

Active Appearance Model

JIA Pei

Email: jp4work@gmail.com

Vision Open Working Group

First Edition

May 29, 2010

Contents

1	Build AAM	1
1.1	AAM Shape	1
1.2	AAM Texture	1
2	AAM Fitting	3
2.1	Basic AAM Fitting	3
2.2	ICIA AAM Fitting	4
2.3	How to implement Inverse Compositional Image Alignment? .	7
3	AAM Fitting Considering A Global Shape Transform	11

Abstract

As one of the most important human-computer interfaces (HCI), face recognition has been researched over decades. Particularly, active shape models (ASM) and active appearance models (AAM) have been widely and successfully used for the face representation, which is able to tell the user's intention by calculating the user's face direction.

As the ancestor of ASM, active contour models (ACM) [2], also named snakes, were first proposed by Kass *et al.* [11], which can stretch and deform to fit image features to which they are attracted. C. Xu and J.L. Prince proposed GVF [17] as a new external force for snakes, the same conception as diffusion equations defined in diverse physics fields like fluid flow, etc., which greatly improve the snakes' fitting effect to the concave boundaries.

ASM was originally proposed by Cootes *et al.* [6, 8]. The advantage of ASM is that the instances of models "can only deform in ways found in a training set" [10], which is perfectly suitable for the representation of the specific deformable objects, for instance, face.

AAM was also first proposed by Cootes *et al.* [4, 9], and it's closely related to the concepts of Active Blobs [14] and Morphable Models [13]. AAM not only seeks to match the position of the model points as in ASM, but also match the representation of the texture over the object. [3, 7] compared ASM with AAM in detail, and [12] completely summarized how to implement ICIA AAM for face representation. As a practical application, CMU used AAM for the real-time driver head tracking and driver mental state estimation [1]. It's demonstrated that AAM is extraordinarily fast for head tracking. Meanwhile, AAM can accurately tell the face motion in real time.

Chapter 1

Build AAM

1.1 AAM Shape

The *shape* of an independent AAM is defined in the same way as ASM, which is composed of the coordinates of the v vertices that make up a mesh.

$$\mathbf{s} = [x_1, y_1, x_2, y_2, \dots, x_v, y_v]^T \quad (1.1)$$

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 :

$$\begin{aligned} \mathbf{s} &= \mathbf{s}_0 + \sum_{i=1}^N p_i \mathbf{s}_i \\ &= \mathbf{s}_0 + \mathbf{P}_s \mathbf{b}_s \end{aligned} \quad (1.2)$$

where $\mathbf{b}_s = (p_1, p_2, \dots, p_N)^T$ are the shape parameters and $\mathbf{P}_s = (\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_N)$ are just the orthonormal eigen vectors obtained from the training shapes.

1.2 AAM Texture

From [15], the *texture* is defined as the pixel intensities across the object (here, the face) in question (if necessary after a suitable normalization). Then, the texture of an AAM is a vector of intensities $\mathbf{A}(\mathbf{x})$ defined over the pixels $\mathbf{x} \in \mathbf{s}_0$, where $\mathbf{x} = (x, y)^T$. AAMs allow linear texture variation. This means that the texture $\mathbf{A}(\mathbf{x})$ can be expressed as a base texture $\mathbf{A}_0(\mathbf{x})$ plus a linear combination of M textures $\mathbf{A}_i(\mathbf{x})$:

$$\begin{aligned} \mathbf{A}(\mathbf{x}) &= \mathbf{A}_0(\mathbf{x}) + \sum_{i=1}^M \lambda_i \mathbf{A}_i(\mathbf{x}) \\ &= \mathbf{A}_0(\mathbf{x}) + \mathbf{P}_t \mathbf{b}_t, \forall \mathbf{x} \in \mathbf{s}_0 \end{aligned} \quad (1.3)$$

where $\mathbf{b}_t = (\lambda_1, \lambda_2, \dots, \lambda_M)^T$ are the texture parameters and $\mathbf{P}_t = (\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_M)$ are just the orthonormal eigen vectors obtained from the training textures.

For the IMM face database, the pixel number within the aligned mean face is 31,461. So, totally, there are $31,461 * 3 = 94,383$ elements in a single texture vector, taking three color channels—B, G, R into our consideration.

The following figure shows the mean face texture synthesized from the training IMM face dataset.



Figure 1.1: Mean/Template Face Texture

Chapter 2

AAM Fitting

2.1 Basic AAM Fitting

[5] clearly summarized the basic AAM fitting algorithm. An AAM should be able to 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{b}_s and the texture parameter \mathbf{b}_t have a linear relationship with the same parameter vector, say \mathbf{c} , i.e.:

$$\mathbf{b}_s = \mathbf{C}_s \mathbf{c} \quad (2.1)$$

$$\mathbf{b}_t = \mathbf{C}_t \mathbf{c} \quad (2.2)$$

Simply combining (1.2), (1.3), (2.1) and (2.2) gives:

$$\mathbf{s} = \mathbf{s}_0 + \mathbf{Q}_s \mathbf{c} \quad (2.3)$$

$$\mathbf{A}(\mathbf{x}) = \mathbf{A}_0(\mathbf{x}) + \mathbf{Q}_t \mathbf{c} \quad (2.4)$$

When AAM is put into the real face tracking application, our purpose is always trying to minimize the mean square error (MSE) of the difference between the trained model and the newly coming image:

$$\mathbf{r}(\mathbf{p}) = \mathbf{A}_i(\mathbf{x}) - \mathbf{A}_m(\mathbf{x}) \quad (2.5)$$

$$E(\mathbf{p}) = \mathbf{r}^T \mathbf{r} \quad (2.6)$$

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 $\delta \mathbf{p}$, so that (2.7) goes to minimum.

$$E(\mathbf{p} + \delta \mathbf{p}) = \mathbf{r}(\mathbf{p} + \delta \mathbf{p})^T \mathbf{r}(\mathbf{p} + \delta \mathbf{p}) \quad (2.7)$$

After each iteration, \mathbf{p} is updated to $\mathbf{p} + \delta\mathbf{p}$. The iterations will proceed continuously until $E(\mathbf{p} + \delta\mathbf{p})$ and $E(\mathbf{p})$ are of negligible difference. Thus, the problem is converted to how to calculate the updating value $\delta\mathbf{p}$ in terms of the current \mathbf{p} in each iteration. Apparently, in order to compute $\delta\mathbf{p}$, we first carry out 1-order Taylor expansion, and then carry out the partial derivative on (2.7).

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

Letting the above formula equal to zero gives:

$$\delta\mathbf{p} = -\mathbf{R}\mathbf{r}(\mathbf{p}) \quad (2.8)$$

$$\mathbf{R} = ((\frac{\partial \mathbf{r}(\mathbf{p})}{\partial \mathbf{p}})^T \frac{\partial \mathbf{r}(\mathbf{p})}{\partial \mathbf{p}})^{-1} \quad (2.9)$$

By systematically displacing each parameter from the known optimal value on training images and computing an average over the training dataset, $\frac{\partial \mathbf{r}(\mathbf{p})}{\partial \mathbf{p}}$ could be estimated by numeric differentiation.

2.2 ICIA AAM Fitting

[12] jointly put AAM into a Lucas-Kanade algorithm framework, and form the popular algorithm named inverse compositional image alignment (ICIA) for AAM fitting.

The goal of Lucas-Kanade algorithm is to find the best alignment by minimizing the square sum of the difference between a constant template image $T(\mathbf{x})$ and an input image $I(\mathbf{x})$ with respect to the warp parameters \mathbf{p} . Let $\mathbf{x} = (x, y)^T$ indicate the pixel coordinates and $\mathbf{W}(\mathbf{x}; \mathbf{p})$ denote the parameterized set of allowed warps, where $\mathbf{p} = (p_1, p_2, \dots, p_N)^T$ is a vector of N parameters. The warp $\mathbf{W}(\mathbf{x}; \mathbf{p})$ takes the pixel \mathbf{x} in the coordinate frame of the template T and maps it to the sub-pixel location $\mathbf{W}(\mathbf{x}; \mathbf{p})$ in

the coordinate frame of the image I . Naturally, the goal of Lucas-Kanade algorithm is to minimize

$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]^2 \quad (2.10)$$

To optimize the expression in Equation (2.10), Lucas-Kanade algorithm assumes that a current estimate of \mathbf{p} is known and then iteratively solves for increments to the parameters $\Delta\mathbf{p}$; i.e. the following expression is (approximately) minimized:

$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p} + \Delta\mathbf{p})) - T(\mathbf{x})]^2 \quad (2.11)$$

with respect to $\Delta\mathbf{p}$. Each iteration, the parameter \mathbf{p} is updated by $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$.

By performing first order Taylor expansion on the scalar $I(\mathbf{W}(\mathbf{x}; \mathbf{p} + \Delta\mathbf{p}))$ in terms of \mathbf{p} :

$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x})]^2 \quad (2.12)$$

where $\nabla I = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$ is the gradient of image I evaluated at $\mathbf{W}(\mathbf{x}; \mathbf{p})$. Note here: according to the definitions in “Matrix Calculus [16]”, ∇I should be a row vector of size $1 * 2$, $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ should be a matrix of size $2 * N$. Additionally, $\Delta\mathbf{p}$ is of size $N * 1$. Thus, it’s guaranteed that $\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p}$ in (2.12) is a scalar.

Now, we calculate the partial derivative on (2.12) in terms of $\Delta\mathbf{p}$:

$$2 \sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x})] [\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}] \quad (2.13)$$

and set it to equal zero, then transpose both sides of the equation, and solve the equation, we get:

$$\Delta\mathbf{p} = H^{-1} \sum_{\mathbf{x}} [\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^T [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))] \quad (2.14)$$

where, H is the Hessian Matrix as follows:

$$H = \sum_{\mathbf{x}} [\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^T [\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}] \quad (2.15)$$

Forwards Compositional Image Alignment

The compositional algorithm computes an incremental warp $\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})$ to be composed with the current warp $\mathbf{W}(\mathbf{x}; \mathbf{p})$ as:

$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{W}(\mathbf{x}; \Delta\mathbf{p}); \mathbf{p})) - T(\mathbf{x})]^2 \quad (2.16)$$

Taking the Taylor series expansion in terms of \mathbf{p} , we have:

$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \nabla I(\mathbf{W}(\mathbf{x}; \mathbf{p})) \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x})]^2 \quad (2.17)$$

Inverse Compositional Image Alignment

“The inverse compositional algorithm is a modification of the forwards compositional algorithm where the roles of the template and example images are reversed.” [12] Reversing the roles of the template image $T(\mathbf{x})$ and the example image $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$ results in the inverse compositional algorithm as:

$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{W}(\mathbf{x}; \Delta\mathbf{p}))]^2 \quad (2.18)$$

For $T(\mathbf{W}(\mathbf{x}; \Delta\mathbf{p}))$, taking the 1-order Taylor series expansion in terms of $\Delta\mathbf{p}$ at $\Delta\mathbf{p} = \mathbf{0}$, we have:

$$\begin{aligned} & \sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{W}(\mathbf{x}; \mathbf{0})) - \frac{\partial T(\mathbf{W}(\mathbf{x}; \Delta\mathbf{p}))}{\partial \mathbf{W}(\mathbf{x}; \Delta\mathbf{p})} \frac{\partial \mathbf{W}(\mathbf{x}; \Delta\mathbf{p})}{\partial \Delta\mathbf{p}} \Big|_{\Delta\mathbf{p}=\mathbf{0}} (\Delta\mathbf{p} - \mathbf{0})]^2 \\ &= \sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x}) - \frac{\partial T(\mathbf{W}(\mathbf{x}; \mathbf{0}))}{\partial \mathbf{W}(\mathbf{x}; \mathbf{0})} \frac{\partial \mathbf{W}(\mathbf{x}; \Delta\mathbf{p})}{\partial \Delta\mathbf{p}} \Big|_{\Delta\mathbf{p}=\mathbf{0}} \Delta\mathbf{p}]^2 \\ &= \sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x}) - \frac{\partial T(\mathbf{x})}{\partial \mathbf{x}} \frac{\partial \mathbf{W}(\mathbf{x}; \mathbf{t})}{\partial \mathbf{t}} \Big|_{\mathbf{t}=\mathbf{0}} \Delta\mathbf{p}]^2 \\ &= \sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x}) - \nabla T \frac{\partial \mathbf{W}(\mathbf{x}; \mathbf{p})}{\partial \mathbf{p}} \Big|_{\mathbf{p}=\mathbf{0}} \Delta\mathbf{p}]^2 \end{aligned} \quad (2.19)$$

We denote the solution to the above equation as:

$$\Delta\mathbf{p} = H^{-1} \sum_{\mathbf{x}} [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^T [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})] \quad (2.20)$$

where, H is the Hessian Matrix as follows:

$$H = \sum_{\mathbf{x}} [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^T [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}] \quad (2.21)$$

and the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is evaluated at $(\mathbf{x}; \mathbf{0})$ as in (2.19). Since there is nothing in the Hessian that depends on \mathbf{p} , it is constant across iterations and can be pre-computed.

With the above deduction, we can summarize the Lucas-Kanade Inverse Compositional Image Alignment algorithm as in figure (2.1).

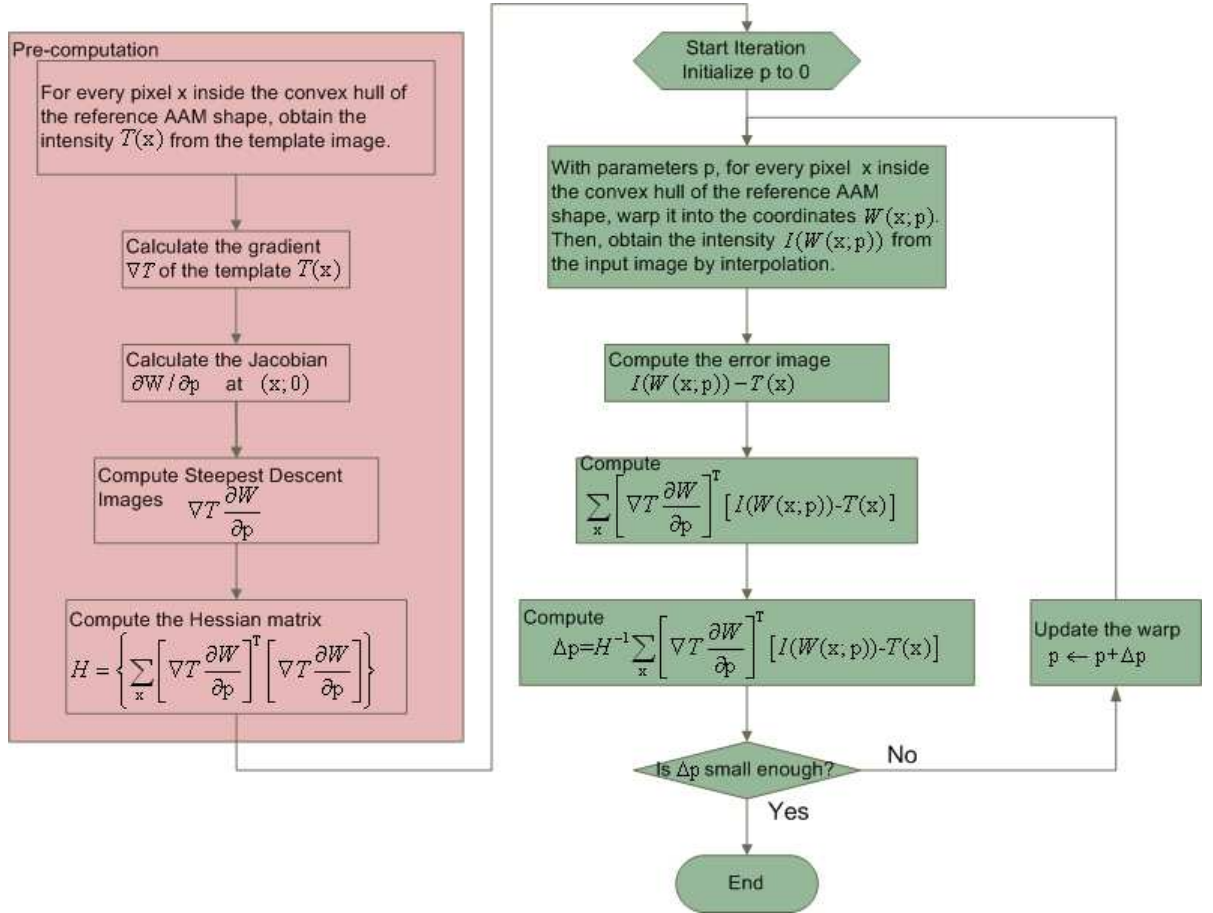


Figure 2.1: Lucas-Kanade Inverse Compositional Image Alignment

2.3 How to implement Inverse Compositional Image Alignment?

In AAM model, \mathbf{p} is a vector of N synthesized parameters to describe a particular shape, i.e., the 58 points' coordinates. During AAM fitting, let's

assume a known \mathbf{p} first. Thus, a triangle mesh could be constructed based on the computed 58 vertexes for the input image. This built triangle mesh should have the same inner structure as the mesh built by Delaunay Triangulation for the template image. For every point $\mathbf{x} = (x, y)$ inside the convex hull of the reference(mean) shape in the template image, we need to figure out a scalar according to (2.19), so that for all points within this convex hull, the sum of all the computed scalars will determine the texture error between the template image and newly input image after a specific warping. Now, let's analyze how to calculate (2.19) during implementation item by item.

- $T(\mathbf{x})$ is just the intensity of point \mathbf{x} in the template image.
- To calculate $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$, first, we warp the coordinates \mathbf{x} in 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 \mathbf{x} within the template image lies in a triangle in the mesh built by Delaunay Triangulation, and we could actually look on the points inside one triangle have the same way to warp according to the same three vertexes. For simplicity, affine transform can be used here as a way to warp, which is called Piecewise Affine Warping in [12]. After the affine transform, the intensity of that warped point could be simply interpolated in the input image by any kind of interpolation method, say, bilinear interpolation.
- As the 3rd item in $\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta \mathbf{p}$, $\Delta \mathbf{p}$ is a column vector of size N , which is to be determined to update the N -size shape parameters \mathbf{p} during the iteration. ∇T , apparently, is a 2-D vector for gradients at the coordinate \mathbf{x} in the template image. While $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is the Jacobian of the warp. Say, if $\mathbf{W}(\mathbf{x}; \mathbf{p}) = (W_x(\mathbf{x}; \mathbf{p}), (W_y(\mathbf{x}; \mathbf{p}))^T$, then

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_N} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_N} \end{pmatrix} \quad (2.22)$$

For every item in the first row in the above array, the scalar W_x is a coordinate about x of a warped point $W(\mathbf{x}; \mathbf{p})$, the scalar p_i refers to the i th element of the parameter vector \mathbf{p} . Simply applying the chain rule to $\frac{\partial W_x}{\partial p_i}$ gives:

$$\frac{\partial W_x}{\partial p_i} = \sum_{j=1}^v \left[\frac{\partial W_x}{\partial x_j} \frac{\partial x_j}{\partial p_i} + \frac{\partial W_x}{\partial y_j} \frac{\partial y_j}{\partial p_i} \right] \quad (2.23)$$

Similarly, for every item in the second row in (2.22), we have:

$$\frac{\partial W_y}{\partial p_i} = \sum_{j=1}^v \left[\frac{\partial W_y}{\partial x_j} \frac{\partial x_j}{\partial p_i} + \frac{\partial W_y}{\partial y_j} \frac{\partial y_j}{\partial p_i} \right] \quad (2.24)$$

In order to understand the above equation more thoroughly, we need to emphasize the 2-step calculation for $W(\mathbf{x}; \mathbf{p})$: 1) build the modeling shape – 58 vertexes by (1.2) – corresponding to $\frac{\partial x_j}{\partial p_i}$ and $\frac{\partial y_j}{\partial p_i}$; 2) calculate all the coordinates corresponding to the points inside the convex hull of the template image by using the same relative position – corresponding to $\frac{\partial W_x}{\partial x_j}$ and $\frac{\partial W_y}{\partial y_j}$.

In fact, every pixel \mathbf{x} in the convex hull of the template shape lies in a triangle. Let's suppose the three vertexes of the triangle containing the point \mathbf{x} are \mathbf{x}_r^0 , \mathbf{x}_s^0 , and \mathbf{x}_t^0 . Any internal point \mathbf{x} can be written as superposition:

$$\mathbf{x} = \alpha \mathbf{x}_r^0 + \beta \mathbf{x}_s^0 + \gamma \mathbf{x}_t^0 \quad (2.25)$$

where

$$\begin{aligned} \alpha &= \frac{x_s^0 y_t^0 - y_s^0 x_t^0 - x y_t^0 + y x_t^0 - y x_s^0 + x y_s^0}{x_s^0 y_t^0 - x_r^0 y_t^0 - x_s^0 y_r^0 - y_s^0 x_t^0 + y_r^0 x_t^0 + y_s^0 x_r^0} \\ \beta &= \frac{x y_t^0 - x_r^0 y_t^0 - x y_r^0 - y x_t^0 + y_r^0 x_t^0 + y x_r^0}{x_s^0 y_t^0 - x_r^0 y_t^0 - x_s^0 y_r^0 - y_s^0 x_t^0 + y_r^0 x_t^0 + y_s^0 x_r^0} \\ \gamma &= \frac{y x_s^0 - y_r^0 x_s^0 - y x_r^0 - x y_s^0 + x_r^0 y_s^0 + x y_r^0}{x_s^0 y_t^0 - x_r^0 y_t^0 - x_s^0 y_r^0 - y_s^0 x_t^0 + y_r^0 x_t^0 + y_s^0 x_r^0} \end{aligned} \quad (2.26)$$

It's obvious that $\alpha + \beta + \gamma = 1$.

To calculate the warped point $\mathbf{w}(\mathbf{x}; \mathbf{p})$ corresponding to \mathbf{x} , by using the same α, β and γ , i.e., an affine transform, we could have:

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) = \alpha \mathbf{x}_r + \beta \mathbf{x}_s + \gamma \mathbf{x}_t \quad (2.27)$$

where \mathbf{x}_r , \mathbf{x}_s , and \mathbf{x}_t are built by (1.2), respectively corresponding to \mathbf{x}_r^0 , \mathbf{x}_s^0 , and \mathbf{x}_t^0 .

From (2.27), it's easy to deduce:

$$\begin{aligned} \frac{\partial W_x}{\partial x_r} = \frac{\partial W_y}{\partial y_r} = \alpha & \quad \frac{\partial W_x}{\partial x_s} = \frac{\partial W_y}{\partial y_s} = \beta & \quad \frac{\partial W_x}{\partial x_t} = \frac{\partial W_y}{\partial y_t} = \gamma \\ \frac{\partial W_x}{\partial y_r} = \frac{\partial W_y}{\partial x_r} = 0 & \quad \frac{\partial W_x}{\partial y_s} = \frac{\partial W_y}{\partial x_s} = 0 & \quad \frac{\partial W_x}{\partial y_t} = \frac{\partial W_y}{\partial x_t} = 0 \end{aligned} \quad (2.28)$$

Obviously, for all the other vertexes built by (1.2), $\frac{\partial W_x}{\partial x_m} = \frac{\partial W_y}{\partial y_m} = 0$, where $m \neq r, s, t$. Thus, in (2.23) and (2.24), there are only three non-zero summation items. Those three items are just corresponding to the three vertexes of the triangle containing the point \mathbf{x} . Thus, for a specific point \mathbf{x} , (2.23) and (2.24) can be deduced to:

$$\begin{aligned}\frac{\partial W_x}{\partial p_i} &= \alpha \frac{\partial x_r}{\partial p_i} + 0 + \beta \frac{\partial x_s}{\partial p_i} + 0 + \gamma \frac{\partial x_t}{\partial p_i} + 0 \\ \frac{\partial W_y}{\partial p_i} &= \alpha \frac{\partial y_r}{\partial p_i} + 0 + \beta \frac{\partial y_s}{\partial p_i} + 0 + \gamma \frac{\partial y_t}{\partial p_i} + 0\end{aligned}\quad (2.29)$$

Now, implementing partial derivative in terms of \mathbf{p} on both sides of (1.2), we will get:

$$\begin{aligned}\begin{pmatrix} \frac{\partial x_1}{\partial p_1} & \frac{\partial x_1}{\partial p_2} & \cdots & \frac{\partial x_1}{\partial p_N} \\ \frac{\partial y_1}{\partial p_1} & \frac{\partial y_1}{\partial p_2} & \cdots & \frac{\partial y_1}{\partial p_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial x_v}{\partial p_1} & \frac{\partial x_v}{\partial p_2} & \cdots & \frac{\partial x_v}{\partial p_N} \\ \frac{\partial y_v}{\partial p_1} & \frac{\partial y_v}{\partial p_2} & \cdots & \frac{\partial y_v}{\partial p_N} \end{pmatrix} &= 0 + \frac{\partial \begin{pmatrix} p_1 \mathbf{s}_1^{x_1} + p_2 \mathbf{s}_2^{x_1} + \cdots + p_N \mathbf{s}_N^{x_1} \\ p_1 \mathbf{s}_1^{y_1} + p_2 \mathbf{s}_2^{y_1} + \cdots + p_N \mathbf{s}_N^{y_1} \\ \cdots \\ p_1 \mathbf{s}_1^{x_v} + p_2 \mathbf{s}_2^{x_v} + \cdots + p_N \mathbf{s}_N^{x_v} \\ p_1 \mathbf{s}_1^{y_v} + p_2 \mathbf{s}_2^{y_v} + \cdots + p_N \mathbf{s}_N^{y_v} \end{pmatrix}}{\partial \mathbf{p}} \\ &= \begin{pmatrix} \mathbf{s}_1^{x_1} & \mathbf{s}_2^{x_1} & \cdots & \mathbf{s}_N^{x_1} \\ \mathbf{s}_1^{y_1} & \mathbf{s}_2^{y_1} & \cdots & \mathbf{s}_N^{y_1} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{s}_1^{x_v} & \mathbf{s}_2^{x_v} & \cdots & \mathbf{s}_N^{x_v} \\ \mathbf{s}_1^{y_v} & \mathbf{s}_2^{y_v} & \cdots & \mathbf{s}_N^{y_v} \end{pmatrix}\end{aligned}\quad (2.30)$$

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,

$$\frac{\partial x_j}{\partial p_i} = \mathbf{s}_i^{x_j} \quad \frac{\partial y_j}{\partial p_i} = \mathbf{s}_i^{y_j}\quad (2.31)$$

Substituting (2.31) into (2.29), we have:

$$\begin{aligned}\frac{\partial W_x}{\partial p_i} &= \alpha \mathbf{s}_i^{x_r} + \beta \mathbf{s}_i^{x_s} + \gamma \mathbf{s}_i^{x_t} \\ \frac{\partial W_y}{\partial p_i} &= \alpha \mathbf{s}_i^{y_r} + \beta \mathbf{s}_i^{y_s} + \gamma \mathbf{s}_i^{y_t}\end{aligned}\quad (2.32)$$

Chapter 3

AAM Fitting Considering A Global Shape Transform

During the AAM model building, the shape model is built on basis of the normalized mesh, which removes translation, rotation and scale variances by Procrustes analysis. Clearly, after this normalization, the built shape model doesn't actually model the translation, rotation and scale. However, as for a testing image, we do need to take this similarity transform into consideration [12] [15].

Bibliography

- [1] S. Baker, I. Matthews, J. Xiao, R. Gross, T. Kanade, and T. Ishikawa. Real-time non-rigid driver head tracking for driver mental state estimation. In *11th World Congress on Intelligent Transportation Systems*, Nagoya, Japan, October 2004.
- [2] Andrew Blake and Michael Isard. *Active Contours*. Springer, 1998.
- [3] T. Cootes, G. Edwards, and C. Taylor. Comparing active shape models with active appearance models. In *10th British Machine Vision Conference (BMVC 1999)*, pages 173–182, Nottingham, UK, September 1999.
- [4] T. Cootes, G. Edwards, and C. Taylor. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):681–685, June 2001.
- [5] 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.
- [6] T. Cootes and C. Taylor. Active shape models - smart snakes. In *Proceedings of British Machine Vision Conference (BMVC 1992)*, pages 266–275. Springer-Verlag, 1992.
- [7] T. Cootes and C. Taylor. Statistical models of appearance for computer vision. Technical report, Ongoing Draft of the Technical Report on Active Shape Models and Active Appearance Models , Imaging Science and Biomedical Engineering, University of Manchester, March 8 2004.
- [8] T. Cootes, C. Taylor, D. Cooper, and J. Graham. Active shape models - their training and application. *Journal of Computer Vision and Image Understanding*, 61(1):38–59, January 1995.

- [9] G. Edwards, C. Taylor, and T. Cootes. Interpreting face images using active appearance models. In *Proceedings of 3rd International Conference on Automatic Face and Gesture Recognition (FG 1998)*, pages 300–305, Nara, Japan, April 14-16 1998.
- [10] G. Hamarneh. Active shape models, modeling shape variations and gray level information and an application to image search and classification. Technical report, The Imaging and Image Analysis Group, Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden, 1998.
- [11] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, pages 321–331, 1998.
- [12] I. Matthews and S. Baker. Active appearance models revisited. *International Journal of Computer Vision*, 60(2):135–164, November 2004.
- [13] S. Romdhani, V. Blanz, C. Basso, and T. Vetter. *Handbook of Face Recognition*, chapter 10. Morphable Models of Faces. Springer-Verlag, January 2004.
- [14] S. Sclaroff and J. Isidoro. Active blobs. In *Proceedings of IEEE International Conference on Computer Vision*, pages 1146–1153, Mumbai, India, 1998.
- [15] M. B. Stegmann. <http://www2.imm.dtu.dk/~aam/main>.
- [16] Wikipedia. Matrix calculus — wikipedia, the free encyclopedia, 2006. [Online; accessed 19-April-2007].
- [17] C. Xu and J.L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing*, 6:359–369, 1998.