched in the same direction, but not so far. In this case, the correct value of $p_1 = 0.5$ and the current estimate of $p_1 = 0$. When I is warped back to the template, $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ is exactly the same as in Example A and therefore the error image is also identical. However, the correct estimate of Δp_1 in Example B should be $0.5 - 0 = 0.5$. Since this is different from the correct estimate of Δp_1 in Example A, this is a counterexample which demonstrates that in general $R_i(\mathbf{x})$ is not a constant (but depends on the current estimate of $\mathbf{p}$.)

Note that although in this simple example the difference between the two cases could be explained by a global affine warp (see Section 4.2), other examples can easily be provided where a global correction does not help. Similarly, although in this example the direction of Δp_i is correct and it is just the magnitude that is wrong, other examples can be provided where the error images are the same, but the directions of the Δp_i are different.

Although the assumption that $R_i(\mathbf{x})$ and $S_i(\mathbf{x})$ are constant is incorrect, previous algorithms have set out to estimate them in a number of different ways. The original AAM formulation [7, 13, 14] estimated the update functions $R_i(\mathbf{x})$ and $S_i(\mathbf{x})$ by systematically perturbing the model parameters Δp_i and $\Delta \lambda_i$ and recording the corresponding error image $E(\mathbf{x})$. The values of $R_i(\mathbf{x})$ and $S_i(\mathbf{x})$ are then estimated by linear regression. Later, the same authors proposed a finite difference approach [10, 11] that is essentially the same as the *difference decomposition*[1] algorithm used in [21].

[1]Note that the author of the original difference decomposition paper [15] may have been aware of the need to use the *compositional* approach described in Section 3.2. It is hard to tell. However, the use of difference decomposition in [21] makes the constant linear assumption in Equation (24) of that paper.

3 Efficient Gradient Descent Image Alignment

As described above, existing AAM fitting algorithms fall into one of two categories. Either they take the analytical, gradient descent approach, with all the advantages of using a principled algorithm, but are very slow, or they make a provably incorrect assumption to obtain efficiency and in the process forfeit fitting accuracy. Ideally one would like to use a fast, efficient gradient descent algorithm such as that proposed by Hager and Belhumeur [16]. Unfortunately, the algorithm in [16] cannot be applied to piecewise affine warps; in fact it only applies to translations, 2D similarity transformations, affine warps, and a small collection of other esoteric warps.

Is there another efficient gradient descent algorithm? The argument in Section 2.3.3 shows that there cannot be any efficient algorithm that solves for $\Delta\mathbf{p}$ and then updates the parameters $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$. Fortunately, this is not the only way to update the parameters. An alternative is to update the entire warp by composing the current warp with the computed incremental warp with parameters $\Delta\mathbf{p}$. The update rule is then:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p}). \tag{10}$$

This *compositional* approach is different, yet provably equivalent, to the usual *additive* approach [3]. This section describes both additive and compositional gradient descent in the framework of the *image alignment* problem. This is closely related to fitting an AAM, and the next section extends the efficient *inverse compositional* algorithm for independent AAMs.

3.1 Lucas-Kanade Image Alignment

The goal of image alignment is to find the location of a constant template image in an input image. The application of gradient descent to image alignment was first described in Lucas and Kanade [20]. The goal of the Lucas-Kanade algorithm is to find the locally "best" alignment by minimizing the sum of squares difference between a constant template image, $A_0(\mathbf{x})$ say, and an

example image $I(\mathbf{x})$ with respect to the warp parameters $\mathbf{p}$:

$$\sum_{\mathbf{x}} [A_0(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]^2. \tag{11}$$

Note the similarity with Equation (7). As in Section 2, $\mathbf{W}(\mathbf{x};\mathbf{p})$ is a warp that maps the pixels $\mathbf{x}$ from the template (i.e. the base mesh) image to the input image and has parameters $\mathbf{p}$. Note that $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ is an image with the same dimensions as the template; it is the input image I warped backwards onto the same coordinate frame as the template.

Solving for $\mathbf{p}$ is a nonlinear optimisation problem. This is true even if $\mathbf{W}(\mathbf{x};\mathbf{p})$ is linear in $\mathbf{p}$ because, in general, the pixel values $I(\mathbf{x})$ are nonlinear in (and essentially unrelated to) the pixel coordinates $\mathbf{x}$. To linearize the problem, the Lucas-Kanade algorithm assumes that an initial estimate of $\mathbf{p}$ is known and then iteratively solves for increments to the parameters $\Delta\mathbf{p}$; i.e. minimise:

$$\sum_{\mathbf{x}} [A_0(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}))]^2 \tag{12}$$

with respect to $\Delta\mathbf{p}$ and then update $\mathbf{p} \leftarrow \mathbf{p}+\Delta\mathbf{p}$. The expression in Equation (12) can be linearized about $\mathbf{p}$ using a Taylor series expansion to give:

$$\sum_{\mathbf{x}} \left[A_0(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) - \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \right]^2, \tag{13}$$

where ∇I is the *gradient* of the image evaluated at $\mathbf{W}(\mathbf{x};\mathbf{p})$, and $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is the *Jacobian* of the warp evaluated at $\mathbf{p}$. The closed form solution of Equation (13) for $\Delta\mathbf{p}$ is:

$$\Delta\mathbf{p} \;=\; \mathbf{H}^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} [A_0(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))] \tag{14}$$

where $\mathbf{H}$ is the Gauss-Newton approximation to the *Hessian* matrix:

$$\mathbf{H} = \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]. \tag{15}$$

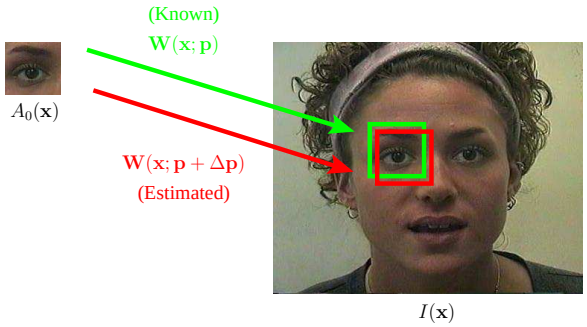


Figure 5: A schematic overview of the Lucas-Kanade (forwards-additive) image alignment algorithm. Given current estimates of the parameters $\mathbf{p}$, Lucas-Kanade linearizes the problem and solves for incremental updates to the parameters $\Delta\mathbf{p}$ that are then added to the current estimates $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$.

An overview of the Lucas-Kanade algorithm is shown in Figure 5. In [3] we refer to this as the *forwards-additive* algorithm. The additive part comes from the iterative update of the warp parameters: $\Delta\mathbf{p}$ is added each time. The forwards part denotes the direction of the warp parameter estimation: the warp projects *into* the image coordinate frame.

The Lucas-Kanade algorithm is slow. In general, both ∇I and $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ depend on $\mathbf{p}$. Hence the Hessian and $\left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}}$ need to be recomputed in every iteration, both of which are slow operations.

3.2 Forwards Compositional Image Alignment

In the Lucas-Kanade algorithm the warp parameters are computed by estimating a $\Delta\mathbf{p}$ offset from the current warp parameters $\mathbf{p}$. The *compositional* framework computes an *incremental warp* $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ to be composed with the current warp $\mathbf{W}(\mathbf{x};\mathbf{p})$. The minimisation is over:

$$\sum_{\mathbf{x}} \left[A_0(\mathbf{x}) - I(\mathbf{W}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p});\mathbf{p}))\right]^2, \tag{16}$$

and the update step involves *composing* the incremental and current warp:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p}). \tag{17}$$

If we compute the solution for $\Delta\mathbf{p}$ in Equation (16) then we have computed the incremental warp in the "image" direction. It can be composed with the current warp using Equation (17) and results

in the *forwards compositional* algorithm [3]. This algorithm was also used in [23]. Taking the Taylor series expansion of Equation (16) gives:

$$\sum_{\mathbf{x}} \left[A_0(\mathbf{x}) - I(\mathbf{W}(\mathbf{W}(\mathbf{x};\mathbf{0});\mathbf{p})) - \nabla I(\mathbf{W}(\mathbf{x};\mathbf{p})) \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta \mathbf{p} \right]^2 . \tag{18}$$

At this point we assume that $\mathbf{p} = 0$ is the identity warp; i.e. $\mathbf{W}(\mathbf{x};\mathbf{0}) = \mathbf{x}$. There are then two differences between Equation (18) and Equation (13). First, the gradient is computed on $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$. Second, the Jacobian is evaluated at $(\mathbf{x};\mathbf{0})$ and therefore is a constant that can be precomputed. The composition update step is computationally more costly than the update step for an additive algorithm, but this is offset by not having to compute the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ in each iteration.

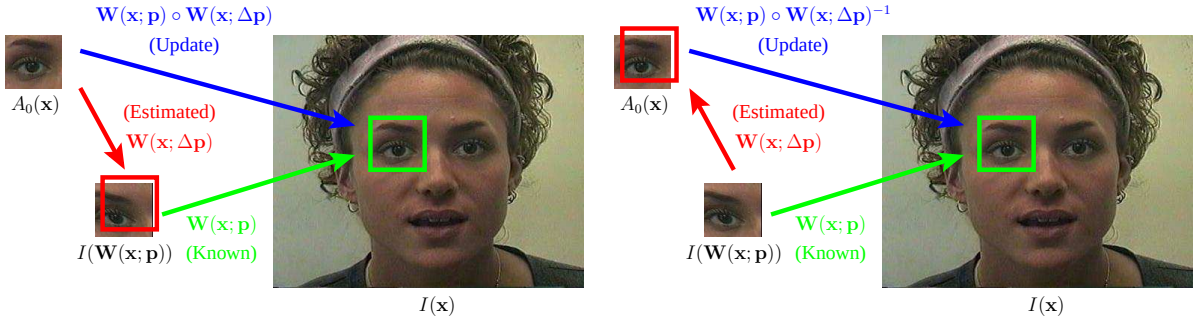
The key point in the forwards compositional algorithm, illustrated in Figure 6(a), is that the update is computed with respect to $\mathbf{p} = \mathbf{0}$ each time. This is why the Jacobian is constant.

3.3 Inverse Compositional Image Alignment

The *inverse compositional* algorithm is a modification of the forwards compositional algorithm where the roles of the template and example image are reversed. Rather than computing the incremental warp with respect to $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ it is computed with respect to the template $A_0(\mathbf{x})$. The proof that this role reversal step results in an equivalent algorithm can be found in [2, 3]. The intuitive reason is that when we reverse the roles of the images (in the compositional case), we just estimate the incremental warp in the opposite "inverse" direction. See Figure 6(b) for an overview of the inverse compositional algorithm.

Reversing the roles of $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ and $A_0(\mathbf{x})$ in Equation (16) results in the inverse compositional algorithm minimizing:

$$\sum_{\mathbf{x}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - A_0(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) \right]^2 , \tag{19}$$



(a) Forwards Compositional (b) Inverse Compositional

Figure 6: (a) A schematic overview of the forwards-compositional image alignment algorithm. Given current estimates of the parameters, the forwards compositional algorithm solves for an incremental warp $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ rather than a simple update to the parameters $\Delta\mathbf{p}$. The incremental warp is then composed with the current estimate of the warp. (b) A schematic overview of the inverse-compositional image alignment algorithm. The roles of $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ and $A_0(\mathbf{x})$ are reversed and the incremental warp $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ is estimated in the other (inverse) direction. The incremental warp therefore has to be inverted before it is composed with the current estimate of the warp.

with respect to $\Delta\mathbf{p}$ and then updating the warp using:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}. \tag{20}$$

Taking the Taylor series expansion of Equation (19) gives:

$$\sum_{\mathbf{x}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p}) - A_0(\mathbf{W}(\mathbf{x};\mathbf{0})) - \nabla A_0 \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p}) \right]^2. \tag{21}$$

Assuming again that $\mathbf{W}(\mathbf{x};\mathbf{0})$ is the identity warp, the solution to this least squares problem is:

$$\Delta\mathbf{p} = \mathbf{H}^{-1} \sum_{\mathbf{x}} \left[\nabla A_0 \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - A_0(\mathbf{x})] \tag{22}$$

where $\mathbf{H}$ is Hessian matrix with I replaced by A_0:

$$\mathbf{H} = \sum_{\mathbf{x}} \left[\nabla A_0 \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla A_0 \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]. \tag{23}$$

Since the template A_0 is constant and the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is always evaluated at $\mathbf{p} = \mathbf{0}$, most of the computation in Equations (22) and (23) can be moved to a precomputation step and performed

The Inverse Compositional Algorithm

Pre-compute:

(3) Evaluate the gradient ∇A_0 of the template $A_0(\mathbf{x})$
(4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x}; \mathbf{0})$
(5) Compute the steepest descent images $\nabla A_0 \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
(6) Compute the Hessian matrix using Equation (23)

Iterate Until Converged:

(1) Warp I with $\mathbf{W}(\mathbf{x}; \mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$
(2) Compute the error image $I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - A_0(\mathbf{x})$
(7) Compute $\sum_{\mathbf{x}} [\nabla A_0 \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - A_0(\mathbf{x})]$
(8) Compute $\Delta \mathbf{p}$ using Equation (22)
(9) 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 inverse compositional algorithm [3]. All of the computationally demanding steps are performed in a pre-computation step. The main algorithm simply consists of image warping (Step 1), image differencing (Step 2), image dot products (Step 7), multiplication with the inverse of the Hessian (Step 8), and the update to the warp (Step 9). All of these steps can be implemented efficiently.

only once. The result is a very efficient image alignment algorithm, see Figure 7 for the details. The main algorithm just iterates: image warping (Step 1), image differencing (Step 2), image dot products (Step 7), multiplication with the precomputed inverse of the Hessian (Step 8), and update to the warp (Step 9). All of these steps can be implemented very efficiently. Note that Steps 1, 2, and 7 parallel equivalent steps in the efficient ad-hoc AAM fitting algorithms. The only additional computation is Steps 8 and 9 which are very efficient. These two steps essentially correct for the current estimates of the parameters $\mathbf{p}$ and avoid the problem illustrated in Figure 4.

4 Applying the Inverse Compositional Algorithm to AAMs

We now show how the inverse compositional algorithm can be applied to independent AAMs. As described in Section 2.1, independent AAMs have separate shape $\mathbf{p}$ and appearance $\boldsymbol{\lambda}$ parameters.

To simplify presentation we initially ignore the global shape normalising transform. (This is still a valid algorithm if one wishes to describe all possible shape variation in the linear shape modes.) We then show how the basic algorithm may be modified to incorporate a global shape normalising transform.

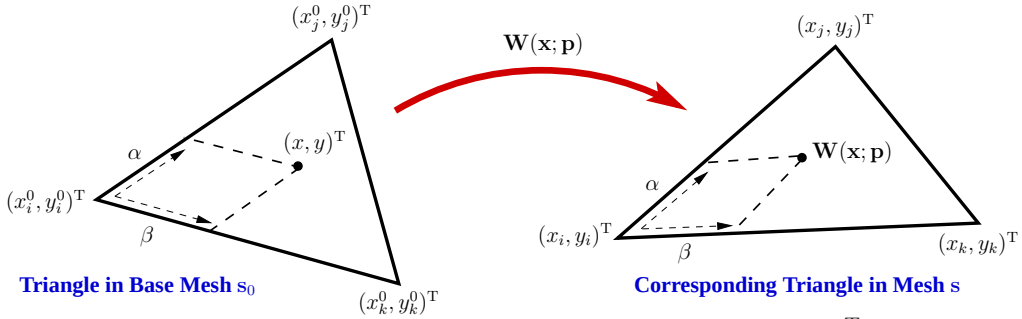


Figure 8: Computing the piecewise affine warp $\mathbf{W}(\mathbf{x};\mathbf{p})$. Each pixel $\mathbf{x} = (x, y)^{\mathrm{T}}$ in the base mesh $\mathbf{s}_0$ lies in a triangle. The pixel $(x, y)^{\mathrm{T}}$ can be decomposed into one vertex plus α times a vector down one side of the triangle plus β times a vector down the other side of the triangle. The destination of $(x, y)^{\mathrm{T}}$ under the piecewise affine warp $\mathbf{W}(\mathbf{x};\mathbf{p})$ is the equivalent expression for the other triangle in mesh $\mathbf{s}$.

4.1 Fitting an AAM Without a Global Shape Transform

We first describe how the inverse compositional AAM algorithm applies when used without either a global shape normalising transform (e.g. similarity transform) or any appearance variation; i.e. when $m = 0$. Comparing Equation (7) with Equation (11) we see that if there is no appearance variation, the inverse compositional image alignment algorithm applies as is. Examining Figure 7 we find that most of the steps in the algorithm are standard vector, matrix, and image operations such as computing image gradients and image differences. The only non-standard steps are: Step 1 warping I with the piecewise affine warp $\mathbf{W}(\mathbf{x};\mathbf{p})$, Step 4 computing the Jacobian of the piecewise affine warp, and Step 9 inverting the incremental piecewise affine warp and composing it with the current estimate of the piecewise affine warp. The following sections describe how each of these steps are performed.

4.1.1 Piecewise Affine Warping

The image $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ is computed by backwards warping the input image I with the warp $\mathbf{W}(\mathbf{x};\mathbf{p})$; i.e. for each pixel $\mathbf{x}$ in the base mesh $\mathbf{s}_0$ we compute $\mathbf{W}(\mathbf{x};\mathbf{p})$ and sample (bilinearly interpolate) the image I at that location. Every pixel $\mathbf{x}$ in the base mesh $\mathbf{s}_0$ lies in a triangle. The vertices of that triangle are $(x_i^0, y_i^0)^{\mathrm{T}}$, $(x_j^0, y_j^0)^{\mathrm{T}}$, and $(x_k^0, y_k^0)^{\mathrm{T}}$. The vertices of the corresponding triangle in the AAM mesh are $(x_i, y_i)^{\mathrm{T}}$, $(x_j, y_j)^{\mathrm{T}}$, and $(x_k, y_k)^{\mathrm{T}}$. These vertices are computed from the current shape parameters $\mathbf{p}$ using Equation (2).

One way to implement the piecewise affine warp is illustrated in Figure 8. Consider the pixel $\mathbf{x} = (x, y)^{\mathrm{T}}$ in the triangle $(x_i^0, y_i^0)^{\mathrm{T}}$, $(x_j^0, y_j^0)^{\mathrm{T}}$, and $(x_k^0, y_k^0)^{\mathrm{T}}$ in the base mesh $\mathbf{s}_0$. This pixel can be uniquely expressed as:

$$\mathbf{x} \;=\; (x, y)^{\mathrm{T}} \;=\; (x_i^0, y_i^0)^{\mathrm{T}} + \alpha \left[(x_j^0, y_j^0)^{\mathrm{T}} - (x_i^0, y_i^0)^{\mathrm{T}} \right] + \beta \left[(x_k^0, y_k^0)^{\mathrm{T}} - (x_i^0, y_i^0)^{\mathrm{T}} \right] \tag{24}$$

where:

$$\alpha \;=\; \frac{(x - x_i^0)(y_k^0 - y_i^0) - (y - y_i^0)(x_k^0 - x_i^0)}{(x_j^0 - x_i^0)(y_k^0 - y_i^0) - (y_j^0 - y_i^0)(x_k^0 - x_i^0)} \tag{25}$$

and:

$$\beta \;=\; \frac{(y - y_i^0)(x_j^0 - x_i^0) - (x - x_i^0)(y_j^0 - y_i^0)}{(x_j^0 - x_i^0)(y_k^0 - y_i^0) - (y_j^0 - y_i^0)(x_k^0 - x_i^0)}. \tag{26}$$

The result of applying the piecewise affine warp is then:

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) \;=\; (x_i, y_i)^{\mathrm{T}} + \alpha \left[(x_j, y_j)^{\mathrm{T}} - (x_i, y_i)^{\mathrm{T}} \right] + \beta \left[(x_k, y_k)^{\mathrm{T}} - (x_i, y_i)^{\mathrm{T}} \right] \tag{27}$$

where $(x_i, y_i)^{\mathrm{T}}$, $(x_j, y_j)^{\mathrm{T}}$, and $(x_k, y_k)^{\mathrm{T}}$ are the vertices of the corresponding triangle in $\mathbf{s}$. Together, Equation (25), (26), and (27) constitute a simple affine warp:

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) \;=\; (a_1 + a_2 \cdot x + a_3 \cdot y, a_4 + a_5 \cdot x + a_6 \cdot y)^{\mathrm{T}}. \tag{28}$$

The 6 parameters of this warp $(a_1, a_2, a_3, a_4, a_5, a_6)$ can easily be computed from the shape parameters $\mathbf{p}$ by combining Equations (2), (25), (26), and (27). This computation only needs to be performed once per triangle, not once per pixel. To implement the piecewise affine warp efficiently, the computation should be structured:

- Given $\mathbf{p}$ compute $(x_i, y_i)^{\mathrm{T}}$ for all vertices in $\mathbf{s}$.
- Compute $(a_1, a_2, a_3, a_4, a_5, a_6)$ for each triangle.
- For each pixel $\mathbf{x}$ in the mesh $\mathbf{s}_0$, lookup the triangle that $\mathbf{x}$ lies in and then lookup the corresponding values of $(a_1, a_2, a_3, a_4, a_5, a_6)$.
- Finally, compute $\mathbf{W}(\mathbf{x}; \mathbf{p})$ using Equation (28).

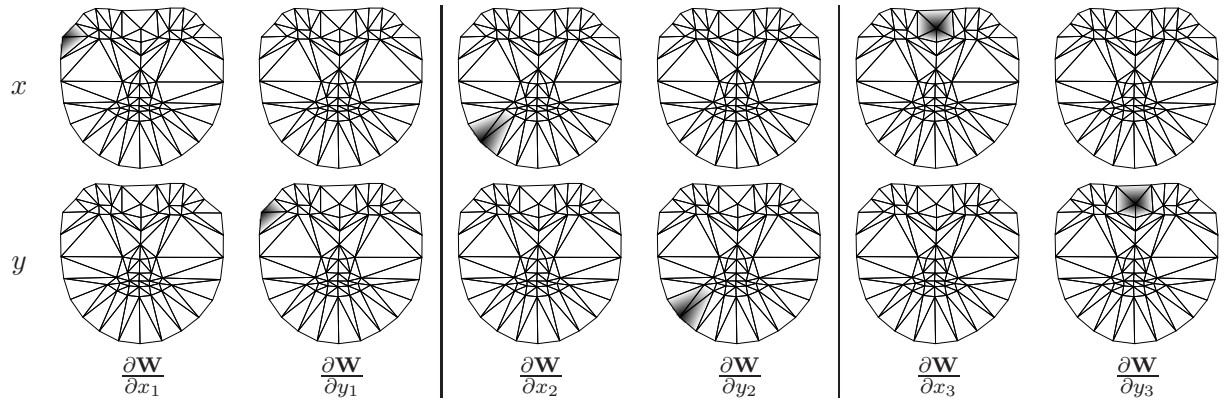


Figure 9: The Jacobians $\frac{\partial \mathbf{W}}{\partial x_i}$ and $\frac{\partial \mathbf{W}}{\partial y_i}$ with respect to the vertices of the mesh $\mathbf{s}$ for 3 different vertices i. The x component of the Jacobian is in the top row and the y component is in the bottom row.

If we raster scan the mesh $\mathbf{s}_0$ we can avoid looking up $(a_1, a_2, a_3, a_4, a_5, a_6)$ most of the time by creating a lookup table that codes when the triangle identity changes.

4.1.2 Computing the Warp Jacobian

The destination of the pixel $\mathbf{x}$ under the piecewise affine warp $\mathbf{W}(\mathbf{x}; \mathbf{p})$ depends on the AAM shape parameters $\mathbf{p}$ through the vertices of the mesh $\mathbf{s}$. From Equation (1) recall that these vertices are denoted: $\mathbf{s} = (x_1, y_1, x_2, y_2, \ldots, x_v, y_v)^{\mathrm{T}}$. Applying the chain rule to the warp $\mathbf{W}(\mathbf{x}; \mathbf{p})$ gives:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \sum_{i=1}^{v} \left[\frac{\partial \mathbf{W}}{\partial x_i} \frac{\partial x_i}{\partial \mathbf{p}} + \frac{\partial \mathbf{W}}{\partial y_i} \frac{\partial y_i}{\partial \mathbf{p}} \right]. \tag{29}$$

The first components of the Jacobian are $\frac{\partial \mathbf{W}}{\partial x_i}$ and $\frac{\partial \mathbf{W}}{\partial y_i}$, the Jacobians of the warp with respect to the vertices of the mesh $\mathbf{s}$. From Equation (27) we see that:

$$\frac{\partial \mathbf{W}}{\partial x_i} = (1 - \alpha - \beta, 0)^{\mathbf{T}} \qquad \text{and} \qquad \frac{\partial \mathbf{W}}{\partial y_i} = (0, 1 - \alpha - \beta)^{\mathbf{T}}. \tag{30}$$

These Jacobians are images the size of the base mesh $\mathbf{s}_0$. Examples of these Jacobians are included in Figure 9. Each image is the Jacobian with respect to a particular vertex. The Jacobian $\frac{\partial \mathbf{W}}{\partial x_i}$ denotes the rate of change of the destination of the warp $\mathbf{W}(\mathbf{x}; \mathbf{p})$ with respect to the vertex x_i. As can be seen, the Jacobian is only non-zero in the triangles around x_i. It takes the maximum value of 1 at the vertex x_i and decays away linearly as described by Equation (30).

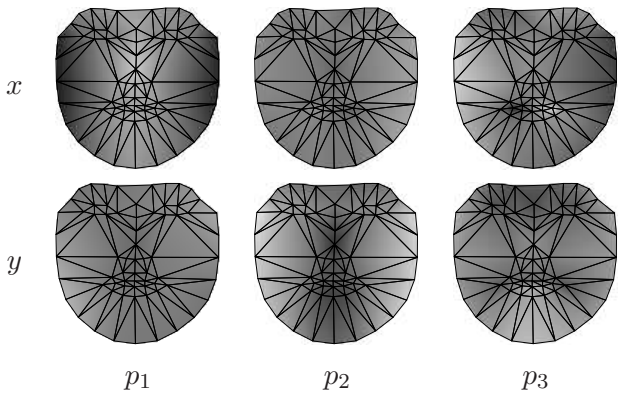


Figure 10: The Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ corresponding to the three shape vectors in Figure 1. The first mode p_1 mostly corresponds to a left-right rotation of the face, the second mode p_2 to the up-down rotation of the head, and the third mode p_3 to motion of the mouth.

The second components of the Jacobian are $\frac{\partial x_i}{\partial \mathbf{p}}$ and $\frac{\partial y_i}{\partial \mathbf{p}}$. Differentiating Equation (2) gives:

$$\frac{\partial x_i}{\partial \mathbf{p}} \; = \; (\mathbf{s}_1^{x_i}, \mathbf{s}_1^{x_i}, \ldots, \mathbf{s}_n^{x_i}) \qquad \text{and} \qquad \frac{\partial y_i}{\partial \mathbf{p}} \; = \; (\mathbf{s}_1^{y_i}, \mathbf{s}_1^{y_i}, \ldots, \mathbf{s}_n^{y_i}) \tag{31}$$

where $\mathbf{s}_j^{x_i}$ denotes the component of $\mathbf{s}_j$ that corresponds to x_i and similarly for y_i. The quantities $\frac{\partial x_i}{\partial \mathbf{p}}$ and $\frac{\partial y_i}{\partial \mathbf{p}}$ are therefore just the shape vectors $\mathbf{s}_i$ rearranged appropriately.

Putting together the components in Equations (30–31) results in the overall Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ looking like those in Figure 10. The array of 2×3 (base mesh $\mathbf{s}_0$ sized) images correspond to the Jacobian for the three shape vectors in Figure 1. In particular, the first pair of images mostly correspond to a left-right rotation of the face, the second pair to the up-down rotation of the head, and the third pair to the opening and closing of the mouth.

4.1.3 Warp Inversion

In Step 9 of the inverse compositional algorithm we must invert the incremental piecewise affine warp $\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})$ to compute $\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})^{-1}$. Since:

$$\mathbf{W}(\mathbf{x}; \Delta\mathbf{p}) \; = \; \mathbf{W}(\mathbf{x}; \mathbf{0}) + \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \; = \; \mathbf{x} + \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} + \mathrm{O}(\Delta\mathbf{p}^2) \tag{32}$$

(remember that $\mathbf{W}(\mathbf{x};\mathbf{0}) = \mathbf{x}$ is the identity warp) we therefore have:

$$\mathbf{W}(\mathbf{x};\Delta\mathbf{p}) \circ \mathbf{W}(\mathbf{x};-\Delta\mathbf{p}) \;=\; \mathbf{x} - \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\Delta\mathbf{p} + \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\Delta\mathbf{p} \;=\; \mathbf{x} + \mathrm{O}(\Delta\mathbf{p}^2). \tag{33}$$

It therefore follows that to first order in $\Delta\mathbf{p}$:

$$\mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1} \;=\; \mathbf{W}(\mathbf{x};-\Delta\mathbf{p}). \tag{34}$$

Note that the two Jacobians in Equation (33) are not evaluated at exactly the same location, but since they are evaluated at points $\mathrm{O}(\Delta\mathbf{p})$ apart, they are equal to zeroth order in $\Delta\mathbf{p}$. Since the difference is multiplied by $\Delta\mathbf{p}$ we can ignore the first and higher order terms. Also note that the composition of two warps is not strictly defined and so the argument in Equation (33) is informal. The essence of the argument is correct, however. Once we have the derived the first order approximation to the composition of two piecewise affine warps below, we can then use that definition of composition in the argument above. The result is that the warp $\mathbf{W}(\mathbf{x};-\Delta\mathbf{p})$ followed by the warp $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ is equal to the identity warp to first order in $\Delta\mathbf{p}$.

4.1.4 Composing the Incremental Warp with the Current Warp Estimate

After we have inverted the piecewise affine warp $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ to compute $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$ in Step 9 we must compose the result with the current warp $\mathbf{W}(\mathbf{x};\mathbf{p})$ to obtain $\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$. Given the current estimate of the parameters $\mathbf{p}$ the current mesh vertex locations $\mathbf{s} = (x_1, y_1, \ldots, x_v, y_v)^{\mathrm{T}}$ can be computed using Equation (2). From the previous section, the parameters of $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$ are $-\Delta\mathbf{p}$. Given these parameters, we can use Equation (2) again to estimate the corresponding changes to the base mesh vertex locations:

$$\Delta\mathbf{s}_0 \;=\; -\sum_{i=1}^{n} \Delta p_i \mathbf{s}_i \tag{35}$$

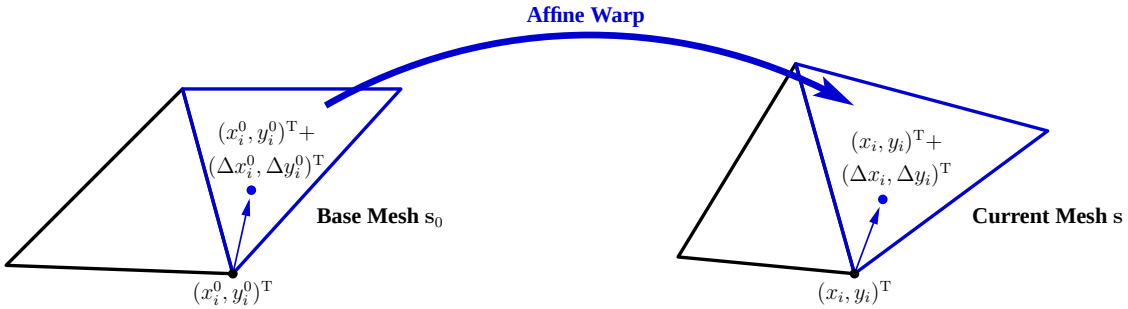


Figure 11: Composing the incremental warp $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$ with the current warp $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$. The current mesh $\mathbf{s}$ and incremental updates to the base mesh $(\Delta x_i^0, \Delta y_i^0)^{\mathrm{T}}$ are known. We need to compute incremental updates to the current mesh $(\Delta x_i, \Delta y_i)^{\mathrm{T}}$. This can be performed by applying the affine warp for each triangle about the i^{th} vertex to $(x_i^0, y_i^0)^{\mathrm{T}} + (\Delta x_i^0, \Delta y_i^0)^{\mathrm{T}}$ to obtain multiple estimates of $(x_i, y_i)^{\mathrm{T}} + (\Delta x_i, \Delta y_i)^{\mathrm{T}}$ which may be averaged to compute the new mesh vertex locations.

where $\Delta\mathbf{s}_0 = (\Delta x_1^0, \Delta y_1^0, \ldots, \Delta x_v^0, \Delta y_v^0)^{\mathrm{T}}$ are the changes to the base mesh vertex locations corresponding to $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$. In order to compose $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$ with $\mathbf{W}(\mathbf{x};\mathbf{p})$, we must compute the corresponding changes to the current mesh vertex locations $\Delta\mathbf{s} = (\Delta x_1, \Delta y_1, \ldots, \Delta x_v, \Delta y_v)^{\mathrm{T}}$. Once we know these locations we can then compute the parameters of $\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$ by solving Equation (2) for the new parameters:

$$p_i^{'} \;=\; \mathbf{s}_i \cdot (\mathbf{s} + \Delta\mathbf{s} - \mathbf{s}_0) \tag{36}$$

where $p_i^{'}$ is the i^{th} parameter of $\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$, $\cdot$ denotes vector dot product, and the shape vectors $\mathbf{s}_i$ are assumed to be orthonormal.

All that remains to be described is how to compute $\Delta\mathbf{s}$ from $\Delta\mathbf{s}_0$. Considering the i^{th} vertex in the mesh, we need to compute $(\Delta x_i, \Delta y_i)^{\mathrm{T}}$ in the current mesh $\mathbf{s}$ from $(\Delta x_i^0, \Delta y_i^0)^{\mathrm{T}}$ in the base mesh $\mathbf{s}_0$. This is illustrated in Figure 11. Now consider any one of the mesh triangles that contains the i^{th} vertex. For this triangle there is an affine warp between the base mesh $\mathbf{s}_0$ and the current mesh $\mathbf{s}$. See Section 4.1.1 for more details. One way to compute $(\Delta x_i, \Delta y_i)^{\mathrm{T}}$ from $(\Delta x_i^0, \Delta y_i^0)^{\mathrm{T}}$ is to apply the affine warp for that specific triangle to $(x_i^0, y_i^0)^{\mathrm{T}} + (\Delta x_i^0, \Delta y_i^0)^{\mathrm{T}}$ and so obtain $(x_i, y_i)^{\mathrm{T}} + (\Delta x_i, \Delta y_i)^{\mathrm{T}}$. The only problem with this approach is which triangle do we use? Using a different triangle means using a different affine warp and so the destination $(x_i, y_i)^{\mathrm{T}} + (\Delta x_i, \Delta y_i)^{\mathrm{T}}$ may be different. This is the reason that the composition of two piecewise

affine warps is hard to define. In general there will be several triangles that share the i^{th} vertex. One possibility is to use the triangle that contains the point $(x_i^0, y_i^0)^{\mathrm{T}} + (\Delta x_i^0, \Delta y_i^0)^{\mathrm{T}}$. The problem with this approach is that the point could lie outside the base mesh $\mathbf{s}_0$. Instead we compute the destination $(x_i, y_i)^{\mathrm{T}} + (\Delta x_i, \Delta y_i)^{\mathrm{T}}$ for every triangle that shares the i^{th} vertex and then average the result. This will tend to smooth the warp at each vertex, but that is desirable anyway.

4.1.5 Including Appearance Variation

We have now described all of the steps needed to apply the inverse compositional algorithm to an independent AAM assuming that there is no appearance variation. More generally, we wish to use the same algorithm to minimise the expression in Equation (7). This can be achieved using the technique proposed in [16]. Rewrite Equation (7) as:

$$\sum_{\mathbf{x}\in\mathbf{s}_0}\left[A_0(\mathbf{x}) + \sum_{i=1}^{m}\lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\right]^2 = \left\| A_0(\mathbf{x}) + \sum_{i=1}^{m}\lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\right\|^2 \quad (37)$$

where $\|\cdot\|$ is the L2 norm. This expression must be minimised simultaneously with respect to $\mathbf{p}$ and $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_m)^{\mathrm{T}}$. If we denote the linear subspace spanned by a collection of vectors A_i by $\mathrm{span}(A_i)$ and its orthogonal complement by $\mathrm{span}(A_i)^{\perp}$, Equation (37) can be rewritten as:

$$\left\| A_0(\mathbf{x}) + \sum_{i=1}^{m}\lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\right\|^2_{\mathrm{span}(A_i)^{\perp}} + \left\| A_0(\mathbf{x}) + \sum_{i=1}^{m}\lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\right\|^2_{\mathrm{span}(A_i)} \quad (38)$$

where $\|\cdot\|^2_L$ denotes the square of the L2 norm of the vector projected into the linear subspace L. The first of the two terms immediately simplifies. Since the norm only considers the components of vectors in the orthogonal complement of $\mathrm{span}(A_i)$, any component in $\mathrm{span}(A_i)$ itself can be dropped. We therefore wish to minimise:

$$\| A_0(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\|^2_{\mathrm{span}(A_i)^{\perp}} + \left\| A_0(\mathbf{x}) + \sum_{i=1}^{m}\lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\right\|^2_{\mathrm{span}(A_i)} . \quad (39)$$

The first of these two terms does not depend upon λ_i. For any $\mathbf{p}$, the minimum value of the second term is always 0. Therefore the minimum value can be found sequentially by first minimising the first term with respect to $\mathbf{p}$ alone, and then using that optimal value of $\mathbf{p}$ as a constant to minimise the second term with respect to the λ_i. Assuming that the basis vectors A_i are orthonormal, the second minimisation has a simple closed-form solution:

$$\lambda_i \; = \; \sum_{\mathbf{x} \in \mathbf{s}_0} A_i(\mathbf{x}) \cdot [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - A_0(\mathbf{x})] \, , \tag{40}$$

the dot product of A_i with the final error image obtained after doing the first minimisation.

Minimising the first term in Equation (39) is very similar to applying the the inverse compositional algorithm to the AAM with no appearance variation. The only difference is that we need to work in linear subspace $\mathrm{span}(A_i)^\perp$ rather than in the full vector space defined over the pixels in $\mathbf{s}_0$. We do not even need to project the error image into this subspace. All we need to do is project $\nabla A_0 \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ into the subspace $\mathrm{span}(A_i)^\perp$ in Step 5 of the inverse compositional algorithm, see Figure 7. The reason error image does not need to be projected into this subspace is because Step 7 of the algorithm is the dot product of the error image with $\nabla A_0 \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$. As long as one of the two terms of the dot product is projected into a linear subspace, the result is the same as if they both were. Effectively, the error due to appearance variation is "projected out".

We refer to $\nabla A_0 \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ as the steepest descent images because this dot-product describes the shortest path down the error surface, ignoring the normalisation provided by the Hessian [3]. Denote the elements of the image $\nabla A_0 \frac{\partial \mathbf{W}}{\partial p_j}$ as $\mathrm{SD}_j(\mathbf{x})$ for each parameter $j = 1, \ldots, n$. The steepest descent images can be projected into $\mathrm{span}(A_i)^\perp$ as follows:

$$\mathrm{SD}_j(\mathbf{x}) = \nabla A_0 \frac{\partial \mathbf{W}}{\partial p_j} \; - \; \sum_{i=1}^{m} \left[\sum_{\mathbf{x} \in \mathbf{s}_0} A_i(\mathbf{x}) \cdot \nabla A_0 \frac{\partial \mathbf{W}}{\partial p_j} \right] A_i(\mathbf{x}) \tag{41}$$

See Figure 12 for a summary of the inverse compositional algorithm with appearance variation and [1] for more details of the various ways that the inverse compositional algorithm can be combined with linear appearance variation.

The Inverse Compositional Algorithm with Appearance Variation

Pre-compute:

(3) Evaluate the gradient ∇A_0 of the template $A_0(\mathbf{x})$
(4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x}; \mathbf{0})$
(5) Compute the modified steepest descent images using Equation (41)
(6) Compute the Hessian matrix using modified steepest descent images

Iterate:

(1) Warp I with $\mathbf{W}(\mathbf{x}; \mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$
(2) Compute the error image $I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - A_0(\mathbf{x})$
(7) Compute dot product of modified steepest descent images with error image
(8) Compute $\Delta \mathbf{p}$ by multiplying by inverse Hessian
(9) Update the warp $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta \mathbf{p})^{-1}$

Post-computation:

(10) Compute λ_i using Equation (40). [Optional step]

Figure 12: The inverse compositional algorithm with appearance variation. The only differences from the algorithm without appearance variation are: (1) the expression in Equation (41) is computed in place of the steepest descent images $\nabla A_0 \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ in Step 5, and (2) Step 10 is added to compute the appearance parameters.

4.2 Including a Global Shape Normalising Transform

When constructing an AAM [10], the training meshes are usually "normalised" before PCA is applied to compute the shape eigenvectors. The *global shape normalising transform* is often a 2D similarity transformation (translation, rotation, and scale), although other global warps such as affine warps could be used instead [4]. The global shape normalising transform separates the (often large) changes in mesh location due to the formation of the object in the image from the local mesh variation due to non-rigid shape deformation. The training meshes are typically normalised using an iterative Procrustes analysis [10, 12] that removes, for example: translation, rotation and scale differences across all training meshes. The shape eigenvectors are then computed by applying PCA to the normalised training meshes.

Obviously, because of this normalisation, the shape vectors of the AAM do not model, for example: translation, rotation, and scale. However, the image data that the AAM is fit to will, in general, be translated, rotated, and scaled by the image formation process (i.e. camera location.) To fit a "normalised" AAM to such data it is therefore necessary to incorporate a matching global

shape transform to the AAM.

In this section we describe the 2D similarity transform we use for global shape normalisation. We also show how to parameterize it to simplify computation for the inverse compositional AAM fitting algorithm that includes a global shape transform.

4.2.1 Parameterizing a Global 2D Similarity Transform

We define $\mathbf{N}(\mathbf{x};\mathbf{q})$ as the global shape normalising transform for the AAM training data. For example, $\mathbf{N}(\mathbf{x};\mathbf{q})$ might be the set of 2D similarity transforms:

$$\mathbf{N}(\mathbf{x};\mathbf{q}) \;=\; \begin{pmatrix} (1+a) & -b \\ b & (1+a) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{42}$$

where the four parameters $\mathbf{q} = (a, b, t_x, t_y)^{\mathrm{T}}$ have the following interpretations. The first pair (a, b) are related to the scale k and rotation θ : $a = k\cos\theta - 1$, and $b = k\sin\theta$. The second pair (t_x, t_y) are the x and y translations. Equation (42) is parameterized so that the identity transform is $\mathbf{q} = \mathbf{0}$.

Note that the above is not the only way to parameterize the set of 2D similarity transformations. Another way is to define the set as a special subset of the set of piecewise affine warps used in AAMs. The base mesh is $\mathbf{s}_0 = (x_1^0, y_1^0, \ldots, x_v^0, y_v^0)^{\mathrm{T}}$. If we choose $\mathbf{s}_1^* = \mathbf{s}_0 = (x_1^0, y_1^0, \ldots, x_v^0, y_v^0)^{\mathrm{T}}$, $\mathbf{s}_2^* = (-y_1^0, x_1^0, \ldots, -y_v^0, x_v^0)^{\mathrm{T}}$, $\mathbf{s}_3^* = (1, 0, \cdots, 1, 0)^{\mathrm{T}}$, and $\mathbf{s}_4^* = (0, 1, \cdots, 0, 1)^{\mathrm{T}}$ then the set of allowed linear AAM shape variation is exactly equal to the set of 2D similarity transformations:

$$\mathbf{N}(\mathbf{x};\mathbf{q}) \;=\; \mathbf{s}_0 + \sum_{i=1}^{4} q_i \mathbf{s}_i^* \tag{43}$$

where $\mathbf{q} = (q_1, q_2, q_3, q_4)$. Note that it is straightforward to transform between the parameters of this linear parameterization and the parameters of the more common (nonlinear) parameterization in Equation (42):

$$\begin{aligned} a &= q_1 & \qquad t_x &= q_3 \\ b &= q_2 & \qquad t_y &= q_4. \end{aligned} \tag{44}$$

We use the representation of $\mathbf{N}$ in Equation (43) because this is similar to that of $\mathbf{W}$ and therefore much of the analysis in Section 4.1 can be reused, most notably the derivation of the Jacobian.

4.2.2 Adding a Global Shape Normalising Transform to an AAM

Given $\mathbf{N}(\mathbf{x};\mathbf{q})$, the definition of an AAM is then augmented from Equation (4) to:

$$M(\mathbf{N}(\mathbf{W}(\mathbf{x};\mathbf{p});\mathbf{q})) \;=\; A(\mathbf{x}) \;=\; A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) \tag{45}$$

where M is a 2D image of the appropriate size and shape that contains the model instance. Given appearance parameters $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_m)^{\mathrm{T}}$, the AAM appearance $A(\mathbf{x})$ is generated in the base mesh $\mathbf{s}_0$. The model instance M is then created by warping the appearance A from the base mesh $\mathbf{s}_0$ first with the linear shape warp $\mathbf{W}(\mathbf{x};\mathbf{p})$ and then with the normalising warp $\mathbf{N}(\mathbf{x};\mathbf{q})$.

Note that this new definition of an AAM in Equation (45) is different from just augmenting the linear shape variation to include a 2D similarity transformation. For example, we could prepend the four 2D similarity vectors $\mathbf{s}_1^*, \mathbf{s}_2^*, \mathbf{s}_3^*$ and $\mathbf{s}_4^*$ to the AAM shape vectors $\mathbf{s}_1$ to $\mathbf{s}_n$ and then orthonormalise. The AAM would then be able to move under 2D similarity transformations, *as well as* the original linear AAM shape variation. This is not the same as moving under the linear shape variation *followed by* the 2D similarity transformation. Whether or not to use a global shape transform, what it should be, and how it affects the performance of an AAM, is an interesting, and relatively unstudied question.

In the following description, we will assume that the global shape transform $\mathbf{N}(\mathbf{x};\mathbf{q})$ is the 2D similarity transform defined in Equation (43) by the shape vectors $\mathbf{s}_1^* = (x_1^0, y_1^0, \ldots, x_v^0, y_v^0)^{\mathrm{T}}$, $\mathbf{s}_2^* = (-y_1^0, x_1^0, \ldots, -y_v^0, x_v^0)^{\mathrm{T}}$, $\mathbf{s}_3^* = (1, 0, \cdots, 1, 0)^{\mathrm{T}}$ and $\mathbf{s}_4^* = (0, 1, \cdots, 0, 1)^{\mathrm{T}}$ that are then orthonormalised. The orthonormalisation allows us to directly compute new parameters in the composition step. Assuming that the base shape is zero mean ($\sum_i^v x_i^0 = 0$, $\sum_i^v y_i^0 = 0$), the only change to Section 4.2.1 is that the similarity parameter values in Equation (44) must be weighted by the inverse of the orthonormalising multiplier when converting between (q_1, q_2, q_3, q_4) and (a, b, t_x, t_y).

Furthermore, we assume that the two sets of shape vectors $\mathbf{s}_i$ and $\mathbf{s}_i^*$ are orthogonal to each other. This should happen automatically when the AAM is constructed. When each shape vector is normalised by removing the similarity transform, we effectively project it into the subspace orthogonal to $\mathbf{s}_i^*$. Since the shape vectors $\mathbf{s}_i$ are computed by applying PCA to a collection of vectors that are orthogonal to $\mathbf{s}_i^*$, they themselves should be orthogonal to $\mathbf{s}_i^*$. In practice, due to various sources of noise, $\mathbf{s}_i$ and $\mathbf{s}_i^*$ are never quite orthogonal to each other. This minor error only affects the composition step and can either be ignored or, preferably, the complete set of $\mathbf{s}_i$ and $\mathbf{s}_i^*$ can be orthonormalised.

4.3 Fitting an AAM Including a Global Shape Transform

We now describe how the inverse compositional algorithm can be used to fit an AAM with a global shape transformation; i.e. apply the inverse compositional algorithm to the warp:

$$\mathbf{N} \circ \mathbf{W}(\mathbf{x}; \mathbf{q}, \mathbf{p}) \;=\; \mathbf{N}(\mathbf{W}(\mathbf{x}; \mathbf{p}); \mathbf{q}) \tag{46}$$

rather than the warp $\mathbf{W}(\mathbf{x}; \mathbf{p})$. Fitting the AAM to an image $I(\mathbf{x})$ then consists of minimising:

$$\sum_{\mathbf{x} \in \mathbf{s}_0} \left[A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{N}(\mathbf{W}(\mathbf{x}; \mathbf{p}); \mathbf{q})) \right]^2 \tag{47}$$

simultaneously with respect to the appearance parameters $\boldsymbol{\lambda}$, the linear shape parameters $\mathbf{p}$, and the global shape warp parameters $\mathbf{q}$. To do this, we repeat the steps in Section 4.1 for the warp $\mathbf{N} \circ \mathbf{W}$ with parameters $(\mathbf{q}, \mathbf{p})$ rather than the warp $\mathbf{W}$ with parameters $\mathbf{p}$.

4.3.1 Warping

To perform the piecewise affine warping to compute $I(\mathbf{N}(\mathbf{W}(\mathbf{x}; \mathbf{p}); \mathbf{q}))$ as in Section 4.1.1 we need to know the destination of the base mesh $\mathbf{s}_0$ under the warp $\mathbf{N} \circ \mathbf{W}$. As a convenient abuse of terminology, denote the destination of the mesh $\mathbf{s}$ under the warp $\mathbf{W}(\mathbf{x}; \mathbf{p})$ by $\mathbf{W}(\mathbf{s}; \mathbf{p})$. From

Equation (2) we have:

$$\mathbf{W}(\mathbf{s}_0;\mathbf{p}) \;=\; \mathbf{s}_0 + \sum_{i=1}^{n} p_i \mathbf{s}_i. \tag{48}$$

We now want to compute $\mathbf{N}(\mathbf{W}(\mathbf{s}_0;\mathbf{p});\mathbf{q})$; i.e. we need to warp every vertex in $\mathbf{W}(\mathbf{s}_0;\mathbf{p})$ with the 2D similarity transform $\mathbf{N}(\mathbf{x};\mathbf{q})$. This can be performed in the following manner. Since:

$$\mathbf{N}(\mathbf{s}_0;\mathbf{q}) \;=\; \mathbf{s}_0 + \sum_{i=1}^{4} q_i \mathbf{s}_i^* \tag{49}$$

we can compute the destination of $\mathbf{s}_0$ under $\mathbf{N}$, and in particular, the destination of any one triangle under $\mathbf{N}$. Using the technique described in Section 4.1.1 we can compute the affine warp for this triangle. Since the set of 2D similarity transforms is a subset of the set of affine warps, we can just apply this affine warp to every vertex in $\mathbf{W}(\mathbf{s}_0;\mathbf{p})$ to compute the destination of the base mesh under $\mathbf{N}\circ\mathbf{W}$. The piecewise affine warping is then performed exactly as described in Section 4.1.1.

4.3.2 Computing the Jacobian

The Jacobian of the warp $\mathbf{N}\circ\mathbf{W}$ is $(\frac{\partial}{\partial\mathbf{q}}\mathbf{N}\circ\mathbf{W}, \frac{\partial}{\partial\mathbf{p}}\mathbf{N}\circ\mathbf{W})$. Since $\mathbf{W}(\mathbf{x};\mathbf{0}) = \mathbf{N}(\mathbf{x};\mathbf{0}) = \mathbf{x}$, the identity warp, and we are evaluating the Jacobian at $\mathbf{p}=\mathbf{0}$, $\mathbf{q}=\mathbf{0}$, we have:

$$\frac{\partial}{\partial\mathbf{q}}\mathbf{N}\circ\mathbf{W} \;=\; \frac{\partial\mathbf{N}}{\partial\mathbf{q}} \tag{50}$$

and:

$$\frac{\partial}{\partial\mathbf{p}}\mathbf{N}\circ\mathbf{W} \;=\; \frac{\partial\mathbf{W}}{\partial\mathbf{p}}. \tag{51}$$

Since the representation of the warps $\mathbf{W}$ and $\mathbf{N}$ is the same (linear shape variation defined on the base mesh), the computation of the Jacobian $(\frac{\partial}{\partial\mathbf{q}}\mathbf{N}\circ\mathbf{W}, \frac{\partial}{\partial\mathbf{p}}\mathbf{N}\circ\mathbf{W}) = (\frac{\partial\mathbf{N}}{\partial\mathbf{q}}, \frac{\partial\mathbf{W}}{\partial\mathbf{p}})$ is exactly as described in Section 4.1.2, just for $\frac{\partial\mathbf{N}}{\partial\mathbf{q}}$ we use $\mathbf{s}_i^*$, and for $\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$ we use $\mathbf{s}_i$.

4.3.3 Warp Inversion

In Section 4.1.3 we showed that $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1} = \mathbf{W}(\mathbf{x};-\Delta\mathbf{p})$ to first order in $\Delta\mathbf{p}$. Replacing the warp $\mathbf{W}$ with the warp $\mathbf{N}\circ\mathbf{W}$ and the parameters $\Delta\mathbf{p}$ with the parameters $(\Delta\mathbf{q},\Delta\mathbf{p})$ yields:

$$\mathbf{N}\circ\mathbf{W}(\mathbf{x};\Delta\mathbf{q},\Delta\mathbf{p})^{-1} \;=\; \mathbf{N}\circ\mathbf{W}(\mathbf{x};-\Delta\mathbf{q},-\Delta\mathbf{p}) \tag{52}$$

to first order in $\Delta\mathbf{q}$ and $\Delta\mathbf{p}$.

4.3.4 Warp Composition

The first thing we need to do is compute the destination of the base mesh $\mathbf{s}_0$ under:

$$(\mathbf{N}\circ\mathbf{W})(\mathbf{x};\mathbf{q},\mathbf{p})\circ(\mathbf{N}\circ\mathbf{W})(\mathbf{x};\Delta\mathbf{q},\Delta\mathbf{p})^{-1} \;\approx\; (\mathbf{N}\circ\mathbf{W})(\mathbf{x};\mathbf{q},\mathbf{p})\circ(\mathbf{N}\circ\mathbf{W})(\mathbf{x};-\Delta\mathbf{q},-\Delta\mathbf{p}). \tag{53}$$

The destination of $\mathbf{s}_0$ under $(\mathbf{N}\circ\mathbf{W})(\mathbf{x};-\Delta\mathbf{q},-\Delta\mathbf{p})$ can be computed similarly to the computation for the warping above. First we compute:

$$\mathbf{W}(\mathbf{s}_0;-\Delta\mathbf{p}) \;=\; \mathbf{s}_0 - \sum_{i=1}^{n}\Delta p_i\mathbf{s}_i. \tag{54}$$

We then compute:

$$\mathbf{N}(\mathbf{s}_0;-\Delta\mathbf{q}) \;=\; \mathbf{s}_0 - \sum_{i=1}^{4}\Delta q_i\mathbf{s}_i^* \tag{55}$$

and an affine warp for $\mathbf{N}(\mathbf{s}_0;-\Delta\mathbf{q})$ using the technique described in Section 4.1.1. We then apply this affine warp to $\mathbf{W}(\mathbf{s}_0;\Delta\mathbf{p})$ to compute $\mathbf{N}\circ\mathbf{W}(\mathbf{s}_0;-\Delta\mathbf{q},-\Delta\mathbf{p})$. The destination of the base mesh $\mathbf{s}_0$ under $(\mathbf{N}\circ\mathbf{W})(\mathbf{x};\mathbf{q},\mathbf{p})$ was computed to perform the piecewise affine warp. We can then use the technique in Section 4.1.4 to combine $\mathbf{N}\circ\mathbf{W}(\mathbf{s}_0;-\Delta\mathbf{q},-\Delta\mathbf{p})$ and $\mathbf{N}\circ\mathbf{W}(\mathbf{s}_0;\mathbf{q},\mathbf{p})$ to compute the destination of the base mesh $\mathbf{s}_0$ under the warp in Equation (53). Denote the result $\mathbf{s}^\dagger$.

The second thing we need to do is find a new set of $\mathbf{p}$ and $\mathbf{q}$ such that:

$$\mathbf{N}\circ\mathbf{W}(\mathbf{s}_0;\mathbf{q},\mathbf{p}) \;=\; \mathbf{s}^\dagger. \tag{56}$$

In general solving this equation for $\mathbf{q}$ and $\mathbf{p}$ is a non-linear optimisation. For $\mathbf{N}$ a 2D similarity transform, however, the problem can be solved fairly easily. First note that:

$$\mathbf{N} \circ \mathbf{W}(\mathbf{s}_0; \mathbf{q}, \mathbf{p}) \;=\; \mathbf{N}(\mathbf{s}_0 + \sum_{i=1}^{n} p_i \mathbf{s}_i; \mathbf{q}). \tag{57}$$

Since $\mathbf{N}$ can also take the form in Equation (42), this expression equals:

$$\mathbf{N}(\mathbf{s}_0; \mathbf{q}) + \begin{pmatrix} (1+a) & -b \\ b & (1+a) \end{pmatrix} \sum_{i=1}^{n} p_i \mathbf{s}_i \tag{58}$$

where we have again abused the terminology. The multiple of the 2×2 matrix with the $2v$ dimensional shape vectors is performed by extracting each pair of matching xy vertex coordinates in turn, mutliplying by the 2×2 matrix, and then replacing. We can rewrite Equation (58) as:

$$\mathbf{N}(\mathbf{s}_0; \mathbf{q}) + \left[(1+a) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \sum_{i=1}^{n} p_i \mathbf{s}_i \right] + \left[b \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \sum_{i=1}^{n} p_i \mathbf{s}_i \right]. \tag{59}$$

The second[2] term in Equation (59) is orthogonal to $\mathbf{s}_i^*$ because $\mathbf{s}_i$ is orthogonal to $\mathbf{s}_i^*$. The third term in Equation (59) is orthogonal to $\mathbf{s}_i^*$ because, for every vector in $\mathbf{s}_i^*$ if we switch the role of x and y and change the sign of one of them we still have a vector that is (a constant multiple of) one of the other $\mathbf{s}_i^*$.

Since $\mathbf{N}(\mathbf{s}_0; \mathbf{q}) = \mathbf{s}_0 + \sum_{i=1}^{4} q_i \mathbf{s}_i^*$ we therefore wish to solve:

$$\mathbf{s}_0 + \sum_{i=1}^{4} q_i \mathbf{s}_i^* + \left[(1+a) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \sum_{i=1}^{n} p_i \mathbf{s}_i \right] + \left[b \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \sum_{i=1}^{n} p_i \mathbf{s}_i \right] \;=\; \mathbf{s}^\dagger \tag{60}$$

the solution of which for q_i, using the orthogonality relations discussed above, is:

$$q_i \;=\; \mathbf{s}_i^* \cdot (\mathbf{s}^\dagger - \mathbf{s}_0). \tag{61}$$

[2]This argument is only applicable to 2D similarity warps. A similar argument may also be possible for affine warps. For a general global warp $\mathbf{N}$, however, there is likely to be no corresponding derivation.

Once the parameters $\mathbf{q}$ are known we can then compute:

$$p_i \;=\; \mathbf{s}_i \cdot (\mathbf{N}(\mathbf{s}^\dagger; \mathbf{q})^{-1} - \mathbf{s}_0). \tag{62}$$

Note that these last two equations correspond to Equation (36) in Section 4.1.4.

4.3.5 Appearance Variation

The treatment of appearance variation is exactly as in Section 4.1.5. Solving for the warp $\mathbf{N} \circ \mathbf{W}$ rather than for $\mathbf{W}$ does not change anything except that the steepest descent images for both $\mathbf{p}$ and $\mathbf{q}$ need to be projected into $\mathrm{span}(A_i)^\perp$ using the equivalent of Equation (41).

The inverse compositional AAM fitting algorithm including appearance variation and global shape transform is summarised in Figure 13. In Step 5 we compute the modified steepest descent images:

$$\mathrm{SD}_j(\mathbf{x}) = \boldsymbol{\nabla} A_0 \frac{\partial \mathbf{N}}{\partial q_j} \;-\; \sum_{i=1}^{m} \left[\sum_{\mathbf{x} \in \mathbf{s}_0} A_i(\mathbf{x}) \cdot \boldsymbol{\nabla} A_0 \frac{\partial \mathbf{N}}{\partial q_j} \right] A_i(\mathbf{x}) \tag{63}$$

for each of the four normalised similarity parameters (q_1, q_2, q_3, q_4) and:

$$\mathrm{SD}_{j+4}(\mathbf{x}) = \boldsymbol{\nabla} A_0 \frac{\partial \mathbf{W}}{\partial p_j} \;-\; \sum_{i=1}^{m} \left[\sum_{\mathbf{x} \in \mathbf{s}_0} A_i(\mathbf{x}) \cdot \boldsymbol{\nabla} A_0 \frac{\partial \mathbf{W}}{\partial p_j} \right] A_i(\mathbf{x}) \tag{64}$$

for $\mathbf{p}$ where $j = 1, \ldots, n$. The concatenated steepest descent images form a single vector with four images for $\mathbf{q}$ followed by n images for $\mathbf{p}$. If we denote the elements of this vector $\mathrm{SD}_j(\mathbf{x})$ for $j = 1, \ldots, n+4$, the $(j,k)^{\mathrm{th}}$ element of the $(n+4) \times (n+4)$ Hessian matrix is then computed in Step 6 as:

$$\mathbf{H}_{j,k} = \sum_{\mathbf{x} \in \mathbf{s}_0} \mathrm{SD}_j(\mathbf{x}) \cdot \mathrm{SD}_k(\mathbf{x}). \tag{65}$$

We compute the appearance parameters in Step (10) using:

$$\lambda_i \;=\; \sum_{\mathbf{x} \in \mathbf{s}_0} A_i(\mathbf{x}) \cdot \left[I(\mathbf{N}(\mathbf{W}(\mathbf{x}; \mathbf{p}); \mathbf{q})) - A_0(\mathbf{x}) \right]. \tag{66}$$

Inverse Compositional Algorithm with Appearance Variation and Global Shape Transform

Pre-compute:

(3) Evaluate the gradient ∇A_0 of the template $A_0(\mathbf{x})$
(4) Evaluate the Jacobians $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ and $\frac{\partial \mathbf{N}}{\partial \mathbf{q}}$ at $(\mathbf{x}; \mathbf{0})$
(5) Compute the modified steepest descent images using Equations (63) and (64)
(6) Compute the Hessian matrix using Equation (65)

Iterate:

(1) Warp I with $\mathbf{W}(\mathbf{x}; \mathbf{p})$ followed by $\mathbf{N}(\mathbf{x}; \mathbf{q})$ to compute $I(\mathbf{N}(\mathbf{W}(\mathbf{x}; \mathbf{p}); \mathbf{q}))$
(2) Compute the error image $I(\mathbf{N}(\mathbf{W}(\mathbf{x}; \mathbf{p}); \mathbf{q})) - A_0(\mathbf{x})$
(7) Compute $\sum_{\mathbf{x} \in \mathbf{s}_0} \mathrm{SD}_i(\mathbf{x}) \cdot [I(\mathbf{N}(\mathbf{W}(\mathbf{x}; \mathbf{p}); \mathbf{q})) - A_0(\mathbf{x})]$ for $i = 1, \ldots, n+4$
(8) Compute $(\Delta \mathbf{q}, \Delta \mathbf{p})$ by multiplying the resulting vector by the inverse Hessian
(9) Update $(\mathbf{N} \circ \mathbf{W})(\mathbf{x}; \mathbf{q}, \mathbf{p}) \leftarrow (\mathbf{N} \circ \mathbf{W})(\mathbf{x}; \mathbf{q}, \mathbf{p}) \circ (\mathbf{N} \circ \mathbf{W})(\mathbf{x}; \Delta \mathbf{q}, \Delta \mathbf{p})^{-1}$

Post-computation:

(10) Compute λ_i using Equation (66). [Optional step]

Figure 13: The inverse compositional AAM fitting algorithm with appearance variation and global shape transform. Most of the computation is similar to that in Figure 12 for the inverse compositional algorithm with appearance variation, except there are $n+4$ shape parameters, 4 in $\mathbf{q}$ and n in $\mathbf{p}$. We therefore compute $n+4$ modified steepest descent images in Step 5, a $(n+4) \times (n+4)$ Hessian matrix in Step 6, and $n+4$ steepest descent parameter updates in Step 7. These updates are formed into a $n+4$ dimensional vector and multiplied by the inverse of the Hessian in Step 8 to give the $n+4$ dimensional vector $(\Delta \mathbf{q}, \Delta \mathbf{p})$.

4.4 Other Extensions to the Algorithm

We have described how the inverse compositional image alignment algorithm can be applied to AAMs. The field of image alignment is well studied and over the years a number of extensions and heuristics have been developed to improve the performance of the algorithms. Most of these can easily be applied to the algorithm that we have just described. Three examples include:

Hierarchical Processing: The fitting algorithm can be applied hierarchically on a Gaussian image pyramid to reduce the likelihood of falling into a local minimum [4].

Progressive Transformation Complexity: The fitting algorithm can be applied incrementally to more and more complex warps; i.e. first fit on a small number of the shape parameters $(p_1, \ldots, p_n)^{\mathrm{T}}$ and then incrementally add more and more complexity [4].

Levenberg-Marquardt: It is possible to use the Levenberg-Marquardt inverse compositional algorithm instead of the Gauss-Newton inverse compositional algorithm described in this paper. See [3] for the details of that algorithm.

5 Empirical Evaluation

We have proposed a new fitting algorithm for AAMs. The performance of AAM fitting algorithms depends on a wide variety of factors. For example, it depends on whether hierarchical processing, progressive transformation complexity, and adaptive step-size algorithms such as Levenberg-Marquardt are used. See Section 4.4. The performance can also be very dependent on minor details such as the definition of the gradient filter used to compute ∇A_0. Comparing like with like is therefore very difficult. Another thing that makes empirical evaluation hard is the wide variety of AAM fitting algorithms [6, 7, 11, 18, 21] and the lack of a standard test set.

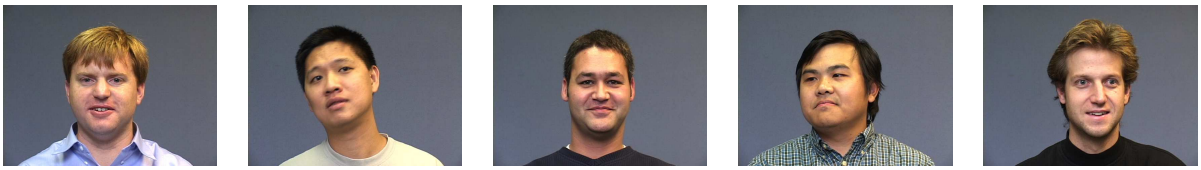
In our evaluation we take the following philosophy. Instead of comparing our algorithm with the original AAM fitting algorithm or any other algorithm (the results of which would have limited meaning), we set up a collection of systematic experiments where we only vary one component of the algorithm. In this paper, we have discussed three main changes to AAM fitting:

1. We use the inverse compositional warp update rather than the additive; i.e. we use Step 9 in Figure 13 to update the warp parameters rather than simply updating $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$.
2. We use an analytical derivation of the steepest descent images rather than a numerical approximation [11, 15, 21].
3. We project out the appearance variation as described in Section 4.1.5 rather than fitting for it as in [11, 15, 21].

Each of these changes can be made independently and so we evaluate each one independently. We compare our AAM fitting algorithm with and without each of these changes. For example, in the first experiment we try the two alternatives for Step 9 of the algorithm and compare them. Every other line of code in the implementation is exactly the same.

5.1 Constructing the AAMs

One choice we need to make to test the algorithms is the content of the AAM. There are a wide variety of applications for AAMs. An AAM could be constructed for a specific person and used to track that person's face. Alternatively, a generic AAM could be constructed using images of a



Training 1 Training 2 Training 3 Training 4 Training 5

Figure 14: Five of the 110 training images used to construct the 5-person AAM. Figures 1 and 2 contain the first few modes of the shape and appearance variation of the resulting AAM.

variety of people's faces and might be used for face recognition. Although there is no reason to expect the *relative* performance of the algorithms to depend on the AAM used, we present results using two different AAM models to validate our conclusions more fully. Specifically, we chose to use a person specific AAM and a multi-person AAM constructed for 5-people.

Five of the 110 training images used to construct the 5-person AAM are included in Figure 14. The mean shape and first 3 (of 13) shape modes for this model are shown in Figure 1. The mean appearance and first 3 (of 42) appearance modes are shown in Figure 2. Figure 15 contains 4 of the 30 training images used to construct the person-specific AAM, the mean shape, the 3 shape modes, the mean appearance, and the first 3 (of 9) appearance modes.

5.2 Test Data, Ground Truth, and Experimental Procedure

We evaluate both of the AAMs on a collection of images. The person specific AAM was evaluated on 300 frames. The 5-person AAM was evaluated on 900 frames, 180 frames for each of the five people. The accompanying files 'test_video_ps.mpg' and 'test_video_5p.mpg' contain the test sequences. Note that although the test data is arranged as videos, the evaluation of the algorithms treats each frame as a separate test case and is an evaluation of single-frame fitting performance.

To generate ground truth data, we fit the appropriate AAM to each frame of the test data. This is achieved through a combination of: (1) hand initialisation and re-initialisation, and (2) tracking through the sequences. We visually check that the fit and the reconstruction is good for each frame in the test data. The result is illustrated in Figure 16 and the accompanying movies 'ground_truth_ps.mpg' and 'ground_truth_5p.mpg'. Ideally we want to use these fitting results as

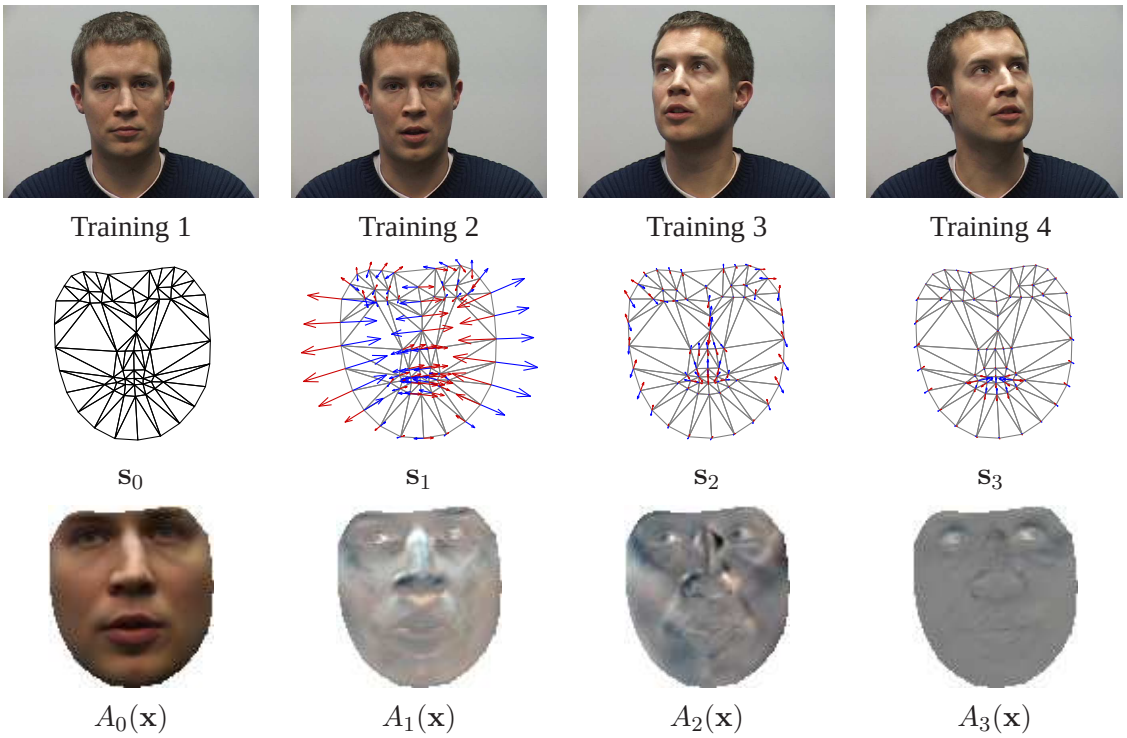


Figure 15: The person-specific AAM. Top Row: Four of the 30 training images used to construct the AAM. Middle Row: The mean shape and all 3 shape variation modes. Bottom Row: The mean appearance and the first 3 (of 9) appearance variation modes.

ground-truth. This, however, is problematic because it may bias the results towards the algorithm used to generate the ground truth. Specifically, we used the inverse compositional algorithm with analytically computed steepest descent images and projected-out appearance variation to generate the ground-truth. To detect any possible bias in the ground-truth we create a new sequence by overlaying the reconstructed AAM on the original movie. We therefore have four movies. The original movies, and two new synthetic movies containing the background of the original movies and the face region synthetically generated from the fitted AAM parameters. By comparing the performance of the algorithms on these sequences we should be able to detect any bias in the ground-truth. A side-by-side comparison of the real and synthetic movies is contained in the files 'synthetic_vs_real_ps.mpg' and 'synthetic_vs_real_5p.mpg'. Empirically we found no difference in the relative performance of any of the algorithms on the corresponding real and synthetic data, and so conclude that there is no bias. For lack of space, we just present the results on the synthetic data. The results on the original data are almost identical and can be obtained on the authors webpages.

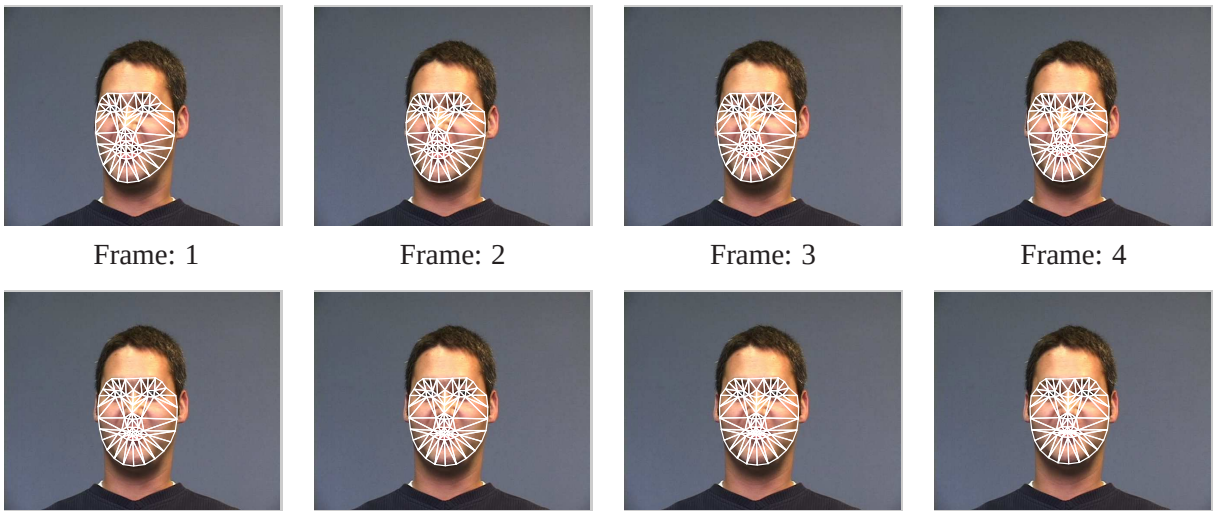


Figure 16: Example frames from the 5-person test sequence overlaid with the ground truth mesh. The ground truth result is used in two ways: (1) to create the model reconstructed synthetic test sequence, and (2) to define the initial conditions for all of the perturbation experiments.

Also note that the convergence plots in Figures 19, 20, 21, and 22 show that all of the algorithms converge almost exactly to the ground truth, at least some of the time. This provides even more evidence that there is no bias in the ground-truth.

Our experimental procedure consists of running the algorithms on a large number of inputs, evaluating the results, and averaging. Each input consists of: (1) one of the images from the appropriate test data, and (2) the ground truth shape, appearance, and similarity parameters. Each input test case is then generated as follows. Given the test image, the ground-truth parameters for that image are randomly perturbed to generate the initial parameter estimates to start the algorithms with. The shape parameters are randomly generated from independent Gaussian distributions with variance equal to a multiple of the eigenvalue of that mode in the PCA performed during AAM construction. The similarity transform parameters are generated using the same procedure as in [3]. Two distinguished points[3] in the mesh are perturbed with Gaussian noise of a certain variance and the similarity transform parameters then solved for. Finally, the appearance parameters are set to

[3]The motion of two points defines a similarity warp. The experiments in [3] use an affine warp and so three points are required. The amount of perturbation is then sampled by varying the variance. Other approaches to perturbing the similarity warp are possible. For example, we could perturb the mesh points with equal magnitude perturbations to the orthonormalised 4 eigenvectors used to implement the similarity transformation. The high-level meaning of doing this is unclear, however.

be the mean appearance.

The results presented in the following sections are the result of averaging the performance over 20 randomly generated trials for each of the 300/900 frames in the test data. Figure 17 shows four example random perturbations from the ground truth. Six frames from one of the example trials are shown in Figure 18 for the inverse compositional algorithm. The algorithm converges well before 20 iterations. More example trials for the person-specific AAM are illustrated in 'trials_ps.mpg'.

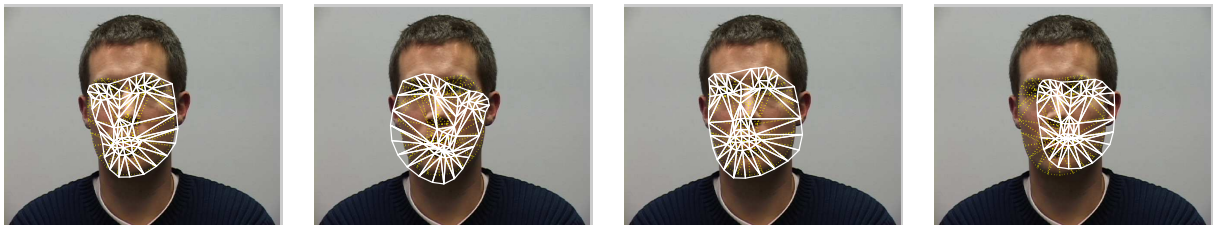


Figure 17: Examples of perturbed initial conditions. The ground truth result (dotted mesh) is randomly perturbed by a chosen variance to form the starting point (solid mesh). The algorithms are then run for 20 iterations and tested for convergence.

Figure 18: Six frames from the 20 iterations of a perturbation trial. In the initial frame the total mesh point location error from the ground truth mesh is 21.8 pixels. By the 10th iteration the error is ≤ 1.0 and so the trial has already passed the convergence test.

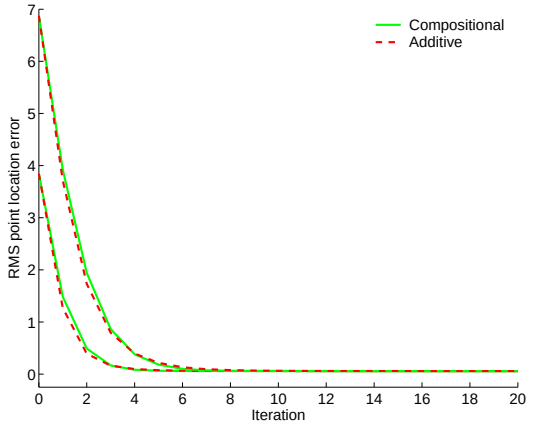
5.3 The Evaluation Metrics

Given the initial, perturbed, parameters, the AAM fitting algorithm should hopefully converge to the ground-truth parameters. We measure this convergence in two ways. The first measure is the *average rate of convergence.* We plot the RMS error in the mesh point locations against the iteration number of the algorithm. If the algorithm converges the RMS error should reduce to close to zero. These graphs are averaged (for approximately the same starting error) over all cases where all algorithms converge. We say that an algorithm converges if the RMS mesh point error is less than 1.0 pixels after 20 iterations. The second measure is the *average frequency of convergence.* We count the number of times each algorithm has converged (after 20 iterations), divide by the total number of trials, and multiply by 100 to convert to a percentage.

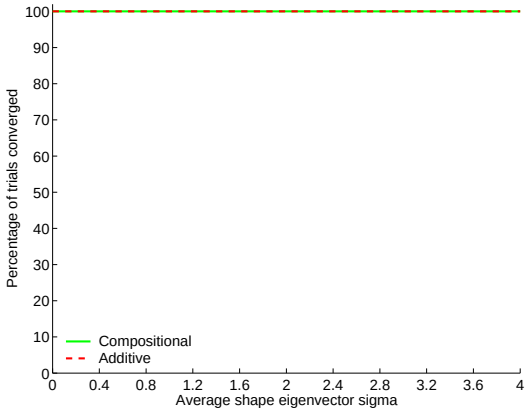
We do not measure the accuracy of the appearance parameters because once the shape parameters have been estimated, estimating the appearance parameters is a simple linear operation. See Equation (40). Also, comparing the appearance algorithms by their appearance estimates is not possible because the appearance is not computed until the "project out" algorithm has converged.

5.4 Experiment 1: The Update Rule

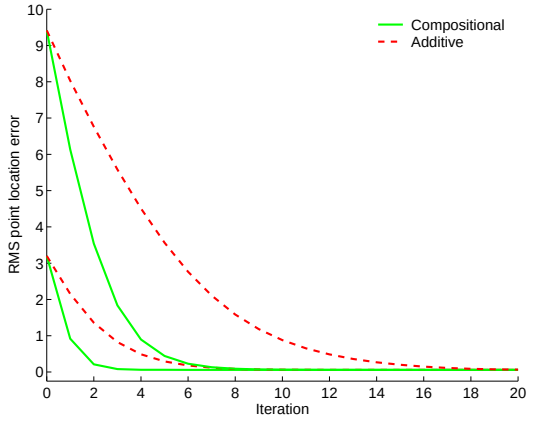
In the first experiment we compare the inverse compositional update in Step 9 of the algorithm with the usual additive update of: $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$. (Because the roles of the image and the template were switched in the derivation of the inverse compositional algorithm, we actually update $\mathbf{p} \leftarrow \mathbf{p} - \Delta\mathbf{p}$.) All previous AAM fitting algorithms, including [7, 11, 21] use the additive update. The results for the person-specific AAM are included in Figure 19 and those for the 5-person AAM in Figure 20. For both figures in (a) and (b) we plot results obtained by perturbing only the shape parameters, in (c) and (d) we plot results perturbing only the similarity transform parameters, and in (e) and (f) we plot results perturbing both the shape and similarity parameters. The weighting between the shape and similarity transform parameters for the "both" case was chosen so that the knee points in the frequency of convergence plots for shape and similarity occur at roughly the same point.



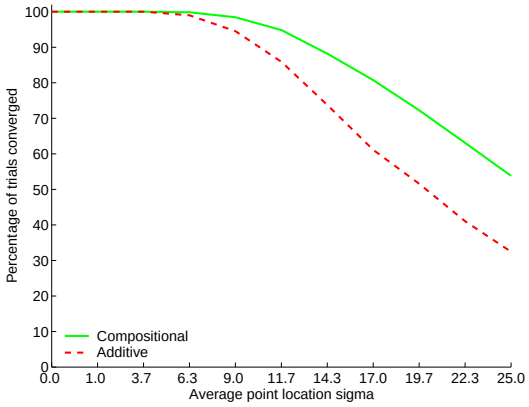
(a) Rate of Convergence Perturbing Shape



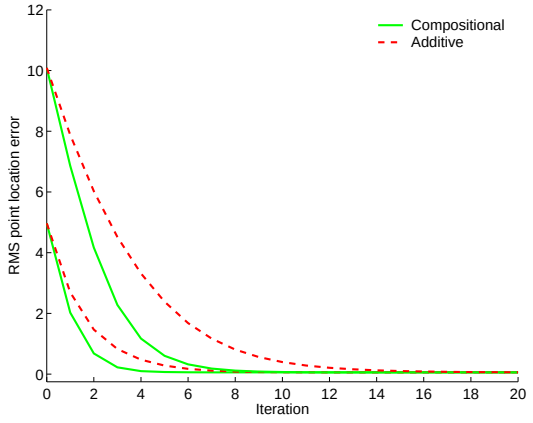
(b) Frequency of Convergence Perturbing Shape



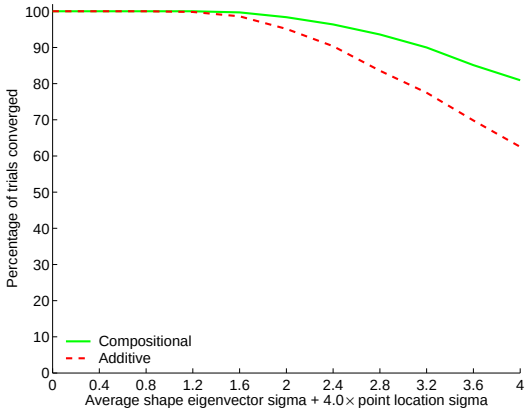
(c) Rate of Convergence Perturbing Similarity



(d) Frequency of Convergence Perturbing Similarity

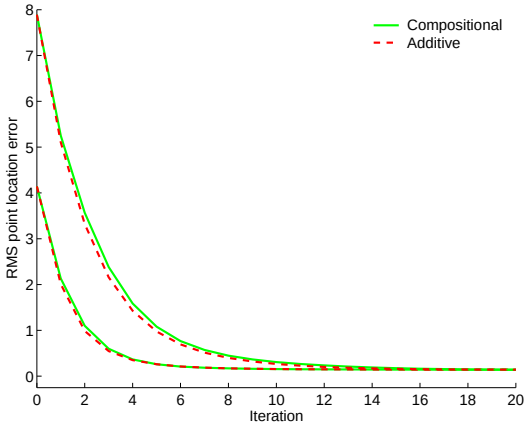


(e) Rate of Convergence Perturbing Both

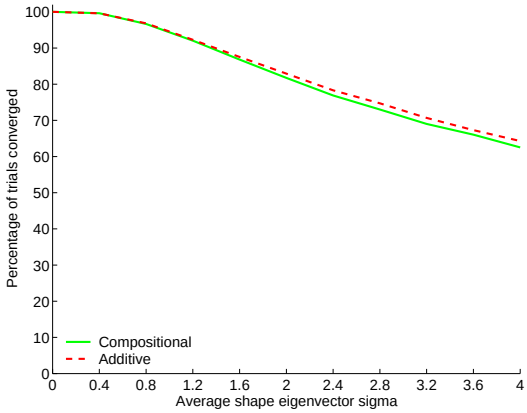


(f) Frequency of Convergence Perturbing Both

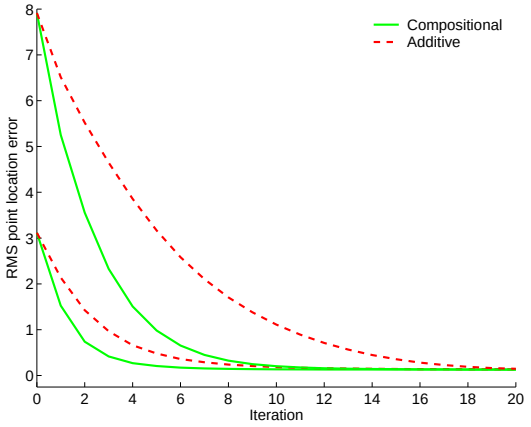
Figure 19: Person-Specific AAM Results: The results of comparing the additive and inverse compositional updates to the warp in Step 9 of the algorithm. We plot the rate of convergence in (a), (c), and (e), and the frequency of convergence in (b), (d), and (f). Two curves are shown for each algorithm in the rate of convergence plots in (a), (c), and (e) for two different sets of the initial parameter estimates corresponding to two different degrees of perturbation to the ground-truth parameters. In (a) and (b) we just perturb the shape parameters, in (c) and (d) we just perturb the similarity transform parameters, and in (e) and (f) we perturb both sets of parameters. In general, the compositional update outperforms the additive update.



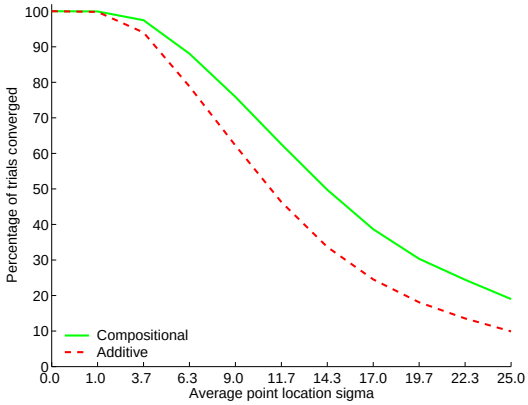
(a) Rate of Convergence Perturbing Shape



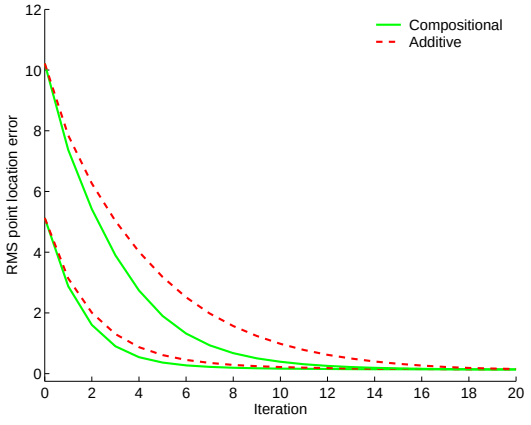
(b) Frequency of Convergence Perturbing Shape



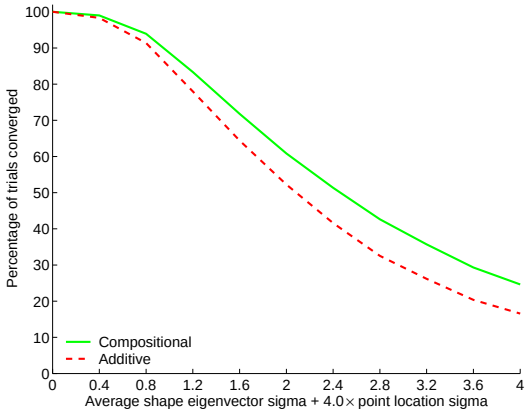
(c) Rate of Convergence Perturbing Similarity



(d) Frequency of Convergence Perturbing Similarity



(e) Rate of Convergence Perturbing Both



(f) Frequency of Convergence Perturbing Both

Figure 20: 5-Person AAM Results: The results of comparing the additive and inverse compositional updates to the warp in Step 9 of the algorithm. We plot the rate of convergence in (a), (c), and (e), and the frequency of convergence in (b), (d), and (f). Two curves are shown for each algorithm in the rate of convergence plots in (a), (c), and (e) for two different sets of the initial parameter estimates corresponding to two different degrees of perturbation to the ground-truth parameters. In (a) and (b) we just perturb the shape parameters, in (c) and (d) we just perturb the similarity transform parameters, and in (e) and (f) we perturb both sets of parameters. In general, the compositional update outperforms the additive update.

The first thing to notice in Figures 19 and 20 is that in most cases the inverse compositional update outperforms the additive. The rate of convergence is faster and the frequency of convergence is higher. The one exception is for perturbing only the shape component in (a) and (b). In this case, the global similarity transform hides the problem described in Section 2.3.3 and so the addition and inverse composition updates perform very similarly. When the normalising similarity transformation is used, the shape variation is very small relative to the mesh. The additive update is therefore a reasonable approximation for the shape parameters. The constant linear update approximation problem is clear for the similarity transform parameters in (c) and (d), and the combined performance for perturbing both shape and similarity in (e) and (f) is also poor.

Comparing Figures 19 and 20 it is clear that the performance for the person-specific AAM is far better than the performance for the 5-person AAM. This is expected because of the greater shape and appearance variation in the 5-person AAM. Note however, that the relative performance of the algorithms is the same in both cases.

In the following experiments we only show results for perturbing both shape and similarity transform parameters. In all cases the "both" results are representative of those obtained for perturbing shape or similarity transform parameters alone.

5.5 Experiment 2: Computation of the Steepest Descent Images

The inverse compositional algorithm uses the *steepest descent images* $\nabla A_0 \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$. The analytically derived steepest descent images play exactly the same role as the numerically estimated images $\frac{\partial \mathbf{r}}{\partial \mathbf{p}}$ in [11] (and the corresponding finite difference images in [15, 21].) The difference between these two sets of images is that in the additive formulation, the steepest descent images are a function of $\mathbf{p}$ whereas in the inverse compositional formulation they are provably constant. Otherwise their roles and meaning are identical.

Is it better to analytically compute or numerically estimate the steepest descent images? To answer this question, we apply the same inverse compositional algorithm using both approaches, the analytically derived $\nabla A_0 \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$, and the numerically estimated $\frac{\partial \mathbf{r}}{\partial \mathbf{p}}$ computed as described in [11].

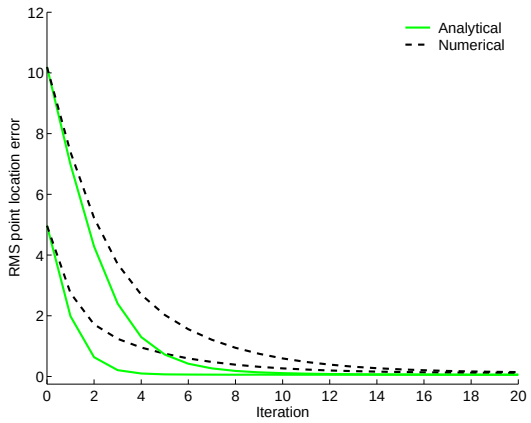
Because we use the inverse compositional update, the steepest descent images in both cases should be evaluated at $\mathbf{p} = \mathbf{0}$. The only change to the numerical estimate computation described in [11] then is that we always perturb from a starting point of $\mathbf{p} = \mathbf{0}$.

The results in Figure 21 show that the analytically derived steepest descent images perform better for both the person-specific and 5-person AAMs. This is most clearly seen in the 5-person case where the numerical estimate results consistently converge away from the ground truth and therefore the frequency of convergence is very poor. One reason for this difference may be that the computation of the analytically derived steepest descent images does not depend on the background. We use a 3×3 plane fitting algorithm to estimate the gradient of the template ∇A_0 that can cope with arbitrary "missing" pixels at the boundary of the base mesh. In the numerical case it is not clear how to best minimise background difference effects at the boundary.

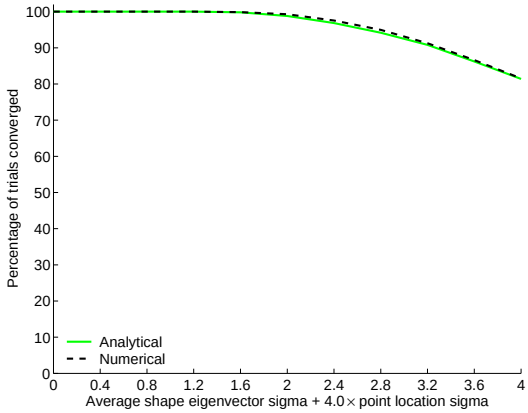
5.6 Experiment 3: Appearance Variation

In our algorithm we "project out" the appearance variation. See Section 4.3.5. Projecting out the appearance has the advantage that there are less unknowns and so the algorithm is more efficient. This approach should be contrasted with that in [11, 15, 21] where the appearance parameters are treated like any other and are iteratively updated. There are potentially benefits of iteratively updating the appearance parameters. In particular, any correlation between the appearance and shape parts of the model can be accounted for in the Hessian matrix. It is straightforward to modify the inverse compositional algorithm to model the appearance variation in this way. The steepest descent images must be computed for the parameters of appearance part of the model. The steepest descent image for the appearance parameter λ_i is simply $A_i(\mathbf{x})$. These steepest descent images are then appended to the vector of steepest descent images, the Hessian increases in size appropriately, and otherwise the algorithm remains the same. The additive approach is the correct way to update the appearance parameters. See [1] for more details of the inverse compositional algorithm with linear appearance variation.

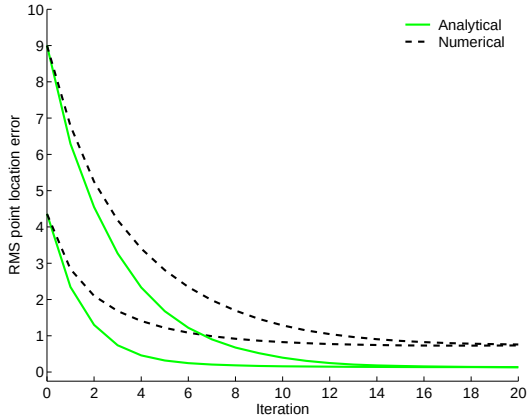
We plot results comparing the "project out appearance" approach with the "explicitly modeling



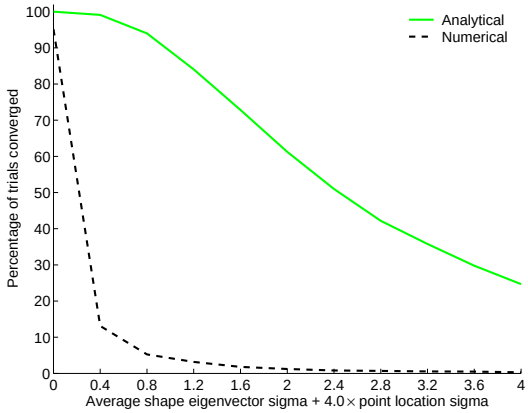
(a) Rate of Convergence: Person-Specific AAM



(b) Frequency of Convergence: Person-Specific AAM



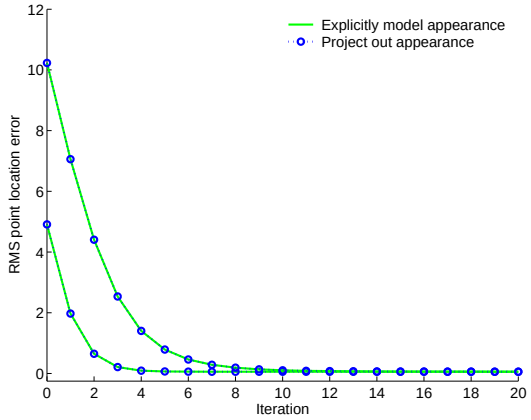
(c) Rate of Convergence: 5-Person AAM



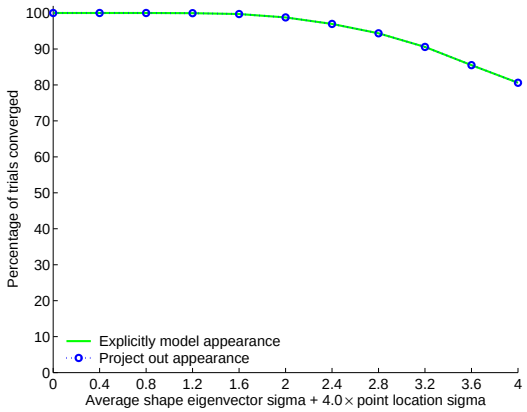
(d) Frequency of Convergence: 5-Person AAM

Figure 21: A comparison of using the analytically derived steepest descent images versus the numerical finite difference images used in [11, 15, 21]. The results show that the analytically derived images perform significantly better, especially in the 5-person case. The two sets of curves in the rate of convergence plot correspond to two different magnitudes of the initial perturbation.

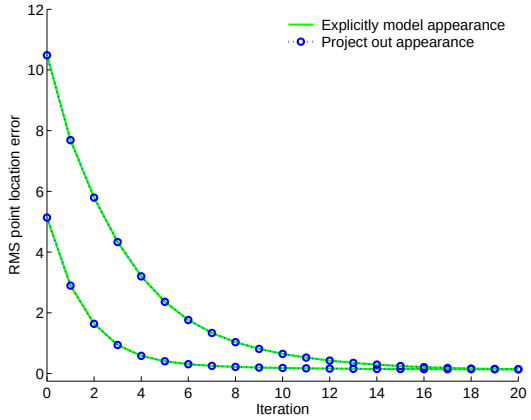
appearance" approach in Figure 22. Again, all other aspects of the two algorithms are identical. Both use the inverse compositional update and the analytical steepest descent images. The results in Figure 22 show that the two approaches perform identically for both AAM models. Note, however, that the project out approach is far more efficient because less parameters are updated in each iteration of the algorithm. The "project out appearance" approach is therefore the better choice. Note, however, that there are possibly better, but far slower, ways to solve for the appearance variation [1].



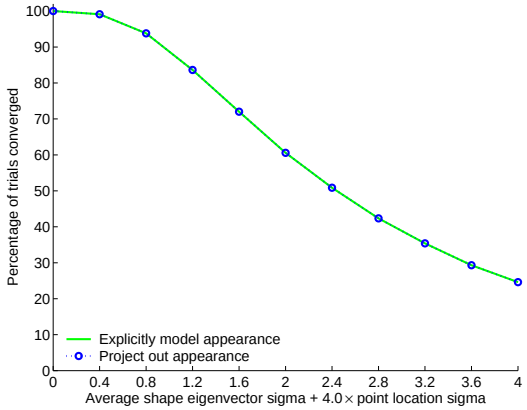
(a) Rate of Convergence: Person-Specific AAM



(b) Frequency of Convergence: Person-Specific AAM



(c) Rate of Convergence: 5-Person AAM



(d) Frequency of Convergence: 5-Person AAM

Figure 22: A comparison of the "project out appearance" approach described in Section 4.3.5 with the usual approach of "explicitly modelling appearance" used in [11]. The results show no difference in the performance of the two approaches. The "project out appearance" approach is therefore the better choice because it is far more computationally efficient. The two sets of curves in the rate of convergence plot correspond to two different magnitudes of the initial perturbation.

5.7 Computational Efficiency

One possible concern with the inverse compositional algorithm is that the time taken to perform the warp update in Step 9 might be quite long. Although the description of the update is quite involved, the actual calculation is minimal because the number of vertices in the mesh is far less than the number of pixels. In Table 1 we include timing results for our Matlab implementations of the four algorithms compared in this section. "With Additive Update" is the algorithm compared with in Section 5.4: the inverse compositional algorithm with Step 9 replaced with the additive update. "With Numerical Gradient" is the algorithm compared with in Section 5.5: the inverse compositional algorithm with a numerical estimate of the steepest descent images. Finally, "Explicitly

Table 1: Timing results for our Matlab implementations of the four algorithms evaluated in Section 5 in *milliseconds*. These results were obtained on a dual 2.4GHz P4 machine and are for the person-specific AAM with 19,977 pixels, 3 shape parameters, 4 similarity transform parameters, and 9 appearance parameters. Note that: (1) the inverse compositional algorithm is not significantly slower than the additive algorithm, (2) the pre-computation of the numerical steepest descent images is slow, and (3) explicitly modeling appearance is significantly slower than projecting out appearance.

Pre-computation:

	Step 3	Step 4	Step 5	Step 6	Total
Inverse Compositional	12,682	5,726	90.2	85.2	18,583
With Additive Update	12,682	5,726	90.2	85.2	18,583
With Numerical Gradient	-	-	251,012	85.2	251,097
Explicitly Modeling Appearance	12,682	5,726	34.3	435.5	18,878

Per Iteration:

	Step 1	Step 2	Step 7	Step 8	Step 9	Total
Inverse Compositional	2.7	3.0	12.0	0.1	0.2	17.8
With Additive Update	2.7	3.0	12.0	0.1	0.1	17.7
With Numerical Gradient	2.7	3.0	12.0	0.1	0.2	17.8
Explicitly Modeling Appearance	2.7	6.4	26.7	0.1	0.3	36.2

Post Computation:

	Step 10
Inverse Compositional	7.2
With Additive Update	7.2
With Numerical Gradient	7.2
Explicitly Modeling Appearance	-

Modeling Appearance" is the algorithm compared with in Section 5.6: the inverse compositional algorithm extended to compute the appearance parameters using gradient descent.

The results in Table 1 were obtained on a dual 2.4GHz P4 machine and are for the person-specific AAM with 19,977 pixels, 3 shape parameters, 4 similarity transform parameters, and 9 appearance parameters. The results for the 5-person AAM are similar and omitted for lack of space. As can be seen, the inverse compositional update in Step 9 is negligible compared with the other steps and so the algorithm is only marginally slower than the additive algorithm. Also note: (1) that the analytical computation of the steepest descent images in Steps 3, 4, and 5 is significantly faster than the numerical computation and (2) the explicitly modeling appearance

algorithm is significantly slower primarily because there are more parameters to solve for in each iteration. The computation of the error image in Step 2 is also more involved. We also have a compiled implementation of our algorithm written in "C". This implementation runs at 230 frames per second for the person-specific AAM.

6 Conclusion

Previous AAM fitting algorithms fall into one of two categories: (1) inefficient gradient descent algorithms and (2) efficient, but ad-hoc, algorithms that make the approximation that there is a constant linear relationship between the error image and the additive updates to the (shape) parameters. In this paper we have proposed an algorithm for fitting AAMs that has the advantages of both types of algorithms. It is an analytical gradient descent algorithm with well understood convergence properties that is even more efficient than the ad-hoc algorithms. The algorithm is an extension to the inverse compositional image alignment algorithm [3]. Overall our algorithm outperforms previous approaches in terms of: (1) the speed of convergence (fewer iterations are needed to converge to any given accuracy), (2) the frequency of convergence (our algorithm is more likely to convergence from a large distance away), and (3) the computational cost (the algorithm is faster mainly because the appearance variation is projected out.)

The inverse compositional AAM fitting algorithm can only be applied to *independent* AAMs. It cannot be applied to *combined* AAMs (see Section 2.2 for a definition of independent and combined AAMs) which parameterize the shape and appearance variation with a single set of parameters and so introduce a coupling between shape and appearance. Although this may seem like a serious limitation because combined AAMs can parameterize the same visual phenomenon with fewer parameters, in practice it is not. The nonlinear optimisation in our algorithm is only over the n shape parameters and so is actually lower dimensional than the equivalent combined AAM optimisation which would have more than n parameters. Currently we do not see an easy way to extend our algorithm to combined AAMs, but of course we may be wrong.

The inverse compositional algorithm is an image alignment algorithm. Hence most standard extensions to image alignment can be applied with this algorithm. See Section 4.4. For other extensions, such as the use of robust norms [5, 16, 21], the situation is not quite so clear. For example, when a robust norm is used, the orthogonal decomposition of the norm in Equation (38) is not applicable. The efficient treatment of appearance is therefore more complex [1].

Acknowledgments

We thank Tim Cootes for discussions on the incorporation of the global shape transformation in Section 4.3. Elements of the AAM fitting algorithm appeared in [2]. We thank the reviewers of [2] and this IJCV paper for their feedback. The research described in this paper was conducted under U.S. Office of Naval Research contract N00014-00-1-0915. Additional support was provided through U.S. Department of Defense award N41756-03-C4024.

References

[1] 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.

[2] S. Baker and I. Matthews. Equivalence and efficiency of image alignment algorithms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 1090–1097, 2001.

[3] S. Baker and I. Matthews. Lucas-Kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, Feb. 2004. Previously appeared as CMU Robotics Institute Technical Report CMU-RI-TR-02-16.

[4] J. Bergen, P. Anandan, K. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *Proceedings of the European Conference on Computer Vision*, pages 237–252, 1992.

[5] 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.

[6] 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.

[7] T. Cootes, G. Edwards, and C. Taylor. Active appearance models. In *Proceedings of the European Conference on Computer Vision*, volume 2, pages 484–498, 1998.

[8] T. Cootes, G. Edwards, and C. Taylor. A comparitive evaluation of active appearance models algorithms. In *Proceedings of the British Machine Vision Conference*, 1998.

[9] T. Cootes and P. Kittipanya-ngam. Comparing variations on the active appearance model algorithm. In *Proceedings of the British Machine Vision Conference*, volume 2, pages 837–846, 2002.

[10] T. F. Cootes. Statistical models of appearance for computer vision. Online technical report available from http://www.isbe.man.ac.uk/˜bim/refs.html, Sept. 2001.

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

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

[13] G. J. Edwards. *Learning to Identify Faces in Images and Video Sequences*. PhD thesis, University of Manchester, Division of Imaging Science and Biomedical Engineering, 1999.

[14] G. J. Edwards, C. J. Taylor, and T. F. Cootes. Interpreting face images using active appearance models. In *Proc. International Conference on Automatic Face and Gesture Recognition*, pages 300–305, June 1998.

[15] M. Gleicher. Projective registration with difference decomposition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 331–337, 1997.

[16] 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.

[17] X. Hou, S. Li, H. Zhang, and Q. Cheng. Direct appearance models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 828–833, 2001.

[18] M. Jones and T. Poggio. Multidimensional morphable models: A framework for representing and matching object classes. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 683–688, 1998.

[19] A. Lanitis, C. J. Taylor, and T. F. Cootes. Automatic interpretation and coding of face images using flexible models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):742–756, 1997.

[20] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

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

[22] S. Sclaroff and J. Isidoro. Active blobs: region-based, deformable appearance models. *Computer Vision and Image Understanding*, 89(2/3):197–225, Feb. 2003.

[23] H.-Y. Shum and R. Szeliski. Construction of panoramic image mosaics with global and local alignment. *International Journal of Computer Vision*, 16(1):63–84, 2000.

[24] T. Vetter and T. Poggio. Linear object classes and image synthesis from a single example image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):733–742, 1997.