2D Statistical Models

JIA Pei

Email: jp4work@gmail.com

Vision Open Working Group

Second Edition

October 21, 2010

Contents

Chapter 1

Principal Component Analysis

Principal component analysis (PCA) has been ubiquitously applied in many data analysis and signal precessing areas ([10]). It is the most successful statistical model to represent deformable objects, both in 2D and 3D. The primary purpose of PCA is for data compaction, or dimensionality reduction. It tries to find in cascade a set of uncorrelated components to retain as much sample variance as possible, which means it firstly finds a principle component which accounts for the biggest variability in the data, then it looks for succeeding components which not only accounts for as much of the remaining variability as possible, but also perpendicular to the former chosen components. These uncorrelated components are named principal components, which are proved to be corresponding to the biggest eigenvalues and eigenvectors of the variance-covariance matrix of the sample data.

An intuitionistic explanation of PCA is depicted in figure 1.1. Clearly, the scattered sample data have different variances in different directions. The long axis shows the direction with the biggest variance and the short axis is the direction perpendicular to the long axis. In this case, PCA will first look for the long axis as the first principal component and then the short one.

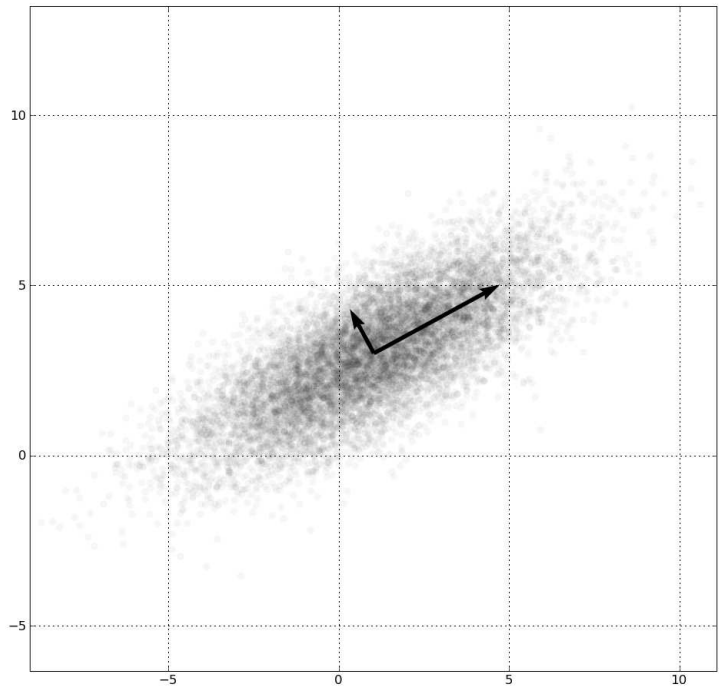


Figure 1.1: Principal Component Analysis (cited from [26])

Chapter 2

2D Statistical Models

2.1 2D Shape Model

The statistical *shape* model for a 2D object is defined by the 2D coordinates of v manually labeled points [5], which actually compose a 2D triangulation mesh.

$$\mathbf{s} = \begin{pmatrix} x_1 & x_2 & \cdots & x_v \\ y_1 & y_2 & \cdots & y_v \end{pmatrix} \tag{2.1}$$

Every manually labeled point is defined as:

$$\mathbf{x_i} = (x_i, y_i)^T,\ i = 1, 2, \cdots, v \tag{2.2}$$

For the actual implementation, a 2D shape model is always reshaped to

$$\mathbf{s} = (x_1, x_2, \cdots, x_v, y_1, y_2, \cdots, y_v) \tag{2.3}$$

The shape statistical model allows linear shape variation. This means an arbitrary shape $\mathbf{s}$ can be looked on as a base shape $\mathbf{s}_0$ plus a linear combination of N shape vectors $\mathbf{s}_i$:

$$\mathbf{s} = \mathbf{s}_0 + \sum_{i=1}^{N} p_i \mathbf{s}_i = \mathbf{s}_0 + \mathbf{p}\mathbf{P}_\mathbf{s} \tag{2.4}$$

where p_i are the shape parameters and $\mathbf{s}_i$ are orthonormal eigenvectors, with $\mathbf{P_s} = \begin{pmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \cdots \\ \mathbf{s}_N \end{pmatrix}$ and $\mathbf{p} = (p_1 \quad p_2 \quad \cdots \quad p_N)$.

An arbitrary shape $\mathbf{s}$ might have various triangulation structures due to different rules. Here in our application, Delaunay Triangulation [21] is implemented on the mean shape $\mathbf{s}_0$ to define the shape's inner triangulation structure, which is absolutely both existing and unique.

For all training shapes $\mathbf{s}^j, j = 1, 2, \cdots, F$, each shape $\mathbf{s}^j$ is corresponding to a vector of shape parameters $\mathbf{p}^j$, F is the number of training shapes. For simplicity, all ith elements of all shape parameters $\mathbf{p}^j$, namely, the set $\{p_i^1, p_i^2, \cdots, p_i^F\}$ is looked on as of normal distribution and its mean and standard deviation are respectively denoted as $\bar{p}_i$ and σ_{p_i}. To evaluate whether an arbitrary shape $\mathbf{s}$ is in conformity with the training dataset, immediate constraints are enforced on all its shape parameters $p_i^{\mathbf{s}}$:

$$\|p_i^{\mathbf{s}} - \bar{p}_i\| \leq 3\sigma_{p_i} \quad i = 1, 2 \cdots, N \tag{2.5}$$

2.2 2D Texture Model

From [18], the statistical *texture* model for a 2D object is defined as the pixel intensities across the object (here, the face) in question (if necessary after a suitable normalization). Then, the texture statistical model is a vector of intensities $\mathbf{A}$ defined over all the u pixels inside the base mesh $\mathbf{s}_0$, i.e., $\mathbf{A}$ is defined over all $\mathbf{x} \in \mathbf{s}_0$, where $\mathbf{x} = (x, y)^T$. Equation (2.6) shows a texture vector when RGB channels or a single channel is considered respectively:

$$\begin{aligned} \mathbf{A} &= \begin{pmatrix} b_1, b_2, \cdots, b_u \\ g_1, g_2, \cdots, g_u \\ r_1, r_2, \cdots, r_u \end{pmatrix} \\ \mathbf{A} &= (gray_1, gray_2, \cdots, gray_u) \end{aligned} \tag{2.6}$$

For the actual implementation, a three-channel texture model is always reshaped to

$$\mathbf{A} = (b_1, b_2, \cdots, b_u, g_1, g_2, \cdots, g_u, r_1, r_2, \cdots, r_u) \tag{2.7}$$

The texture statistical model allows linear texture variation. This means an arbitrary texture $\mathbf{A}$ is looked on as a base texture $\mathbf{A}_0$ plus a linear combination of M textures $\mathbf{A}_i$:

$$\mathbf{A} = \mathbf{A}_0 + \sum_{i=1}^{M} \lambda_i \mathbf{A}_i = \mathbf{A}_0 + \lambda \mathbf{P}_{\mathbf{t}} \tag{2.8}$$

where λ_i are the texture parameters and $\mathbf{A}_i$ are orthonormal eigenvectors, with $\mathbf{P_t} = \begin{pmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \cdots \\ \mathbf{A}_M \end{pmatrix}$ and $\lambda = (\lambda_1 \quad \lambda_2 \quad \cdots \quad \lambda_M)$.

For all training textures $\mathbf{A}^j, j = 1, 2, \cdots, F$, each texture $\mathbf{A}^j$ is corresponding to a vector of texture parameters λ^j, F is the number of training shapes. For simplicity, all ith elements of all texture parameters λ^j, namely, the set $\{\lambda_i^1, \lambda_i^2, \cdots, \lambda_i^F\}$ is looked on as of normal distribution and its mean and standard deviation are respectively denoted as $\bar{\lambda_i}$ and σ_{λ_i}. To evaluate whether an arbitrary texture $\mathbf{a}$ is in conformity with the training dataset, immediate constraints are enforced on all its texture parameters $\lambda_i^{\mathbf{a}}$:

$$\|\lambda_i^{\mathbf{a}} - \bar{\lambda_i}\| \leq 3\sigma_{\lambda_i} \quad i = 1, 2 \cdots, M \tag{2.9}$$

By the way, few of the former documentary publications emphasized the global texture constraints. Particularly, for ICIA AAM (refer to 3.2.4) and ICIA AAM considering global shape normalization (refer to 3.3.3), (3.13) and (3.23) respectively indicate that no matter how $\mathbf{p}$ or how $\mathbf{p}$ and $\mathbf{q}$ are selected, the second term in (3.12) or (3.22) could always be minimized to 0. However, this conclusion is made upon the assumption without constrains in (2.9). With this special observance, a substantial contribution of [9] is just to discover some favorable global texture constraints to achieve better fitting performance for dynamic image sequences.

2.3 2D Concatenated Appearance Model

[4] first introduced the concept of appearance model, which is supposed to be able to simultaneously represent both the shape and texture variability trained from the face database. Meanwhile, the shape and texture are often correlated. Therefore, we may simply assume that both the shape parameter vector $\mathbf{p}$ and the texture parameter λ have a linear relationship with the same parameter vector, say $\mathbf{c}$, i.e.:

$$\mathbf{p} = \mathbf{cC_s} \tag{2.10}$$

$$\lambda = \mathbf{cC_t} \tag{2.11}$$

Simply combining (2.4), (2.8), (2.10) and (2.11) gives:

$$\mathbf{s} = \mathbf{s}_0 + \mathbf{cC_sP_s} = \mathbf{s}_0 + \mathbf{cQ_s} \tag{2.12}$$

$$\mathbf{A} = \mathbf{A}_0 + \mathbf{cC_tP_t} = \mathbf{A}_0 + \mathbf{cQ_t} \tag{2.13}$$

Now, the concatenated appearance model is synthesized by only one vector of parameters, that is $\mathbf{c}$, instead of using both the shape model parameters $\mathbf{p}$ and the texture model parameters λ.

2.4 Building 2D Statistical Models - A Summary

The process of building statistical shape model, texture model, and concatenated appearance model could be simply summarized in figure 2.1.

The goal of ASMs or AAMs is just to find the best match for an arbitrary face with reference to the trained template face. More specifically, the key points' locations should be given out after fitting the model, no matter the fitting process is based on a shape model only, or based on a shape-texture joint model.

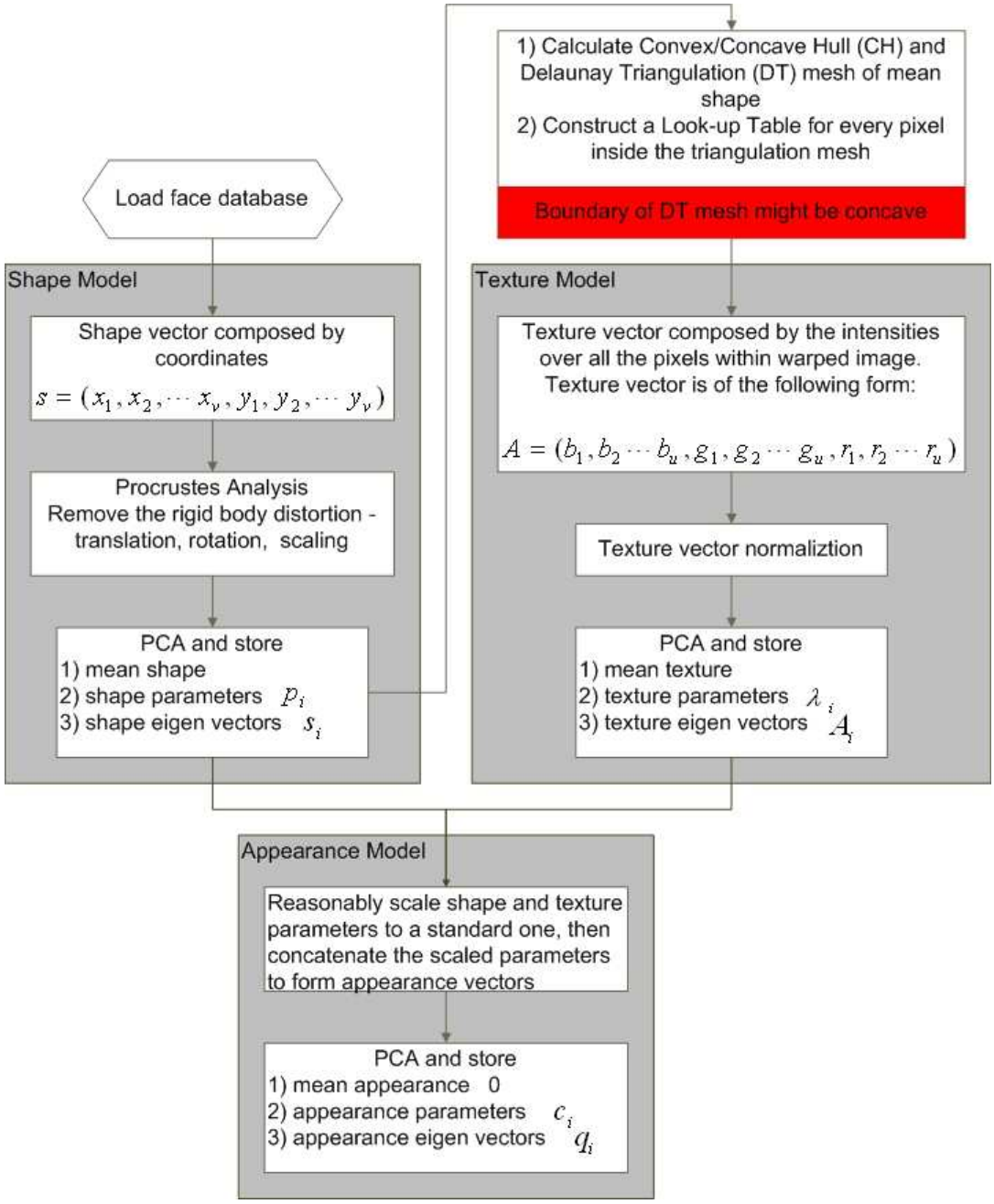


Figure 2.1: Statistical Model Building Framework

Chapter 3

2D Statistical Model Fitting

3.1 Active Shape Model

ASMs try to locally optimize the position of every model point in conformity with the global shape constraint, based on the trained point distribution models (PDM) [23].

All ASMs are of multiple objectives, namely, for every model point, find the best match within a small image patch around the location of the corresponding trained point, subject to multiple shape parameter constraints (2.5). If every objective is looked on as independent of others, then multiple objectives of ASMs can be easily integrated into a single objective in (3.1).

$$C_{ASM}(\mathbf{s}) = \frac{1}{v}\sum_{i=1}^{v} C(\mathbf{x_i}) \quad \text{subject to} \quad \|p_j^{\mathbf{s}} - \bar{p_j}\| \leq 3\sigma_{p_j} \;\; \forall j \in \{1, 2, \cdots, N\} \tag{3.1}$$

, where $C(\mathbf{x_i})$ is the cost at the model point $\mathbf{x_i} = (x_i, y_i)^T$, and $C_{ASM}(\mathbf{s})$ is the total average cost of shape $\mathbf{s}$. The difference among ASM variants is just how to define the cost at respective model point $\mathbf{x_i}$, including:

- how to represent the local textures around $\mathbf{x_i}$ – a gradient vector along one direction or two gradient vectors along two orthogonal directions, direct image intensities over a small patch around $\mathbf{x_i}$, various transforms of the small image patch or various features extracted from the small image patch, etc.

- which cost function is to be used to evaluate the cost at $\mathbf{x_i}$ – Mahalanobis distance, fast normalized cross correlation (FNCC), etc.

Popular ASM variants are summarized in table 3.1.

Table 3.1: ASM categories

ASM Variants	Local Texture Representation	Cost Function
1D Profile ASM	a vector of gradients	Mahalanobis distance
2D Profile ASM	two vectors of gradients	
Milborrow's 2D Profile ASM	image intensities	
direct LTC-ASM	image intensities	FNCC
Gabor LTC-ASM	Gabor features	Mahalanobis distance or FNCC
Haar LTC-ASM	Haar features	
LBP LTC-ASM	LBP features	

3.1.1 1D Profile ASM

1D profile ASM [5] is based on profile normal to the boundary of every model point.

For an arbitrary model point, the boundary could be defined by two line segments, the segment connecting this point and the point that this point connected from, and the segment connecting this point and the point that this point connected to. The unit normal vector to the boundary could be defined just by these three points' coordinates.

Along the normal direction, at each side of the model point $\mathbf{x_i}, 1 \leq i \leq v$, k pixels' derivatives (rather than the original gray-level values) are sampled and sequentially listed in a vector $\mathbf{g}_i = (g_{i,-k}, g_{i,-(k-1)}, \cdots, g_{i,k})^T$. $\mathbf{g}_i$ is then normalized to form the sequential feature set (essentially, a vector of scaled gradients) for this specific key point. Formularily,

$$\mathbf{g}_i \to \frac{1}{\sum_{j=-k}^{k} |g_{ij}|} \mathbf{g}_i \tag{3.2}$$

For every given model point $\mathbf{x_i}$ of each training image, a normalized vector of derivatives could be extracted. If we look on all normalized vectors from all training images at this specific point as a multivariate Gaussian, the quality of fitting a new object to the model at this specific point could be evaluated by Mahalanobis distance:

$$f(\mathbf{g}_i^*) = \sqrt{(\mathbf{g}_i^* - \bar{\mathbf{g}}_i)^T \mathbf{S}_i^{-1} (\mathbf{g}_i^* - \bar{\mathbf{g}}_i)} \tag{3.3}$$

where $\bar{\mathbf{g}}_i$ and $\mathbf{S}_i^{-1}$ are respectively mean vector and inverse of covariance matrix in terms of the training vectors at model point $\mathbf{x}_i$, $\mathbf{g}_i^*$ is the newly extracted sequential feature set of the object at the same point under concern.

The Mahalanobis distance $f(\mathbf{g}_i^*)$ is just the cost function at point $\mathbf{x}_i$ adopted by 1D profile ASM, namely:

$$C(\mathbf{x_i}) = f(\mathbf{g}_i^*) \tag{3.4}$$

3.1.2 2D Profile ASM

1D profile ASM has a quite obvious shortcoming: every model point can only search its optimal location along its normal to the boundary. In order to extend its flexibility, instead of sampling a single vector of gradients just along the profile normal, an additional vector of gradients along the profile tangential is further used. By sampling two vectors of gradients along these two directions, two sequential feature sets $\mathbf{g}_i^1$ and $\mathbf{g}_i^2$ can be obtained. The optimization process of every single model point can be realized by searching for the optimum along two directions in turn. Namely, we search for the first optimal location using cost function $f(\mathbf{g}_i^{1^*})$ in the normal direction. After the location has been estimated along the normal direction, the tangent across this optimal location can be easily determined. We then carry out the second search for the optimal location in this tangential direction, using cost function $f(\mathbf{g}_i^{2^*})$.

Note: Milborrow's 2D profile ASM [16] is actually a type of direct LTC-ASMs. Please refer to 3.1.3 for details.

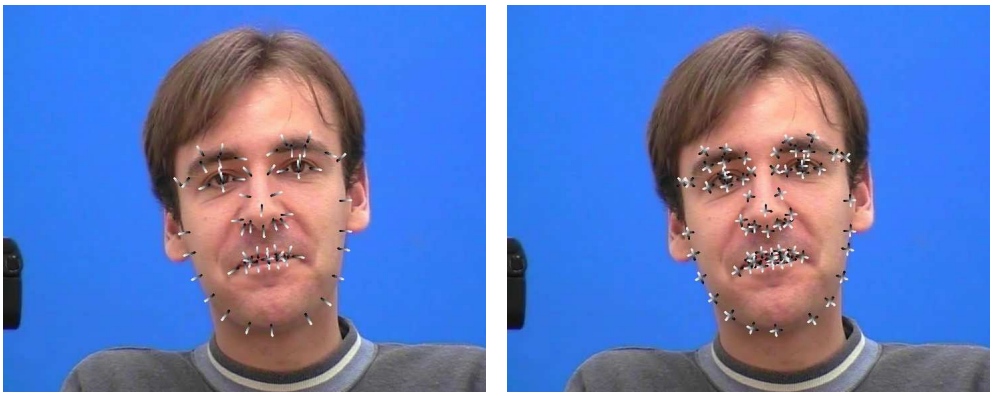
Figure 3.1 shows 1D and 2D profile normals of the manually labeled points on an arbitrarily selected face sample from FRANCK database. Every line segment in 3.1(a) represents a profile normal in the normal direction. The line segments in 3.1(b) represent all profile normals both in the normal direction and in the tangential direction. For all line segments, white end and black end of one line segment indicates its direction.

3.1.3 Local Texture Constrained ASM

Direct LTC-ASM

Direct LTC-ASM is directly based on the small image patch around each model point.

For every given model point $\mathbf{x_i}$ of each training image, a small image patch around this model point could be extracted. If we look on the mean image patch of all training images at $\mathbf{x_i}$ as the standard template image T_i, and a small area bigger than the size of this image patch from a new testing image around $\mathbf{x_i}$ as the search image S_i, the quality of fitting a new object to the model at $\mathbf{x_i}$ could be evaluated by image template matching using fast normalized cross-correlation (FNCC).



(a) 1D Profile (b) 2D Profile

Figure 3.1: 1D and 2D profiles

The NCC is just the cost function at point $\mathbf{x}_i$ adopted by direct LTC-ASM, namely:

$$C(\mathbf{x_i}) = \mathbf{NCC}_{S_i}^{T_i}(\mathbf{x_i}) \tag{3.5}$$

Note: Here, $\mathbf{x_i}$ is not the patch's top left corner but its center in the search image.

Milborrow's 2D profile ASM

Milborrow's 2D profile ASM [16] is actually a type of direct LTC-ASMs. It samples the direct image intensities within "a square or circular region around the landmark" (cite from [16]). By sequentially connecting the samples of each row within the region, a new sequential feature set $\mathbf{g}_i^*$ is now being generated. The cost function of Milborrow's method is exactly the same as (3.4).

Transform Based LTC-ASM

Instead of directly using intensities over the image patch around model point $\mathbf{x_i}$, some representative texture features can be extracted after some transforms and compose an attribute vector. Then, measurement like Mahalanobis distance or FNCC can be used to evaluate the similarity between the trained attribute vector and the testing attribute vector.

Typical and representative transforms include Gabor transform, Haar wavelet transform, LBP transform, other wavelet transforms, etc. Considering the speed issue, Haar-wavelet and LBP features have been adopted in

Haar-wavelet LTC-ASM [27, 28] and LBP LTC-ASM [13, 14]. It has been proved in [13, 14] that LBP LTC-ASM have achieved superior fitting results.

Practical Difficulty for Transform Based LTC-ASM

Normally, transform based LTC-ASMs extract a great many representative features over small image patches in multiple scales. These oceanic data are too many to be handled in practice. The total numbers of direct image intensity features, multi-scale LBP and Haar features extracted over a squared image patch in LTC-ASM are enumerated in table 3.2.

Table 3.2: Number of multi-scale features for one local patch

Patch Size	Pixels	LBP	Haar Basic	Haar Core	Haar All
$8 * 8$	64	81	2,056	2,569	3,001
$16 * 16$	256	1,600	32,384	41,600	50,878
$32 * 32$	1,024	27,225	510,112	664,057	832,665

Supposing an image patch of size $16 * 16$ is adopted for each key point, and if all extracted LBP features are to be considered, and if Mahalanobis distance is adopted as the cost function, the covariance matrix for every key point will be of size $1,600 * 1,600$. Therefore, for IMM face database with 58 labeled key points, it requires at least $1,600*1,600*58*8/1,024^3 \approx 1.1$ giga memory to just load the covariance matrices, which is not operable at all for low cost IW, or even ordinary laptops. The oceanic data is also the reason why to train an ensemble cascade for face detection is quite slow.

Therefore, a possible way to make transform based LTC-ASM applicable is to carry out feature selection. Problems such as how to select the most representative features, how to define the representability of respective features or feature combinations can be retrieved in [8]. This difficulty is left as an open issue and unresolved at the current stage.

3.1.4 Generalized Multi-scale ASM Fitting Algorithm

To speed up the search process and also improve the robustness of ASM fitting performance, multi-scale ASMs [2] are adopted in our experiments. This involves searching for the face in a coarse image first, and then refining the face key points' locations in a series of finer resolution images. Algorithm 1 shows the entire process of generalized multi-scale ASM fitting algorithms.

Algorithm 1 Generalized multi-scale ASM fitting

1: Pre-computation:

- Sequential feature set for each model point j at level l $\bar{\mathbf{g}}_{lj}$

2: Initialization:

- $\mathbf{p} = \mathbf{0}, \mathbf{s}_{model} = \mathbf{s_0}$
- $LEVEL = 4, l = LEVEL - 1, pCLOSE = 0.9, EPOCH = 4, TOL = 2$
- $PTS = number\ of\ model\ points$

3: Either estimate the current face shape $\mathbf{s}_{real}$ as the trained statistical mean shape based on Adaboost face detection result, or by the tracking result of the last frame. Denote every shape point as $u_j, 0 \leq j < PTS$.
4: **while** $(l >= 0)$ **do**
5: **for** $i = 0$ to $EPOCH$ **do**
6: $n_{good} = 0$;
7: **for all** j such that $0 \leq j < PTS$ **do**
8: Find the best match between the testing image and trained template image within a small image patch around j. Record this best match point as v_j.
9: **end for**
10: Update all best match points $v_j, 0 \leq j < PTS$ to $v'_j, 0 \leq j < PTS$ in conformity with multiple trained shape parameter constraints, say, every shape parameter must be within 3σ around its mean value.
11: **for all** j such that $0 \leq j < PTS$ **do**
12: $s_j = u_j - v'_j$
13: **if** $s_j < TOL$ **then**
14: $n_{good}++$;
15: **end if**
16: $u_j = v'_j$
17: **end for**
18: **if** $n_{good} > PTS * pCLOSE$ **then**
19: break;
20: **end if**
21: **end for**
22: $l--$;
23: **end while**

3.2 Active Appearance Model

Unlike ASM that seeks to match the position of every model point directly, AAM tries to match both the shape and the holistic texture representation of the object simultaneously. In this section, two popular AAM variants, basic AAM and ICIA AAM and their siblings such as CLM, FAIA AAM and IAIA AAM will be discussed.

3.2.1 Basic AAM Fitting

The objective of basic AAM is to minimize the mean square error (MSE) of the difference between the real image $\mathbf{A}_{real}$ and the image built from the model parameters $\mathbf{A}_{model}$:

$$\begin{aligned} \mathbf{r}(\mathbf{c}) &= \mathbf{A}_{real} - \mathbf{A}_{model} && (3.6) \\ E(\mathbf{c}) &= \mathbf{r}(\mathbf{c})^T \mathbf{r}(\mathbf{c}) && (3.7) \end{aligned}$$

Looking on the above problem as a classical optimization problem, by using numeric differentiation method, we may simply cope with it in an iterative way. That is to say, for each iteration, our aim is to find the most suitable $\delta\mathbf{c}$, so that (3.8) goes to minimum.

$$E(\mathbf{c} + \delta\mathbf{c}) = \mathbf{r}(\mathbf{c} + \delta\mathbf{c})^T \mathbf{r}(\mathbf{c} + \delta\mathbf{c}) \quad (3.8)$$

After each iteration, $\mathbf{c}$ is updated to $\mathbf{c} + \delta\mathbf{c}$. The iterations will proceed continuously until $E(\mathbf{c}+\delta\mathbf{c})$ and $E(\mathbf{c})$ are of negligible difference. Thus, the problem is converted to how to calculate the updating value $\delta\mathbf{c}$ in terms of the current $\mathbf{c}$ in each iteration. In order to compute $\delta\mathbf{c}$, we first carry out 1-order Taylor expansion, and then carry out the partial derivative on (3.8).

$$\begin{aligned} \frac{\partial E(\mathbf{c}+\delta\mathbf{c})}{\partial \delta\mathbf{c}} &= \frac{\partial \mathbf{r}(\mathbf{c}+\delta\mathbf{c})^T \mathbf{r}(\mathbf{c}+\delta\mathbf{c})}{\partial \delta\mathbf{c}} \\ &= \frac{\partial (\mathbf{r}(\mathbf{c}) + \frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}}\delta\mathbf{c})^T (\mathbf{r}(\mathbf{c}) + \frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}}\delta\mathbf{c})}{\partial \delta\mathbf{c}} \\ &= \frac{\partial (\mathbf{r}(\mathbf{c})^{\mathbf{T}}\mathbf{r}(\mathbf{c}) + \mathbf{2}\mathbf{r}(\mathbf{c})^{\mathbf{T}}\frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}}\delta\mathbf{c} + \delta\mathbf{c}^{\mathbf{T}}(\frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}})^{\mathbf{T}}\frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}}\delta\mathbf{c})}{\partial \delta\mathbf{c}} \\ &= 0 + 2\mathbf{r}(\mathbf{c})^{\mathbf{T}}\frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}} + 2\delta\mathbf{c}^{\mathbf{T}}(\frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}})^{\mathbf{T}}\frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}} \end{aligned}$$

Letting the above formula equal to zero gives:

$$\delta\mathbf{c} = -\mathbf{H}\mathbf{r}(\mathbf{c}) \quad (3.9)$$

$$\mathbf{H} = ((\frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}})^{\mathbf{T}} \frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}})^{-\mathbf{1}} (\frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}})^{\mathbf{T}} \tag{3.10}$$

Thus, the basic AAM fitting could be summarized in algorithm 2.

3.2.2 Constrained Local Model

Cristinacce's constrained local model (CLM) [6] is quite similar to basic AAM. The only difference between CLM and basic AAM is CLM concatenates texture patches around every model point to form the texture vector, instead of considering all textures over the whole face. The fitting process of CLM is exactly the same as basic AAM as introduced in 3.2.1. Consequently, we may regard CLM as an improved version of basic AAM.

3.2.3 Lucas-Kanade Image Alignment

Inevitably, basic AAM and all ASM variants rely on the background. Thus, professor Cootes suggests to "train on edge strength images, rather than raw intensity images, which could mitigate this background disturbances a little bit". In 2004, CMU summarized two subtle algorithms IAIA and ICIA for AAM fitting [15], which successfully avoided the disturbance from the background.

"The goal of image alignment is to find the location of a constant template image in an input image." (cited from [15]). Lucas-Kanade method is applied to calculate the motion between two image frames taken at different time for every pixel position [11] [12]. A revised version of Lucas-Kanade algorithm for AAM could be found in [15], which is to find the best alignment by minimizing the following $L2$ norm:

$$E(\mathbf{x}) = \|\mathbf{A}_0(\mathbf{x}) + \sum_{i=1}^{M} \lambda_i \mathbf{A}_i(\mathbf{x}) - \mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}))\|^2 \tag{3.11}$$

where $\mathbf{A}_0(\mathbf{x}) + \sum_{i=1}^{M} \lambda_i \mathbf{A}_i(\mathbf{x})$ comes from (2.8); $\mathbf{W}(\mathbf{x};\mathbf{p})$ is the destination pixel coordinates warped from the source pixel $\mathbf{x}$ using the shape parameters $\mathbf{p}$ on an arbitrary input image; and $\mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}))$ is just the intensity (intensities) vector of all warped pixels $\mathbf{W}(\mathbf{x};\mathbf{p})$ in this input image. AAM fitting is just to minimize the above expression with respect to both shape parameters $\mathbf{p}$ and texture parameters λ.

Algorithm 2 Basic AAM Fitting

1: Initialization:

- Precomputed $\mathbf{H}$
- $\mathbf{c} = \mathbf{0}, \mathbf{p} = \mathbf{0}, \lambda = \mathbf{0}, \mathbf{c}_{temp} = \mathbf{0}$
- $\mathbf{s}_{model} = \mathbf{s_0}, \mathbf{A}_{model} = \mathbf{A}_0, \mathbf{A}_{error} = \mathbf{0}$
- $i = 0, EPOCH = 30, d = MAX$

2: Extract and normalize the current image texture $\mathbf{A}_{real}$ in terms of the modeled shape $\mathbf{s}_{model}$
3: $\mathbf{A}_{error} = \mathbf{A}_{real} - \mathbf{A}_{model}$ refer to (3.6)
4: $d = \|\mathbf{A}_{error}\|$, $d_{temp} = d$
5: **while** $(i < EPOCH)$ && $(d > FLT\_EPSILON)$ **do**
6: $\delta\mathbf{c} = -\mathbf{H}\mathbf{A}_{error}$ refer to (3.9)
7: **for** $k = 0$ to $k = 5$ **do**
8: $\kappa = 0.5^k$
9: $\mathbf{c}_{temp} = \mathbf{c} + \kappa\delta\mathbf{c}$
10: Rebuild the modeled shape $\mathbf{s}_{model}$ and texture $\mathbf{A}_{model}$ according to (2.12) and (2.13), in terms of $\mathbf{c}_{temp}$
11: Extract and normalize the current image texture $\mathbf{A}_{real}$ in terms of the modeled shape $\mathbf{s}_{model}$
12: $\mathbf{A}_{temp} = \mathbf{A}_{real} - \mathbf{A}_{model}$
13: $d_{temp} = \|\mathbf{A}_{temp}\|$
14: **if** $d_{temp} < d$ **then**
15: break; (break **for**)
16: **end if**
17: **end for**
18: **if** $d_{temp} < d$ **then**
19: $\mathbf{c} = \mathbf{c}_{temp}$;
20: $d = d_{temp}$;
21: $\mathbf{A}_{error} = \mathbf{A}_{temp}$;
22: **else**
23: break; (break **while**)
24: **end if**
25: $++i$;
26: **end while**

Apparently, (3.11) can be spanned as:

$$\begin{aligned} & \|\mathbf{A}_0(\mathbf{x}) + \sum_{i=1}^{M} \lambda_i \mathbf{A}_i(\mathbf{x}) - \mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}))\|^2_{span(\mathbf{A}_i)^\perp} \\ & + \|\mathbf{A}_0(\mathbf{x}) + \sum_{i=1}^{M} \lambda_i \mathbf{A}_i(\mathbf{x}) - \mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}))\|^2_{span(\mathbf{A}_i)} \\ = \; & \|\mathbf{A}_0(\mathbf{x}) - \mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}))\|^2_{span(\mathbf{A}_i)^\perp} \\ & + \|\mathbf{A}_0(\mathbf{x}) + \sum_{i=1}^{M} \lambda_i \mathbf{A}_i(\mathbf{x}) - \mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}))\|^2_{span(\mathbf{A}_i)} \end{aligned} \tag{3.12}$$

Clearly, any vector projected (spanned) in the eigenvector $span(\mathbf{A}_i)$ space can be seamlessly expressed by the linear combination of these eigenvectors $\mathbf{A}_i$. So, no matter how $\mathbf{p}$ is chosen, the second term in (3.12) could always be minimized to 0. Therefore,

$$\lambda_i = \mathbf{A}_i^T(\mathbf{x}) \cdot [\mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p})) - \mathbf{A}_0(\mathbf{x})] \tag{3.13}$$

For the first term in (3.12), we will resort to IAIA or ICIA in the next subsection.

3.2.4 IAIA and ICIA AAM Fitting

The ordinary idea to minimize $\|\mathbf{A}_0(\mathbf{x}) - \mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}))\|^2_{span(\mathbf{A}_i)^\perp}$ is to adopt traditional linear optimization method by linearly adjusting $\mathbf{p}$ in each iteration: $\mathbf{p} \rightarrow \mathbf{p} + \delta\mathbf{p}$. Forward compositional method brings a new thought by adjusting $\mathbf{p}$ in a nesting way, so the optimization problem could be addressed as: $\|\mathbf{A}_0(\mathbf{x}) - \mathbf{I}(\mathbf{W}(\mathbf{W}(\mathbf{x};\delta\mathbf{p});\mathbf{p}))\|^2_{span(\mathbf{A}_i)^\perp}$. Based on forward additive image alignment method, inverse compositional image alignment method is just reversing the roles of the trained model template image $\mathbf{A}_0(\mathbf{x})$ and the input image $\mathbf{I}(\mathbf{W}(\mathbf{W}(\mathbf{x};\delta\mathbf{p});\mathbf{p}))$, which results in

$$\|\mathbf{A}_0(\mathbf{W}(\mathbf{x};\delta\mathbf{p})) - \mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}))\|^2_{span(\mathbf{A}_i)^\perp} \tag{3.14}$$

After taking the 1-order Taylor series expansion in terms of $\delta\mathbf{p}$ at $\delta\mathbf{p} = 0$,

we have:

$$\begin{aligned}
&\|\mathbf{A}_0(\mathbf{W}(\mathbf{x};\mathbf{0})) + \frac{\partial \mathbf{A}_0(\mathbf{W}(\mathbf{x};\delta\mathbf{p}))}{\partial \mathbf{W}(\mathbf{x};\delta\mathbf{p})}\frac{\partial \mathbf{W}(\mathbf{x};\delta\mathbf{p})}{\partial \delta\mathbf{p}} \mid_{\delta\mathbf{p}\to\mathbf{0}} (\delta\mathbf{p}-\mathbf{0}) - \mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}))\|^2_{span(\mathbf{A}_i)^\perp} \\
&= \|\mathbf{A}_0(\mathbf{x}) + \frac{\partial \mathbf{A}_0(\mathbf{W}(\mathbf{x};\mathbf{0}))}{\partial \mathbf{W}(\mathbf{x};\mathbf{0})}\frac{\partial \mathbf{W}(\mathbf{x};\delta\mathbf{p})}{\partial \delta\mathbf{p}} \mid_{\delta\mathbf{p}\to\mathbf{0}} \delta\mathbf{p} - \mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}))\|^2_{span(\mathbf{A}_i)^\perp} \\
&= \|\mathbf{A}_0(\mathbf{x}) + \frac{\partial \mathbf{A}_0(\mathbf{x})}{\partial \mathbf{x}}\frac{\partial \mathbf{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}} \mid_{\mathbf{p}\to\mathbf{0}} \delta\mathbf{p} - \mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}))\|^2_{span(\mathbf{A}_i)^\perp} \\
&= \|\mathbf{A}_0(\mathbf{x}) + \nabla\mathbf{A}_0\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\delta\mathbf{p} - \mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}))\|^2
\end{aligned} \tag{3.15}$$

Here, for simplicity, we

- denote $\nabla\mathbf{A}_0$ as the gradient image for the template image, i.e., $\frac{\partial \mathbf{A}_0(\mathbf{x})}{\partial \mathbf{x}}$
- denote $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ as the Jacobian $\frac{\partial \mathbf{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}}$ evaluated at $(\mathbf{x};0)$
- omit $_{span(\mathbf{A}_i)^\perp}$ hereafter

In order to minimize (3.15), we compute its partial derivative in terms of $\delta\mathbf{p}$ with matrix calculus [22]:

$$2[\mathbf{A}_0^T(\mathbf{x}) - \mathbf{I}^T(\mathbf{W}(\mathbf{x};\mathbf{p})) + \delta\mathbf{p}^T(\nabla\mathbf{A}_0\frac{\partial \mathbf{W}}{\partial \mathbf{p}})^T](\nabla\mathbf{A}_0\frac{\partial \mathbf{W}}{\partial \mathbf{p}}) \tag{3.16}$$

Apparently, the solution to the above equation is:

$$\delta\mathbf{p} = \mathbf{H}[\mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p})) - \mathbf{A}_0(\mathbf{x})] \tag{3.17}$$

$$\mathbf{H} = ((\nabla\mathbf{A}_0\frac{\partial \mathbf{W}}{\partial \mathbf{p}})^T(\nabla\mathbf{A}_0\frac{\partial \mathbf{W}}{\partial \mathbf{p}}))^{-1}(\nabla\mathbf{A}_0\frac{\partial \mathbf{W}}{\partial \mathbf{p}})^T \tag{3.18}$$

Since there is nothing depending on $\mathbf{p}$, $\mathbf{H}$ is constant across iterations and can be pre-computed.

If we denote $I(\mathbf{x})$ is the intensity of a single pixel $\mathbf{x} = (x, y)^T$ inside $\mathbf{A}_0$, (3.17) could be explained in a much clearer way. Apparently, $\nabla\mathbf{A}_0$ is the image gradients over $\mathbf{A}_0$. For every single pixel, the gradient is $\nabla I(\mathbf{x}) = (\frac{\partial I(\mathbf{x})}{\partial x}, \frac{\partial I(\mathbf{x})}{\partial y})$, which is just one element of $\nabla\mathbf{A}_0$. This $\nabla I(\mathbf{x})$ should be a row vector of size $1 * 2$ according to "Matrix Calculus [22]". For the same pixel, $\mathbf{w}(\mathbf{x};\mathbf{p})$ denotes the warped pixel, which should be a column vector of size $2 * 1$. Meanwhile, $\mathbf{p}$ is a column vector of size $N * 1$ (N refer to

(2.1)). Thus, $\frac{\partial \mathbf{w}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}}$ is of size $2 * N$. Therefore, for every pixel inside $\mathbf{A}_0$, $\nabla I(\mathbf{x})\frac{\partial \mathbf{w}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}}$ is a row vector of size $1*N$. Furthermore, for all pixels within $\mathbf{A}_0$, modified steepest descent image $\mathbf{MSDI} = (\nabla \mathbf{A}_0 \frac{\partial \mathbf{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}})_{span(\mathbf{A}_i)^\perp}$ is just a matrix of size $(3u) * N$ or $u * N$ depending on whether it is a color image or a gray-level image. Hence, the Hessian matrix is of size $N * N$.

What's more, there is one thing to be emphasized: the template image always keeps constant during fitting iterations. Actually, we never carry out a warp on the template image as $\mathbf{W}(\mathbf{x};\delta\mathbf{p})$. What we are going to update is the parameter $\mathbf{p}$ which actually triggers the warping for the real-time input image, instead of the template image. Unlike forward additive image alignment (FAIA) algorithm, for inverse additive image alignment (IAIA) algorithm, during fitting iterations, $\mathbf{p}$ is going to be updated in the way of $\mathbf{p} \rightarrow \mathbf{p} - \delta\mathbf{p}$. In addition, for inverse compositional image alignment (ICIA) algorithm, $-\delta\mathbf{p}$ is directly used to build up a mesh. The relationship between the current object shape and the shape to be updated to is just corresponding to the relationship between the trained mean aligned shape and this mesh constructed by $-\delta\mathbf{p}$. Therefore, without updating shape parameter $\mathbf{p}$, ICIA AAM estimates the new shape in a composition way. Please refer to [15] for details of warp inversion methods for IAIA and ICIA AAMs.

With the above deduction, IAIA and ICIA algorithms are summarized in algorithm 3, also shown in figure 3.2.

3.3 Global Shape Normalization for Statistical Models

3.3.1 An Open Issue

In the above section, AAM variants are based on the assumption that the face shape has already been normalized according to some uniform criterion. But, it's emphasized in [15] that "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."

During the training process of statistical models, iterative Procrustes analysis [1, 3, 7, 24] is applied to align a set of shapes iteratively, which ensures the residual variation in the training set is only due to shape non-rigid variations.

In summary, the shape model could be built in two standard steps as:

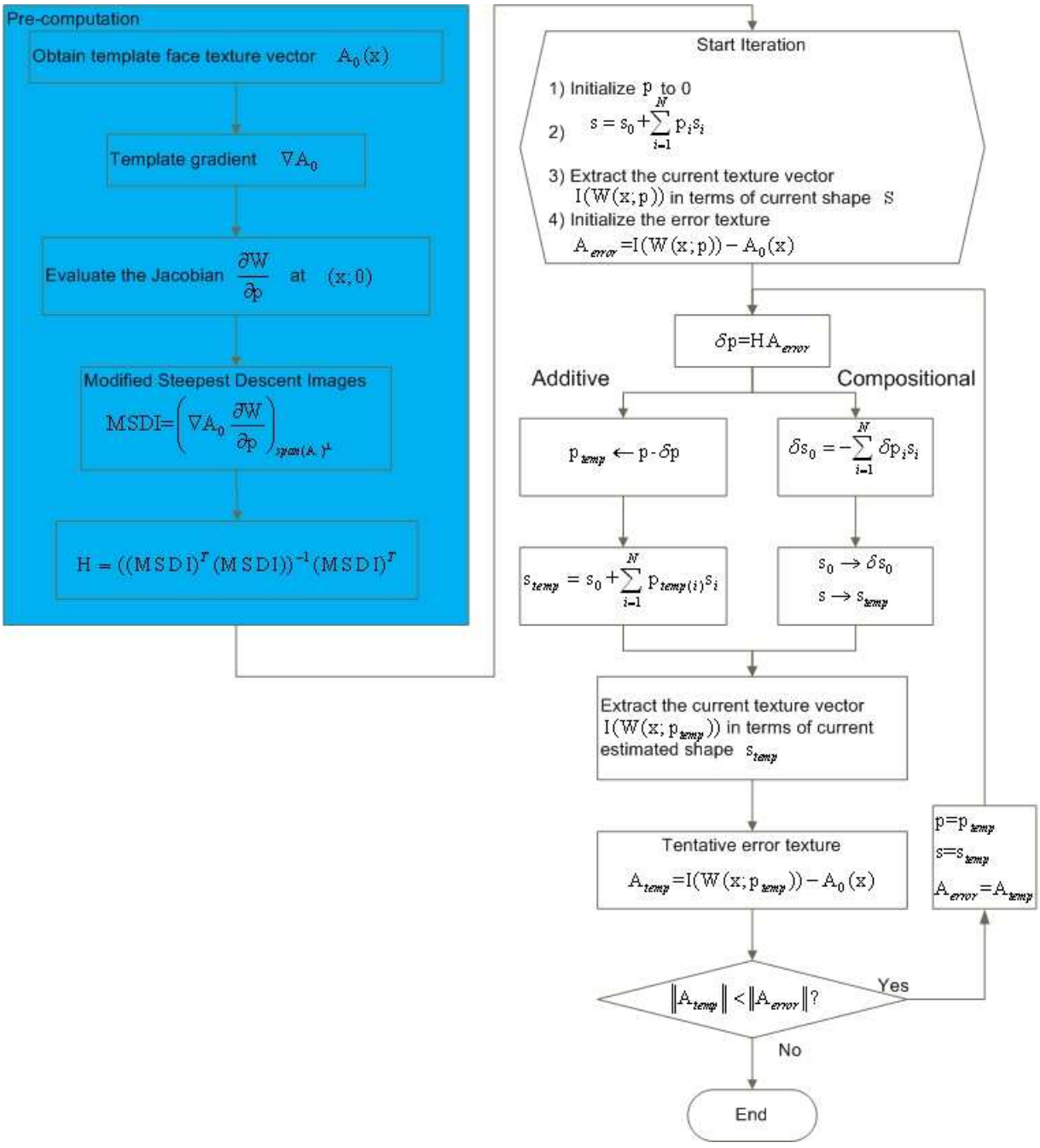


Figure 3.2: Inverse additive image alignment algorithm

Algorithm 3 IAIA and ICIA Fitting

1: Pre-computation:

- Template image gradient $\nabla \mathbf{A}_0$
- Modified steepest descent image $\mathbf{MSDI} = (\nabla \mathbf{A}_0 \dfrac{\partial \mathbf{W}}{\partial \mathbf{p}})_{span(\mathbf{A}_i)^\perp}$,
- $\mathbf{H} = (\mathbf{MSDI}^T \mathbf{MSDI})^{-1} \mathbf{MSDI}^T$

2: Initialization:

- $\mathbf{p} = \mathbf{0}$
- $\mathbf{s}_{model} = \mathbf{s}_{temp} = \mathbf{s_0}$, $\mathbf{A}_{model} = \mathbf{A}_0(\mathbf{x})$, $\mathbf{A}_{error} = \mathbf{0}$
- $i = 0, EPOCH = 30, d = MAX$

3: Extract the current image texture $\mathbf{A}_{real}$ in terms of the modeled shape $\mathbf{s}_{model}$, and normalize it
4: $\mathbf{A}_{error} = \mathbf{A}_{real} - \mathbf{A}_{model}$
5: $d = \|\mathbf{A}_{error}\|$, $d_{temp} = d$
6: **while** $(i < EPOCH)$ && $(d > FLT\_EPSILON)$ **do**
7: $\delta \mathbf{p} = \mathbf{H}\mathbf{A}_{error}$ refer to (3.17)
8: For IAIA: $\mathbf{p}_{temp} = \mathbf{p} - \delta \mathbf{p}$; and rebuild the modeled shape $\mathbf{s}_{temp}$ in terms of $\mathbf{p}_{temp}$, according to (2.4);
For ICIA: $\delta \mathbf{s_0} = -\sum_1^N \delta p_i \mathbf{s}_i$; and the relationship between $\mathbf{s}_0$ and $\delta \mathbf{s_0}$ is just corresponding to the relationship between current shape $\mathbf{s}_{model}$ and its update $\mathbf{s}_{temp}$. Compose $\mathbf{s}_{temp}$ under the shape parameter constraints.
9: Extract and normalize the current image texture $\mathbf{A}_{temp}$ in terms of the modeled shape $\mathbf{s}_{temp}$
10: $\mathbf{A}_{error} = \mathbf{A}_{temp} - \mathbf{A}_0(\mathbf{x})$
11: $d_{error} = \|\mathbf{A}_{error}\|$
12: **if** $d_{error} < d$ **then**
13: $\mathbf{p} = \mathbf{p}_{temp}$;
14: $d = d_{temp}$;
15: $\mathbf{s}_{model} = \mathbf{s}_{temp}$;
16: $\mathbf{A}_{error} = \mathbf{A}_{temp}$;
17: **else**
18: break;
19: **end if**
20: $++i$;
21: **end while**

1. Procrustes analysis, which is used as a criterion to remove the rigid body distortion, including translation, rotation, and scaling.

2. PCA is then carried out on the "normalized" shapes to analyze the non-rigid body variations.

However, during AAM fitting process, the real face is unknown and to be determined. That is to say, we've got no idea how to normalize the estimated face because unlike the faces in the training dataset which have been manually annotated accurately, the estimated face is not the ground truth face at all. If Procrustes analysis is carried out on this estimated face and produce an estimated normalized face, whether this estimated normalized face is able to be represented by the linear composition of the trained face shape eigenvectors is unclear, due to the reason that for the real image, the rigid distortion has been accurately removed by Procrustes analysis or not is unclear at all. As a conclusion, global shape normalization is a must to be taken into consideration for AAMs, which should be able to estimate the face translation, rotation and scaling.

3.3.2 Global Shape Normalization for Basic AAM

If basic AAM is adopted, with a view to global shape normalization, fitting the trained AAM to an **arbitrary** image $\mathbf{A}(\mathbf{x})$ is equal to minimizing:

$$\mathbf{r}(\mathbf{c}|\mathbf{t}) = \mathbf{A}_{real}(\mathbf{x}) - \mathbf{A}_{model}(\mathbf{x}) \tag{3.19}$$

$$E(\mathbf{c}|\mathbf{t}) = \mathbf{r}(\mathbf{c}|\mathbf{t})^T \mathbf{r}(\mathbf{c}|\mathbf{t}) \tag{3.20}$$

, where $\mathbf{c}$ refers to the appearance model parameters in (2.12) and (2.13), and $\mathbf{t}$ refers to the pose parameters. That is to say, different from (3.6), basic AAM with global shape normalization looks on the difference between the real image and the model image as a function of both $\mathbf{c}$ and $\mathbf{t}$, rather than just $\mathbf{c}$. Thus, during the training process, we need to displace each parameter of both $\mathbf{c}$ and $\mathbf{t}$ from a known value on the training images, so that $\frac{\partial \mathbf{r}(\mathbf{c}|\mathbf{t})}{\partial(\mathbf{c}|\mathbf{t})}$ could be estimated by numeric differentiation. The only thing that needs to bear in mind is that the pose parameter vector $\mathbf{t}$ needs to transform a bit so that $\mathbf{t}$ brings linear variation to the model shape when $\mathbf{t}$ shifts a little bit to $\mathbf{t} + \delta\mathbf{t}$. For details, please refer to [5]. By the way, [15] gives a more elegant deduction on the same parameterizing solution to global shape normalization.

3.3.3 Global Shape Normalization for IAIA and ICIA AAMs

If IAIA or ICIA AAM is adopted, with a view to global shape normalizing, fitting the trained AAM to an **arbitrary** image $\mathbf{I}(\mathbf{x})$ is equal to minimizing:

$$E(\mathbf{x}) = \|\mathbf{A}_0(\mathbf{x}) + \sum_{i=1}^{M} \lambda_i \mathbf{A}_i(\mathbf{x}) - \mathbf{I}(\mathbf{N}(\mathbf{W}(\mathbf{x};\mathbf{p});\mathbf{q}))\|^2 \tag{3.21}$$

with respect to the texture parameters λ, the shape parameters $\mathbf{p}$ for AAM and the global shape normalization parameters $\mathbf{q}$, where $\mathbf{W}$ describes the piecewise affine warp, $\mathbf{N}$ refers to the global shape transform for normalization, $\mathbf{I}$ extracts the intensities over the pixels within the estimated object shape, and $\mathbf{A}_0(\mathbf{x}) + \sum_{i=1}^{M} \lambda_i \mathbf{A}_i(\mathbf{x})$ comes from the AAM texture model (2.8).

Apparently, (3.21) can be spanned as:

$$\begin{aligned} & \|\mathbf{A}_0(\mathbf{x}) + \sum_{i=1}^{M} \lambda_i \mathbf{A}_i(\mathbf{x}) - \mathbf{I}(\mathbf{N}(\mathbf{W}(\mathbf{x};\mathbf{p});\mathbf{q}))\|^2_{span(\mathbf{A}_i)^\perp} \\ & + \|\mathbf{A}_0(\mathbf{x}) + \sum_{i=1}^{M} \lambda_i \mathbf{A}_i(\mathbf{x}) - \mathbf{I}(\mathbf{N}(\mathbf{W}(\mathbf{x};\mathbf{p});\mathbf{q}))\|^2_{span(\mathbf{A}_i)} \\ = \; & \|\mathbf{A}_0(\mathbf{x}) - \mathbf{I}(\mathbf{N}(\mathbf{W}(\mathbf{x};\mathbf{p});\mathbf{q}))\|^2_{span(\mathbf{A}_i)^\perp} \\ & + \|\mathbf{A}_0(\mathbf{x}) + \sum_{i=1}^{M} \lambda_i \mathbf{A}_i(\mathbf{x}) - \mathbf{I}(\mathbf{N}(\mathbf{W}(\mathbf{x};\mathbf{p});\mathbf{q}))\|^2_{span(\mathbf{A}_i)} \end{aligned} \tag{3.22}$$

Clearly, any vector projected (spanned) in the eigenvector space $span(\mathbf{A}_i)$ can be seamlessly expressed by a linear combination of these eigenvectors $\mathbf{A}_i(\mathbf{x})$. So, no matter how $\mathbf{p}$ and $\mathbf{q}$ are chosen, the second term in (3.22) could always be minimized to 0. And we have,

$$\lambda_i = \mathbf{A}_i^T(\mathbf{x}) \cdot [\mathbf{I}(\mathbf{N}(\mathbf{W}(\mathbf{x};\mathbf{p});\mathbf{q})) - \mathbf{A}_0(\mathbf{x})] \tag{3.23}$$

For the first term in (3.22), we have to resort to ICIA technology. Unlike forward additive algorithm, inverse compositional algorithm computes the incremental warp with respect to the template $\mathbf{A}_0(\mathbf{x})$.

$$\|\mathbf{A}_0(\mathbf{N}(\mathbf{W}(\mathbf{x};\delta\mathbf{p});\delta\mathbf{q}) - \mathbf{I}(\mathbf{N}(\mathbf{W}(\mathbf{x};\mathbf{p});\mathbf{q}))\|^2_{span(\mathbf{A}_i)^\perp} \tag{3.24}$$

In [15], ICIA AAM fitting algorithm considering global shape transform concatenates the modified steepest descent images (MSDI) for each of the four normalized similarity parameters (q_1, q_2, q_3, q_4) and MSDI for AAM

shape model parameters $(p_1, p_2, \cdots, p_v)$. If we look on global shape normalization as a linear transform which is also able to be represented by a warp as

$$\mathbf{N}(\mathbf{x}; \mathbf{q}) = \mathbf{s}_0 + \sum_{i=1}^{4} q_i \mathbf{s}_i^* \tag{3.25}$$

, and further if we pick up the orthogonal vectors as follows,

$$\begin{aligned} \mathbf{s}_1^* = \mathbf{s}_0 &= (x_1^0, x_2^0, \cdots, x_v^0, y_1^0, y_2^0, \cdots, y_v^0)^T \\ \mathbf{s}_2^* &= (-y_1^0, -y_2^0, \cdots, -y_v^0, x_1^0, x_2^0, \cdots, x_v^0)^T \\ \mathbf{s}_3^* &= (1, 1, \cdots, 1, 0, 0, \cdots, 0)^T \\ \mathbf{s}_4^* &= (0, 0, \cdots, 0, 1, \cdots, 1, 1)^T \end{aligned} \tag{3.26}$$

it's apparent that all the vectors including both $\mathbf{s}_i, 1 \leq i \leq N$ and $\mathbf{s}_i^*, 1 \leq i \leq 4$ are orthogonal to each other. If we normalize $\mathbf{s}_3^*$ and $\mathbf{s}_4^*$, we could get an orthonormal set.

Chapter 4

AAM Implementation

4.1 Basic AAM Implementation

4.1.1 An Undeterminable Linear System

How to determine the precomputed $\mathbf{H}$ in (3.10) is essential for basic AAM fitting algorithm. By systematically displacing each parameter from the known optimal value on training images, $\frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}}$ could be estimated by numeric differentiation, so that $\mathbf{H}$ could be computed as well. In terms of (3.9), this precomputed $\mathbf{H}$, with $\mathbf{r}(\mathbf{c})$ which is the error texture between current extracted texture and the modeled texture, will be used to estimate $\delta\mathbf{c}$ in later AAM fitting iterations. However, how to realize the implementation to calculate this intermediate $\frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}}$, equally, $\mathbf{H}$, is an undeterminable issue. Let's first propose the numeric differentiation method in steps:

1. Given a bunch of known $\delta\mathbf{c}$, which should be representative enough, compute the texture differences $\mathbf{r}(\mathbf{c})$

2. Carry out linear regression to estimate $\mathbf{H}$ according to (3.9)

In this way, the numeric differentiation method turns out a huge hopelessly undetermined system as follows:

$$\mathbf{V} = \mathbf{UG} \tag{4.1}$$

where $\mathbf{V}$ is composed of all experimental parametric displacements, each column is a displacement; $\mathbf{G}$ is composed of all error textures between the extracted textures and the modeled textures under current experimental parametric displacement, each column is an error texture; and $\mathbf{U}$ is the linear regression matrix that we are trying to solve, which is negative value of $\mathbf{H}$.

Let's do a simple calculation as an evidence that the above linear system is absolutely undeterminable. For IMM dataset, totally, there are 240 images of resolution 640*480. After training, there are 26,753 pixels in the template face. And, the training results show there are 12 concatenated parameters to represent the concatenated AAM. If we take the global shape transform into our consideration as well, there will be another 4 parameters to describe this AAM model. Moreover, let's suppose we treat all the training images just as gray-level images (Single channel. If color images, there will be two times more data), for each image among the 240 training images, a vector of size 26,753 will be extracted, which is expected to be linearly represented by 12+4=16 parameters. So, as for displacement experiments, if only 4 displacements will be adopted for each parameter, there will be 240*16*4=15,360 experiments in total. Each experiment is actually a mapping between the 16 parameters and 26,753 pixels. With the above presumption, $\mathbf{V}$ is of size 16*15,360; $\mathbf{U}$ is of size 16*26,753, which is to be determined; $\mathbf{G}$ is of size 26,753*15,360. Apparently, in order to calculate $\mathbf{U}$, we need to get the inverse of the huge matrix $\mathbf{G}$ stored in float or double data type, which is of $26,753 * 15,360 * 4/1024^3 = 1.53$ giga data or so. In fact, normal computers nowadays can't even load such a huge matrix into the memory at one time. Therefore, the direct numerical differentiation method is impractical.

4.1.2 Principal Component Regression

Presented in the online documentation by Stegmann [18], a feasible method to tackle the above difficulty could be principal component regression or reduced rank multivariate linear regression [17]. By projecting the rather large matrix $\mathbf{G}$ into a k-dimensional subspace which captures the major part of the variation in $\mathbf{G}$, the above undeterminable system could possibly find its way out. Let the k eigenvectors of the error texture product $\mathbf{G^T G}$ corresponding to the k largest eigenvalues $(\lambda_1, \cdots, \lambda_k)$ be denoted by $\mathbf{\Phi_k}$ and let $\mathbf{\Lambda_k}$ be equal to $diag(\lambda_1 \cdots \lambda_k)$. According to traits of an eigen system, we have

$$\mathbf{G^T G \Phi_k} = \mathbf{\Phi_k \Lambda_k} \tag{4.2}$$

However, in the above situation for IMM dataset, $\mathbf{G^T G}$ is of size 15,360*15,360, which is still too large for PCA to obtain the eigenvectors and eigenvalues.

What's more, in the real implementation of AAM-API package, Stegmann circumvents this huge matrix inverse calculation by picking up just a part of displacement experiments to calculate $\mathbf{H}$, instead of using all displacement experiments.

4.1.3 Inverse of Column-wise Matrices

Numerically, it's possible to compute the inverse of this large matrix by partitioning it into several small blocks.

For simplicity, let's just suppose two blocks are applied in (4.1). Thus we have

$$(\mathbf{V_1}, \mathbf{V_2}) = \mathbf{U}(\mathbf{G_1}, \mathbf{G_2}) \qquad \mathbf{G_1} \in \Re^{t*s1} \quad \mathbf{G_2} \in \Re^{t*s2} \quad t \geq s1 + s2 \tag{4.3}$$

Normally, the pseudoinverse matrix of $(\mathbf{G_1}, \mathbf{G_2})$ could be given out as

$$(\mathbf{G_1}, \mathbf{G_2})^+ = ((\mathbf{G_1}, \mathbf{G_2})^{\mathbf{T}}(\mathbf{G_1}, \mathbf{G_2}))^{-\mathbf{1}}(\mathbf{G_1}, \mathbf{G_2})^{\mathbf{T}} \tag{4.4}$$

which requires s-dimensional square matrix inversion, where $s = s1 + s2$.

According to [20], the block matrix pseudoinverse could be calculated from two submatrix pseudoinverses, which cost $s1$ and $s2$-square matrix inversions, respectively.

$$(\mathbf{G_1}, \mathbf{G_2})^+ = \begin{pmatrix} (\mathbf{P}^{\perp}_{\mathbf{G_2}}\mathbf{G_1})^+ \\ (\mathbf{P}^{\perp}_{\mathbf{G_1}}\mathbf{G_2})^+ \end{pmatrix} \tag{4.5}$$

The $\mathbf{P}$ matrices are just two orthogonal projection matrices as defined in the following:

$$\begin{aligned} \mathbf{P}^{\perp}_{\mathbf{G_1}} &= \mathbf{I} - \mathbf{G_1}(\mathbf{G_1}^T\mathbf{G_1})^{-1}\mathbf{G_1}^T \\ \mathbf{P}^{\perp}_{\mathbf{G_2}} &= \mathbf{I} - \mathbf{G_2}(\mathbf{G_2}^T\mathbf{G_2})^{-1}\mathbf{G_2}^T \end{aligned} \tag{4.6}$$

Some times, if we take quite a lot of experiments during training process, we can't guarantee for $\mathbf{G}$, there will always be more rows than columns, namely, $t \geq s$. However, pseudoinverse, has the following character:

$$(\mathbf{G}^+)^T = (\mathbf{G}^T)^+$$

Thus, we might simply transpose $\mathbf{G}$ to make sure there are more rows than columns for $\mathbf{G}^T$, so that we could calculate its inverse $(\mathbf{G}^T)^+$ by inverse of block-partitioned matrix technology first, then transpose it back to obtain $\mathbf{G}^+$.

4.1.4 Inverse of Block-wise Matrices

According to [19] and [20], we might be able to only calculate the inverses of even smaller matrices as deduced in the following:

Let's suppose:

$$\mathbf{G} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$$

Consider the matrix-vector equation:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{E} \\ \mathbf{F} \end{bmatrix} = \begin{bmatrix} \mathbf{J} \\ \mathbf{K} \end{bmatrix}$$

If we multiply the top equation by $-\mathbf{CA}^{-1}$ and add to the bottom we may obtain:

$$(\mathbf{D} - \mathbf{CA}^{-1}\mathbf{B})\mathbf{F} = \mathbf{K} - \mathbf{CA}^{-1}\mathbf{J}$$

It's not hard for us to calculate $\mathbf{F}$ now. $(\mathbf{D} - \mathbf{CA}^{-1}\mathbf{B})$ is called the Schur complement [25] of $\mathbf{A}$ and now we denote it as $\mathbf{S_A}$. If we substitute the resolved $\mathbf{F}$ into the top equation, we may get: $\mathbf{AE} + \mathbf{BF} = \mathbf{J}$, from which $\mathbf{E}$ could be calculated consequently.

It's not hard to deduce that:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{CA}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{S_A} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$$

Alternatively, if we multiply the bottom equation by $-\mathbf{BD}^{-1}$ and add to the top we may obtain:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{BD}^{-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{S_D} & \mathbf{0} \\ \mathbf{0} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{D}^{-1}\mathbf{C} & \mathbf{I} \end{bmatrix}$$

where $\mathbf{S_D} = \mathbf{A} - \mathbf{BD}^{-1}\mathbf{C}$ is the Schur complement of $\mathbf{D}$.

Now, we may deduce the blockwise matrix inversion formulae:

$$\begin{aligned} \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} &= \begin{bmatrix} \mathbf{I} & -\mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{S_A^{-1}} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{CA}^{-1} & \mathbf{I} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{BS_A^{-1}CA}^{-1} & -\mathbf{A}^{-1}\mathbf{BS_A^{-1}} \\ -\mathbf{S_A^{-1}CA}^{-1} & \mathbf{S_A^{-1}} \end{bmatrix} \end{aligned} \tag{4.7}$$

Similarly, we have:

$$\begin{aligned} \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} &= \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{D}^{-1}\mathbf{C} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{S_D^{-1}} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\mathbf{BD}^{-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{S_D^{-1}} & -\mathbf{S_D^{-1}BD}^{-1} \\ -\mathbf{D}^{-1}\mathbf{CS_D^{-1}} & \mathbf{D}^{-1} + \mathbf{D}^{-1}\mathbf{CS_D^{-1}BD}^{-1} \end{bmatrix} \end{aligned} \tag{4.8}$$

Comparing the left top item of (4.7) and (4.8), a very important equation could be inferred:

$$(\mathbf{A} - \mathbf{BD}^{-1}\mathbf{C})^{-1} = \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}(\mathbf{D} - \mathbf{CA}^{-1}\mathbf{B})^{-1}\mathbf{CA}^{-1}$$

Both (4.7) and (4.8) afford a possible method to simplify the huge matrix inversion to a block-wise matrix inversion with four smaller matrices. However, it's still not able to deal with all situations when the training images are of higher resolutions or more displacement experiments are taken in our training process.

4.1.5 Alternative Adopted In AAM-API

In order to avoid calculating inverse of the huge matrix, two possible substitutive schemes could be:

1. Multivariate linear regression.
2. Simply averaging all $\frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}}$.

For the first scheme, a preestimated matrix $\mathbf{H}$ should be afforded first. Then, every training sample is loaded iteratively to adjust $\mathbf{H}$. Thus, the memory storage issue is not a problem any longer. The calculation stops until all samples meet their satisfaction, or a predefined criterion is met. Apparently, this method is quite time-consuming, and how to define the criterion to stop the iterations is difficult.

In the implementation of AAM-API [18], Stegmann calculate the average of $\frac{\partial \mathbf{r}(\mathbf{c})}{\partial \mathbf{c}}$ over the training dataset after each parameter c_i in $\mathbf{c}$ is displaced from the known optimal value. In fact, we may do it in this way:

$$\frac{\partial \mathbf{r}(\mathbf{c})}{\partial c_i} = \frac{\mathbf{r}(\mathbf{c} + \delta \mathbf{c_1}) - \mathbf{r}(\mathbf{c} + \delta \mathbf{c_2})}{\delta c_i^1 - \delta c_i^2} \tag{4.9}$$

where $\delta \mathbf{c_1}$ is a positive displacement with only the i-th parameter c_i is displaced by adding a positive value δc_i^1, while $\delta \mathbf{c_2}$ is a negative displacement with only the i-th parameter c_i is displaced by adding a negative value δc_i^2.

4.2 ICIA AAM Implementation

In AAM model, $\mathbf{p}$ is a vector of N synthesized parameters to describe a particular shape, i.e., the labeled points' coordinates. With a specific $\mathbf{p}$, a corresponding shape coordinate vector $\mathbf{s}$ could be obtained, which could be used to construct a triangle mesh. This built triangle mesh should have the same inner triangulation structure as the Delaunay Triangulation mesh of the trained template image $\mathbf{s}_0$. Now, let's analyze how to calculate (3.15) during implementation item by item.

- To calculate $\mathbf{I}(\mathbf{W}(\mathbf{x}; \mathbf{p}))$, we first warp the coordinates $\mathbf{x}$ over the template image into the coordinates $\mathbf{W}(\mathbf{x}; \mathbf{p})$ in the input image with the known parameters $\mathbf{p}$. In fact, every concerned pixel in the template image lies in a triangle in the Delaunay Triangulation mesh, and we could look on all the points inside one triangle have the same way to

warp in terms of the three shared vertexes. For simplicity, affine transform can be used here as a way to warp, which is called Piecewise Affine Warping in [15]. After the affine transform, the intensity of each warped point could be simply interpolated in the input image by any kind of interpolation method, such as bilinear interpolation.

- $\mathbf{A}_0(\mathbf{x})$ is just the intensities across all pixels $\mathbf{x}$ within the mean shape of the template image.

- To compute $\nabla\mathbf{A}_0\dfrac{\partial\mathbf{W}}{\partial\mathbf{p}}\delta\mathbf{p}$, $\delta\mathbf{p}$ is a column vector of size N, which is to be determined for updating the N-size shape parameters $\mathbf{p}$ during the iteration. As mentioned in the last subsection 3.2.4, the modified steepest descent image $\mathbf{MSDI} = \nabla\mathbf{A}_0\dfrac{\partial\mathbf{W}(\mathbf{x};\mathbf{p})}{\partial\mathbf{p}}$ is just a matrix of size $(3u) * N$ or $u * N$, in which, $\nabla\mathbf{A}_0$ represents the gradient vector across all the pixels inside the mean shape of the template image, in $\mathbf{x}$ and $\mathbf{y}$ directions respectively; and $\dfrac{\partial\mathbf{W}}{\partial\mathbf{p}}$ is the Jacobian of the warp. For each warped pixel: $\mathbf{w}(\mathbf{x};\mathbf{p}) = (\mathbf{w}_x(\mathbf{x};\mathbf{p})), (\mathbf{w}_y(\mathbf{x};\mathbf{p}))^T$. we have,

$$\frac{\partial\mathbf{w}}{\partial\mathbf{p}} = \begin{pmatrix} \frac{\partial\mathbf{w}_x}{\partial p_1} & \frac{\partial\mathbf{w}_x}{\partial p_2} & \cdots & \frac{\partial\mathbf{w}_x}{\partial p_N} \\ \frac{\partial\mathbf{w}_y}{\partial p_1} & \frac{\partial\mathbf{w}_y}{\partial p_2} & \cdots & \frac{\partial\mathbf{w}_y}{\partial p_N} \end{pmatrix} \tag{4.10}$$

For every item in the first row in the above array, the scalar $\mathbf{w}_x$ is the x coordinate of the warped point $\mathbf{w}(\mathbf{x};\mathbf{p})$, the scalar p_i refers to the ith element of the parameter vector $\mathbf{p}$. Simply applying the chain rule to $\dfrac{\partial\mathbf{w}_x}{\partial p_i}$ gives:

$$\frac{\partial\mathbf{w}_x}{\partial p_i} = \sum_{j=1}^{v}[\frac{\partial\mathbf{w}_x}{\partial x_j}\frac{\partial x_j}{\partial p_i} + \frac{\partial\mathbf{w}_x}{\partial y_j}\frac{\partial y_j}{\partial p_i}] \tag{4.11}$$

Similarly, for every item in the second row in (4.10), we have:

$$\frac{\partial\mathbf{w}_y}{\partial p_i} = \sum_{j=1}^{v}[\frac{\partial\mathbf{w}_y}{\partial x_j}\frac{\partial x_j}{\partial p_i} + \frac{\partial\mathbf{w}_y}{\partial y_j}\frac{\partial y_j}{\partial p_i}] \tag{4.12}$$

In fact, every pixel $\mathbf{x} = (x, y)^T$ inside the mean shape of the template image lies in a triangle. Let's suppose the three vertexes of the triangle containing the point $\mathbf{x}$ are $\mathbf{x}_r^0 = (x_r^0, y_r^0)^T$, $\mathbf{x}_s^0 = (x_s^0, y_s^0)^T$, and $\mathbf{x}_t^0 = (x_t^0, y_t^0)^T$. Any point inside this triangle $\mathbf{x}$ can be written as:

$$\mathbf{x} = \alpha\mathbf{x}_r^0 + \beta\mathbf{x}_s^0 + \gamma\mathbf{x}_t^0 \tag{4.13}$$

where

$$\begin{aligned}
\alpha &= \frac{x_s^0 y_t^0 - y_s^0 x_t^0 - x y_t^0 + y x_t^0 - y x_s^0 + x y_s^0}{x_s^0 y_t^0 - x_r^0 y_t^0 - x_s^0 y_r^0 - y_s^0 x_t^0 + y_r^0 x_t^0 + y_s^0 x_r^0} \\
\beta &= \frac{x y_t^0 - x_r^0 y_t^0 - x y_r^0 - y x_t^0 + y_r^0 x_t^0 + y x_r^0}{x_s^0 y_t^0 - x_r^0 y_t^0 - x_s^0 y_r^0 - y_s^0 x_t^0 + y_r^0 x_t^0 + y_s^0 x_r^0} \\
\gamma &= \frac{y x_s^0 - y_r^0 x_s^0 - y x_r^0 - x y_s^0 + x_r^0 y_s^0 + x y_r^0}{x_s^0 y_t^0 - x_r^0 y_t^0 - x_s^0 y_r^0 - y_s^0 x_t^0 + y_r^0 x_t^0 + y_s^0 x_r^0}
\end{aligned} \tag{4.14}$$

It's obvious that $\alpha + \beta + \gamma = 1$.
To calculate the warped point $\mathbf{w}(\mathbf{x}; \mathbf{p})$, by using the same α, β and γ, i.e., an affine transform, we could have:

$$\mathbf{w}(\mathbf{x}; \mathbf{p}) = \alpha \mathbf{x}_r + \beta \mathbf{x}_s + \gamma \mathbf{x}_t \tag{4.15}$$

where $\mathbf{x}_r$, $\mathbf{x}_s$, and $\mathbf{x}_t$ are built by (2.4), respectively corresponding to $\mathbf{x}_r^0$, $\mathbf{x}_s^0$, and $\mathbf{x}_t^0$. From (4.15), it's easy to deduce:

$$\frac{\partial \mathbf{w}_x}{\partial x_r} = \frac{\partial \mathbf{w}_y}{\partial y_r} = \alpha \quad \frac{\partial \mathbf{w}_x}{\partial x_s} = \frac{\partial \mathbf{w}_y}{\partial y_s} = \beta \quad \frac{\partial \mathbf{w}_x}{\partial x_t} = \frac{\partial \mathbf{w}_y}{\partial y_t} = \gamma \tag{4.16}$$

$$\frac{\partial \mathbf{w}_x}{\partial y_r} = \frac{\partial \mathbf{w}_y}{\partial x_r} = 0 \quad \frac{\partial \mathbf{w}_x}{\partial y_s} = \frac{\partial \mathbf{w}_y}{\partial x_s} = 0 \quad \frac{\partial \mathbf{w}_x}{\partial y_t} = \frac{\partial \mathbf{w}_y}{\partial x_t} = 0 \tag{4.17}$$

Obviously, all the other vertexes built by (2.4) have nothing to do with this warped point $\mathbf{w}(\mathbf{x}; \mathbf{p})$. Therefore, for this specific warped point, we have $\dfrac{\partial \mathbf{w}_x}{\partial x_m} = \dfrac{\partial \mathbf{w}_y}{\partial y_m} = 0$, where $m \neq r, s, t$. Thus, in (4.11) and (4.12), there are only three summation items are not zero. Those three items are just corresponding to the three vertexes of the triangle containing the point $\mathbf{x}$. For a specific point $\mathbf{x}$, (4.11) and (4.12) can be deducted to:

$$\begin{aligned}
\frac{\partial \mathbf{w}_x}{\partial p_i} &= \alpha \frac{\partial x_r}{\partial p_i} + 0 + \beta \frac{\partial x_s}{\partial p_i} + 0 + \gamma \frac{\partial x_t}{\partial p_i} + 0 \\
\frac{\partial \mathbf{w}_y}{\partial p_i} &= \alpha \frac{\partial y_r}{\partial p_i} + 0 + \beta \frac{\partial y_s}{\partial p_i} + 0 + \gamma \frac{\partial y_t}{\partial p_i} + 0
\end{aligned} \tag{4.18}$$

Now, implementing partial derivative in terms of $\mathbf{p}$ on both sides of (2.4), we have:

$$\begin{pmatrix} \frac{\partial x_1}{\partial p_1} & \frac{\partial x_1}{\partial p_2} & \cdots & \frac{\partial x_1}{\partial p_N} \\ \frac{\partial y_1}{\partial p_1} & \frac{\partial y_1}{\partial p_2} & \cdots & \frac{\partial y_1}{\partial p_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial x_v}{\partial p_1} & \frac{\partial x_v}{\partial p_2} & \cdots & \frac{\partial x_v}{\partial p_N} \\ \frac{\partial y_v}{\partial p_1} & \frac{\partial y_v}{\partial p_2} & \cdots & \frac{\partial y_v}{\partial p_N} \end{pmatrix} = 0 + \partial \begin{pmatrix} p_1\mathbf{s}_1^{x_1} + p_2\mathbf{s}_2^{x_1} + \cdots + p_N\mathbf{s}_N^{x_1} \\ p_1\mathbf{s}_1^{y_1} + p_2\mathbf{s}_2^{y_1} + \cdots + p_N\mathbf{s}_N^{y_1} \\ \cdots \\ p_1\mathbf{s}_1^{x_v} + p_2\mathbf{s}_2^{x_v} + \cdots + p_N\mathbf{s}_N^{x_v} \\ p_1\mathbf{s}_1^{y_v} + p_2\mathbf{s}_2^{y_v} + \cdots + p_N\mathbf{s}_N^{y_v} \end{pmatrix} / \partial \mathbf{p}$$

$$= \begin{pmatrix} \mathbf{s}_1^{x_1} & \mathbf{s}_2^{x_1} & \cdots & \mathbf{s}_N^{x_1} \\ \mathbf{s}_1^{y_1} & \mathbf{s}_2^{y_1} & \cdots & \mathbf{s}_N^{y_1} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{s}_1^{x_v} & \mathbf{s}_2^{x_v} & \cdots & \mathbf{s}_N^{x_v} \\ \mathbf{s}_1^{y_v} & \mathbf{s}_2^{y_v} & \cdots & \mathbf{s}_N^{y_v} \end{pmatrix} \tag{4.19}$$

where $\mathbf{s}_i^{x_j}$ denotes the component of $\mathbf{s}_i$ that corresponds to the position of x_j and similarly for $\mathbf{s}_i^{y_j}$.
Therefore,

$$\frac{\partial x_j}{\partial p_i} = \mathbf{s}_i^{x_j} \quad \frac{\partial y_j}{\partial p_i} = \mathbf{s}_i^{y_j} \tag{4.20}$$

Substituting (4.20) into (4.18), we have:

$$\frac{\partial \mathbf{w}_x}{\partial p_i} = \alpha\mathbf{s}_i^{x_r} + \beta\mathbf{s}_i^{x_s} + \gamma\mathbf{s}_i^{x_t} \quad \frac{\partial \mathbf{w}_y}{\partial p_i} = \alpha\mathbf{s}_i^{y_r} + \beta\mathbf{s}_i^{y_s} + \gamma\mathbf{s}_i^{y_t} \tag{4.21}$$

Bibliography

[1] F. L. Bookstein. Landmark methods for forms without landmarks: localizing groupdifferences in outline shape. In *Proceedings of the Workshop on Mathematical Methods in Biomedical Image Analysis*, volume 21-22, pages 279–289, June 1996.

[2] P. J. Burt. *Multiresolution Image Processing and Analysis*, chapter The pyramid as structure for efficient computation, pages 6–37. Springer-Verlag, 1984.

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

[4] T. Cootes and P. Kittipanya-ngam. Comparing variations on the active appearance model algorithm. In *10th British Machine Vision Conference (BMVC 2002)*, pages 837–846, Cardiff University, September 2002.

[5] T. Cootes and C. Taylor. Statistical models of appearance for computer vision. Technical report, Imaging Science and Biomedical Engineering, University of Manchester, March 8 2004.

[6] David Cristinacce and Tim Cootes. Feature detection and tracking with constrained local models (best science paper prize). In *Proceedings of 17th BMVA British Machine Vision Conference*, pages 929–938, Edinburgh, Scotland, September 2006.

[7] C. Goodall. Procrustes methods in the statistical analysis of shape. *Journal of the Royal Statistical Society. Series B (Methodological)*, 53(2):285–339, 1991.

[8] P. Jia. Research on feature selection. Master's thesis, Huazhong University of Science and Technology, May 2003. Excellent Master's Thesis

of Huazhong University of Science and Technology, Excellent Master's Thesis of Hubei Province, China.

[9] P. JIA. *Audio-visual based HMI for an Intelligent Wheelchair*. PhD thesis, University of Essex, July 2010.

[10] I.T. Jolliffe. *Principal Component Analysis*. Springer, Aberdeen, UK, 2nd edition, April 2002.

[11] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of Imaging understanding workshop*, pages 121–130, 1981.

[12] Bruce D. Lucas. *Generalized Image Matching by the Method of Differences*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 1984.

[13] S. Marcel, J. Keomany, and Y. Rodriguez. Robust-to-illumination face localisation using active shape models and local binary patterns. Technical Report IDIAPRR 06-47, July 2006.

[14] S. Marcel, Y. Rodriguez, and G. Heusch. On the recent use of local binary patterns for face authentication. *International Journal of Image and Video Processing, Special Issue on Facial Image Processing*, 2007. IDIAP-RR 06-34.

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

[16] S. Milborrow. Locating facial features with active shape models. Master's thesis, University of Cape Town, November 16 2007.

[17] J. O. Rawlings, S. G. Pantula, and D. A. Dickey. *Applied Regression Analysis : A Research Tool*. Springer, second edition, 1998.

[18] M. B. Stegmann. Active appearance models: Theory, extensions and cases, September 20 2000.

[19] H. Thornburg. Autoregressive modeling: Elementary least-squares methods. Technical report, Stanford University, Center for Computer Research in Music and Acoustics (CCRMA) Department of Music, Stanford University Stanford, California 94305, February 2006.

[20] Wikipedia. Block matrix pseudoinverse — wikipedia, the free encyclopedia, 2007. [Online; accessed 14-March-2008].

[21] Wikipedia. Delaunay triangulation — wikipedia, the free encyclopedia, 2007. [Online; accessed 28-December-2007].

[22] Wikipedia. Matrix calculus — wikipedia, the free encyclopedia, 2007. [Online; accessed 11-January-2008].

[23] Wikipedia. Point distribution model — wikipedia, the free encyclopedia, 2008. [Online; accessed 22-October-2008].

[24] Wikipedia. Procrustes analysis — wikipedia, the free encyclopedia, 2008. [Online; accessed 19-February-2008].

[25] Wikipedia. Schur complement — wikipedia, the free encyclopedia, 2008. [Online; accessed 17-March-2008].

[26] Wikipedia. Principal component analysis — wikipedia, the free encyclopedia, 2009. [Online; accessed 29-December-2009].

[27] S. Xin and H. Ai. Face alignment under various poses and expressions. In J. Tao, T. Tan, and R. W. Picard, editors, *The First International Conference on Affective Computing & Intelligent Interaction (ACII2005)*, volume LNCS 3784, pages 40–47, Beijing, China, October 22-24 2005. Computer Science and Technology Department, Tsinghua University, Springer-Verlag Berlin Heidelberg.

[28] F. Zuo and P. H. N. de With. Fast facial feature extraction using a deformable shape model with haar-wavelet based local texture attributes. In *Proceedings of IEEE Conference on ICIP*, pages 1425–1428, 2004.