MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 1537 September, 1995

C.B.C.L. Paper No. 122

Vectorizing Face Images by Interleaving Shape and Texture Computations

David Beymer

email: beymer@ai.mit.edu

Abstract

The correspondence problem in computer vision is basically a matching task between two or more sets of features. Computing feature correspondence is of great importance in computer vision, especially in the subfields of object recognition, stereo, and motion. In this paper, we introduce a *vectorized* image representation, which is a feature-based representation where correspondence has been established with respect to a reference image. The representation consists of two image measurements made at the feature points: shape and texture. Feature geometry, or shape, is represented using the (x, y) locations of features relative to the some standard reference shape. Image grey levels, or texture, are represented by mapping image grey levels onto the standard reference shape. Computing this representation is essentially a correspondence task, and in this paper we explore an automatic technique for "vectorizing" face images. Our face vectorizer alternates back and forth between computation steps for shape and texture, and a key idea is to structure the two computations so that each one uses the output of the other. Namely, the texture computation uses shape for geometrical normalization, and the shape computation uses the texture analysis to synthesize a "reference" image for finding correspondences. A hierarchical coarse-to-fine implementation is discussed, and applications are presented to the problems of facial feature detection and registration of two arbitrary faces.

This report describes research done at the Artificial Intelligence Laboratory and within the Center for Biological and Computational Learning. This research is sponsored by grants from the Office of Naval Research under contracts N00014-91-J-1270 and N00014-92-J-1879; by a grant from the National Science Foundation under contract ASC-9217041. Support for the A.I. Laboratory's artificial intelligence research is provided by ONR contract N00014-91-J-4038. The author is supported by a Howard Hughes Doctoral Fellowship from the Hughes Aircraft Company.

1 Introduction

The computation of correspondence is of great importance in computer vision, especially in the subfields of object recognition, stereo, and motion. The correspondence problem is basically a matching task between two or more sets of features. In the case of object recognition, one set of features comes from a prior object model and the other from an image of the object. In stereo and motion, the correspondence problem involves matching features across different images of the object, where the images may be taken from different viewpoints or over time as the object moves. Common feature points are often taken to be salient points along object contours such as corners or vertices.

A common representation for objects in recognition, stereo, and motion systems is feature-based; object attributes are recorded at a set of feature points. The set of feature points can be situated in either 3D as an object-centered model or in 2D as a view-centered description. To capture object geometry, one of the object attributes recorded at each feature is its position in 2D or 3D. Additionally, if the object has an detailed texture, one may be interested in recording the local surface albedo at each feature point or more simply the image brightness. Throughout this paper we refer to these two attributes respectively as shape and texture.

Given two or more sets of features, correspondence algorithms match features across the feature sets. We define a **vectorized representation** to be a feature-based representation where correspondence has been established relative to a fixed reference object or reference image. Computing the vectorized representation can be thought of as arranging the feature sets into ordered vectors so that the ith element of each vector refers to the same feature point for all objects. Given the correspondences in the vectorized representation, subsequent processing can do things like register images to models for recognition, and estimate object depth or motion.

In this paper, we introduce an algorithm for computing the vectorized representation for a class of objects like the human face. Faces present an interesting class of objects because of the variation seen across individuals in both shape and texture. The intricate structure of faces leads us to use a dense set of features to describe it. Once a dense set of feature correspondences have been computed between an arbitrary face and a "reference" face, applications such as face recognition and pose and expression estimation are possible. However, the focus of this paper is on an algorithm for computing a vectorized representation for faces.

The two primary components of the vectorized representation are shape and texture. Previous approaches in analyzing faces have stressed either one component or the other, such as feature localization or decomposing texture as a linear combination of eigenfaces (see Turk and Pentland [37]). The key aspect of our vectorization algorithm, or "vectorizer", is that the two processes for the analysis of shape and texture are coupled. That is, the shape and texture processes are coupled by making each process use the output of the other. The texture analysis uses shape for geometrical normalization, and shape analysis uses texture to synthesize a reference image for feature correspondence. Empirically, we have found that this links the two processes in a positive feedback loop. Iterating between the shape and texture steps causes the vectorized representation to converge after several iterations.

Our vectorizer is similar to the active shape model of Cootes, *et al.* [17][16][23] in that both iteratively fit a shape/texture model to the input. But there are interesting differences in the modeling of both shape and texture. In our vectorizer there is no model for shape; it is measured in a data-driven manner using optical flow. In active shape models, shape is modeled using a parametric, example-based method. First, an ensemble of shapes are processed using principal component analysis, which produces a set of "eigenshapes". New shapes are then written as linear combinations of these eigenshapes. Texture modeling in their approach, however, is weaker than in ours. Texture is only modeled locally along 1D contours at each of the feature points defining shape. Our approach models texture over larger regions – such as eyes, nose, and mouth templates – which should provide more constraint for textural analysis. In the future we intend to add a model for shape similar to active shape models, as discussed ahead in section 6.2.

In this paper, we start in section 2 by first providing a more concrete definition of our vectorized shape and texture representation. This is followed by a more detailed description of the coupling of shape and texture. Next, in section 3, we present the basic vectorization method in more detail. Section 4 discusses a hierarchical coarse-to-fine implementation of the technique. In section 5, we demonstrate two applications of the vectorizer, facial feature detection and the registration of two arbitrary faces. The latter application is used to map prototypical face transformations onto a face so that new "virtual" views can be synthesized (see Beymer and Poggio [11]). The paper closes with suggestions for future work, including an idea to generalize the vectorizer to multiple poses.

2 Preliminaries

2.1 Vectorized representation

As mentioned in the introduction, the vectorized representation is a feature-based representation where correspondence has been established relative to a fixed reference object or reference image. Computationally, this requires locating a set of features on an object and bringing them into correspondence with some prior reference feature set. While it is possible to define a 3D, object-centered vectorization, the vectorized representation in this paper will be based on 2D views of frontal views of the face. Thus, the representations for shape and texture of faces will be defined in 2D and measured relative to a 2D reference image.

Since the representation is relative to a 2D reference, first we define a standard feature geometry for the reference image. The features on new faces will then be measured relative to the standard geometry. In this paper, the standard geometry for frontal views of faces is

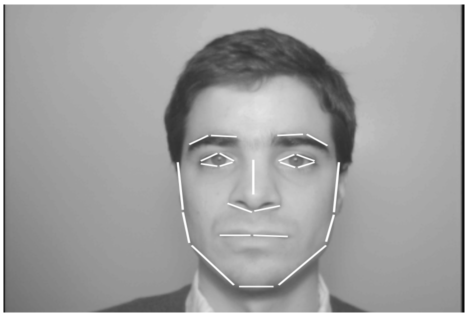

Figure 1: To define the shape of the prototypes off-line, manual line segment features are used. After Beier and Neely [5].

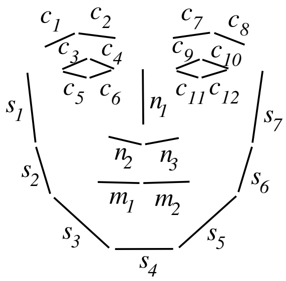

Figure 2: Manually defined shapes are averaged to compute the standard face shape.

defined by averaging a set of line segment features over an ensemble of "prototype" faces. Fig. 1 shows the line segment features for a particular individual, and Fig. 2 shows the average over a set of 14 prototype people. Features are assigned a text label (e.g. "c_1") so that corresponding line segments can be paired across images. As we will explain later in section 3.1, the line segment features are specified manually in an initial off-line step that defines the standard feature geometry.

The two components of the vectorized representation, shape and texture, can now be defined relative to this standard shape.

2.1.1 Shape

Given the locations of n feature points $f_1, f_2, \ldots, f_n$ in an image i_a, an "absolute" measure of 2D shape is represented by a vector $\mathbf{y}_a$ of length $2n$ consisting of the concatenation of the x and y coordinate values

$$\mathbf{y}_a = \begin{pmatrix} x_1 \\ y_1 \\ \vdots \\ x_n \\ y_n \end{pmatrix}.$$

This absolute representation for 2D shape has been widely used, including network-based object recognition (Poggio and Edelman [28]), the linear combinations approach to recognition (Ullman and Basri [38], Poggio [27]), active shape models (Cootes and Taylor [15], Cootes, *et al.* [17]) and face recognition (Craw and Cameron [18][19]).

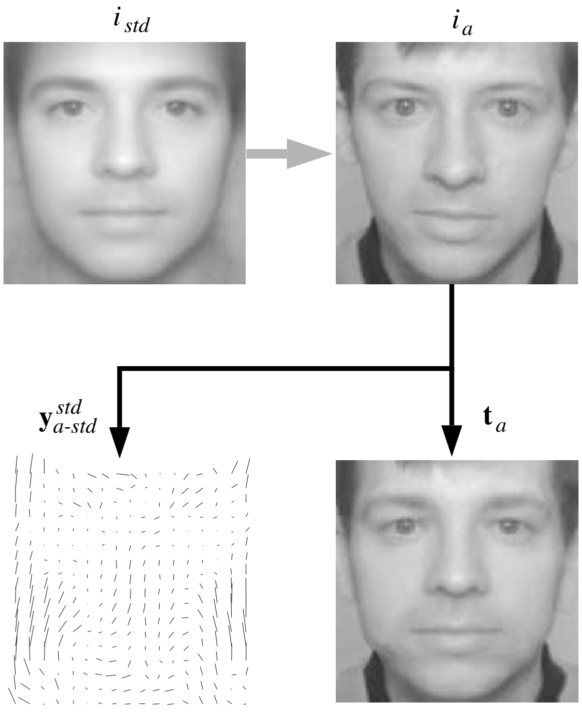

Figure 3: Our vectorized representation for image i_a with respect to the reference image i_{std} at standard shape. First, pixelwise correspondence is computed between i_{std} and i_a, as indicated by the grey arrow. Shape $\mathbf{y}^{std}_{a-std}$ is a vector field that specifies a corresponding pixel in i_a for each pixel in i_{std}. Texture $\mathbf{t}_a$ consists of the grey levels of i_a mapped onto the standard shape.

A relative shape measured with respect to a standard reference shape $\mathbf{y}_{std}$ is simply the difference

$$\mathbf{y}_a - \mathbf{y}_{std},$$

which we denote using the shorthand notation $\mathbf{y}_{a-std}$. The relative shape $\mathbf{y}_{a-std}$ is the difference in shape between the individual in i_a and the mean face shape.

To facilitate shape and texture operators in the run-time vectorization procedure, shape is spatially oversampled. That is, we use a pixelwise representation for shape, defining a feature point at each pixel in a subimage containing the face. The shape vector $\mathbf{y}_{a-std}$ can then be visualized as a vector field of correspondences between a face at standard shape and the given image i_a being represented. If there are n pixels in the face subimage being vectorized, then the shape vector consists of $2n$ values, a $(\delta x, \delta y)$ pair for each pixel. In this dense, pixelwise representation for shape, we need to keep track of the reference image, so the notation is extended to include the reference as a superscript $\mathbf{y}^{std}_{a-std}$. Fig. 3 shows the shape representation $\mathbf{y}^{std}_{a-std}$ for the image i_a. As indicated by the grey arrow, correspondences are measured relative to the reference face i_{std} at standard shape. (Image i_{std} in this case is mean grey level image; modeling grey level texture is discussed more in section 3.1.) Overall, the advantage of using a dense representation is that it allows a simple optical flow calculation to be used for computing shape and a simple 2D warping operator for geometrical normalization.

2.1.2 Texture

Our texture vector is a geometrically normalized version of the image i_a. That is, the geometrical differences among face images are factored out by warping the images to the standard reference shape. This strategy for representing texture has been used, for example, in the face recognition works of Craw and Cameron [18], and Shackleton and Welsh [33]. If we let shape $\mathbf{y}_{std}$ be the reference shape, then the geometrically normalized image $\mathbf{t}_a$ is given by the 2D warp

$$\mathbf{t}_a(x,y) = i_a(x + \mathbf{\Delta x}^{std}_{a-std}(x,y), y + \mathbf{\Delta y}^{std}_{a-std}(x,y)),$$

where $\mathbf{\Delta x}^{std}_{a-std}$ and $\mathbf{\Delta y}^{std}_{a-std}$ are the x and y components of $\mathbf{y}^{std}_{a-std}$, the pixelwise mapping between $\mathbf{y}_a$ and the standard shape $\mathbf{y}_{std}$. Fig. 3 in the lower right shows an example texture vector $\mathbf{t}_a$ for the input image i_a in the upper right.

If shape is sparsely defined, then texture mapping or sparse data interpolation techniques can be employed to create the necessary pixelwise level representation. Example sparse data interpolation techniques include using splines (Litwinowicz and Williams [24], Wolberg [40]), radial basis functions (Reisfeld, Arad, and Yeshurun [31]), and inverse weighted distance metrics (Beier and Neely [5]). If a pixelwise representation is being used for shape in the first place, such as one derived from optical flow, then texture mapping or data interpolation techniques can be avoided.

2.1.3 Separation of shape and texture

How cleanly have we separated the notions of shape and texture in the 2D representations just described? Ideally, the ultimate shape description would be a 3D one where the (x, y, z) coordinates are represented. Texture would be a description of local surface albedo at each feature point on the object. Such descriptions are common for the modeling of 3D objects for computer graphics, and it would be nice for vision algorithms to invert the imaging or "rendering" process from 3D models to 2D images.

What our 2D vectorized description has done, however, is to factor out and explicitly represent the salient aspects of 2D shape. The true spatial density of this 2D representation depends, of course, on the density of features defining standard shape, shown in our case in Fig. 2. Some aspects of 2D shape, such as lip or eyebrow thickness, will end up being encoded in our model for texture. However, one could extend the standard feature set to include more features around the mouth and eyebrows if desired. For texture, there are non-albedo factors confounded in the texture component, such as lighting conditions and the z-component of shape. Overall, though, remember that only one view of the object being vectorized is available, thus limiting our access to 3D information. We hope that the current definitions of shape and texture are a reasonable approximation to the desired decomposition.

Figure 4: Vectorizing face images: if we know who the person is and have prior example views i_a of their face, then we can manually warp i_a to standard shape, producing a reference $\mathbf{t}_a$. New images of the person can be vectorized by computing optical flow between $\mathbf{t}_a$ and the new input. However, if we do not have prior knowledge of the person being vectorized, we can still synthesize an approximation to $\mathbf{t}_a$, $\widehat{\mathbf{t}}_a$, by taking a linear combination of prototype textures.

2.2 Shape/texture coupling

One of the main results of this paper is that the computations for the shape and texture components can be algorithmically coupled. That is, shape can be used to geometrically normalize the input image prior to texture analysis. Likewise, the result of texture analysis can be used to synthesize a reference image for finding correspondences in the shape computation. The result is an iterative algorithm for vectorizing images of faces. Let us now explore the coupling of shape and texture in more detail.

2.2.1 Shape perspective

Since the vectorized representation is determined by an ordered set of feature points, computing the representation is essentially a feature finding or correspondence task. Consider this correspondence task under a special set of circumstances: we know who the person is, and we have prior example views of that person. In this case, a simple correspondence finding algorithm such as optical flow should suffice. As shown in the left two images of Fig. 4, first a prior example i_a of the person's face is manually warped in an off-line step to standard shape, producing a reference image $\mathbf{t}_a$. A new image of the same person can now be vectorized simply by running an optical flow algorithm between the image and reference $\mathbf{t}_a$.

If we have no prior knowledge of the person being vectorized, the correspondence problem becomes more difficult. In order to handle the variability seen in facial appearance across different people, one could imagine using many different example reference images that have been pre-warped to the standard reference shape. These reference images could be chosen, for example, by running a clustering algorithm on a large ensemble of example face images. This solution, however, introduces the problem of having to choose among the reference images for the final vectorization, perhaps based on a confidence measure in the correspondence algorithm.

Going one step further, in this paper we use a statistical model for facial texture in order to assist the correspondence process. Our texture model relies on the assumption, commonly made in the eigenface approach to face recognition and detection (Turk and Pentland [37], Pentland, *et al.* [26]), that the space of grey level images of faces is linearly spanned by a set of example views. That is, the geometrically normalized texture vector $\mathbf{t}_a$ from the input image i_a can be approximated as a linear combination of n prototype textures $\mathbf{t}_{p_j}, 1 \leq j \leq n$

$$\widehat{\mathbf{t}}_a = \sum_{j=1}^{n} \beta_j \mathbf{t}_{p_j}, \tag{1}$$

where the $\mathbf{t}_{p_j}$ are themselves geometrically normalized by warping them to the standard reference shape. The rightmost image of Fig. 4, for example, shows an approximation $\widehat{\mathbf{t}}_a$ that is generated by taking a linear combination of textures as in equation (1). If the vectorization procedure can estimate a proper set of β_j coefficients, then computing correspondences should be simple. Since the computed "reference" image $\widehat{\mathbf{t}}_a$ approximates the texture $\mathbf{t}_a$ of the input and is geometrically normalized, we are back to the situation where a simple correspondence algorithm like optical flow should work. In addition, the linear β_j coefficients act as a low dimensional code for representing the texture vector $\mathbf{t}_a$.

This raises the question of computing the β_j coefficients for the texture model. Let us now consider the vectorization procedure from the perspective of modeling texture.

2.2.2 Texture perspective

To develop the vectorization technique from the texture perspective, consider the simple eigenimage, or "eigenface", model for the space of grey level face images. The eigenface approach for modeling face images has been used recently for a variety of facial analysis tasks, including face recognition (Turk and Pentland [37], Akamatsu, *et al.* [2], Pentland, *et al.* [26]), reconstruction (Kirby and Sirovich [22]), face detection (Sung and Poggio [35], Moghaddam and Pentland [25]), and facial feature detection (Pentland, *et al.* [26]). The main assumption behind this modeling approach is that the space of grey level images of faces is linearly spanned by a set of example face images. To optimally represent this "face space", principal component analysis is applied to the example set, extracting an orthogonal set of eigenimages that define the dimensions of face space. Arbitrary faces are then represented by the set of coefficients computed by projecting the face onto the set of eigenimages.

One requirement on face images, both for the example set fed to principal components and for new images projected onto face space, is that they be geometrically normalized so that facial features line up across all images. Most normalization methods use a global transform, usually a similarity or affine transform, to align two or three major facial features. For example, in Pentland, *et al.* [26], the imaging apparatus effectively registers eyes, and Akamatsu, *et al.* [2] register the eyes and mouth.

However, because of the inherent variability of facial geometries across different people, aligning just a couple of features – such as the eyes – leaves other features misaligned. To the extent that some features are misaligned, even this normalized representation will confound differences in grey level information with differences in local facial geometry. This may limit the representation's generalization ability to new faces outside the original example set used for principal components. For example, a new face may match the texture of one particular linear combination of eigenimages but the shape may require another linear combination.

To decouple texture and shape, Craw and Cameron [18] and Shackelton and Welsh [33] represent shape separately and use it to geometrically normalize face texture by deforming it to a standard shape. Shape is defined by the (x, y) locations of a set of feature points, as in our definition for shape. In Craw and Cameron [18], 76 points outlining the eyes, nose, mouth, eyebrows, and head are used. To geometrically normalize texture using shape, image texture is deformed to a standard face shape, making it "shape free". This is done by first triangulating the image using the features and then texture mapping.

However, they did not demonstrate an effective automatic method for computing the vectorized shape/texture representation. This is mainly due to difficulties in finding correspondences for shape, where probably on the order of tens of features need to be located. Craw and Cameron [18] manually locate their features. Shackelton and Welsh [33], who focus on eye images, use the deformable template approach of Yuille, Cohen, and Hallinan [41] to locate eye features. However, for 19/60 of their example eye images, feature localization is either rated as "poor" or "no fit".

Note that in both of these approaches, computation of the shape and texture components have been separated, with shape being computed first. This differs from our approach, where shape and texture computations are interleaved in an iterative fashion. In their approach the link from shape to texture is present – using shape to geometrically normalize the input. But using a texture model to assist finding correspondences is not exploited.

2.2.3 Combining shape and texture

Our face vectorizer consists of two primary steps, a shape step that computes vectorized shape $\mathbf{y}^{std}_{a-std}$ and a texture step that uses the texture model to approximate the texture vector $\mathbf{t}_a$. Key to our vectorization procedure is linking the two steps in a mutually beneficial manner and iterating back and forth between the two until the representation converges. First, consider how the result of the texture step can be used to assist the shape step. Assuming for the moment that the texture step can provide an estimate $\widehat{\mathbf{t}}_a$ using equation (1), then the shape step estimates $\mathbf{y}^{std}_{a-std}$ by computing optical flow between the input and $\widehat{\mathbf{t}}_a$.

Next, to complete the loop between shape and texture, consider how the shape $\mathbf{y}^{std}_{a-std}$ can be used to compute the texture approximation $\widehat{\mathbf{t}}_a$. The shape $\mathbf{y}^{std}_{a-std}$ is used to geometrically normalize the input image using the backward warp

$$\mathbf{t}_a(\mathbf{x}) = i_a(\mathbf{x} + \mathbf{y}^{std}_{a-std}(\mathbf{x})),$$

where $\mathbf{x} = (x, y)$ is a 2D pixel location in standard shape. This normalization step aligns the facial features in the input image with those in the textures $\mathbf{t}_{p_j}$. Thus, when $\mathbf{t}_a$ is approximated in the texture step by projecting it onto the linear space spanned by the $\mathbf{t}_{p_j}$, facial features are properly registered.

Given initial conditions for shape and texture, our proposed system switches back and forth between texture and shape computations until a stable solution is found. Because of the manner in which the shape and texture computations feed back on each other, improving one component improves the other: better correspondences mean better feature alignment for textural analysis, and computing a better textural approximation improves the reference image used for finding correspondences. Empirically, we have found that the representation converges after several iterations.

Now that we have seen a general outline of our vectorizer, let us explore the details.

3 Basic Vectorization Method

The basic method for our vectorizer breaks down into two main parts, the off-line preparation of the example textures $\mathbf{t}_{p_j}$, and the on-line vectorization procedure applied to a new input image.

3.1 Off-line preparation of examples

The basic assumption made in modeling vectorized texture is that the space of face textures is linearly spanned by a set of geometrically normalized example face textures. Thus, in constructing a vectorizer we must first collect a group of representative faces that will define face space, the space of the textural component in our representation. Before using the example faces in the vectorizer, they are geometrically normalized to align facial features, and the grey levels are processed using principal components or the pseudoinverse to optimize run-time textural processing.

3.1.1 Geometric normalization

To geometrically normalize an example face, we apply a local deformation to the image to warp the face shape into a standard geometry. This local deformation requires both the shape of the example face as well as some definition of the standard shape. Thus, our off-line normalization procedure needs the face shape component for our example faces, something we provide manually. These manual correspondences are averaged to define the standard shape. Finally, a 2D warping operation is applied to do the normalization. We now go over these steps in more detail.

First, to define the shape of the example faces, a set of line segment features are positioned manually for each. The features, shown in Fig. 1, follow Beier and Neely's [5] manual correspondence technique for morphing face images. Pairing up image feature points into line segments gives one a natural control over local scale and rotation

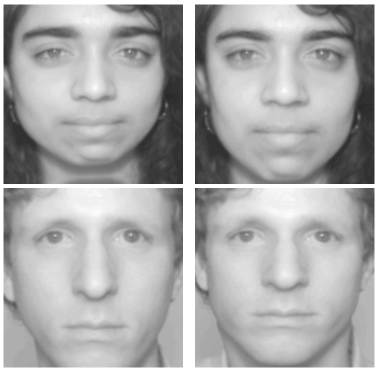

Figure 5: Examples of off-line geometrical normalization of example images. Texture for the normalized images is sampled from the original images – that is why the chin is generated for the second example.

in the eventual deformation to standard shape, as we will explain later when discussing the deformation technique.

Next, we average the line segments over the example images to define the standard face shape (see Fig. 2). We don't have to use averaging – since we are creating a definition, we could have just chosen a particular example face. However, averaging shape should minimize the total amount of distortion required in the next step of geometrical normalization.

Finally, images are geometrically normalized using the local deformation technique of Beier and Neely [5]. This deformation technique is driven by the pairing of line segments in the example image with line segments in the standard shape. Consider a single pairing of line segments, one segment from the example image l_{ex} and one from the standard shape l_{std}. This line segment pair essentially sets up a local transform from the region surrounding l_{ex} to the region surrounding l_{std}. The local transform resembles a similarity transform except that there is no scaling perpendicular to the segment, just scaling along it. The local transforms are computed for each segment pair, and the overall warping is taken as weighted average. Some examples of images before and after normalization are shown in Fig. 5.

3.1.2 Texture processing

Now that the example faces have been normalized for shape, they can be used for texture modeling. Given a new input i_a, the texture analysis step tries to approximate the input texture $\mathbf{t}_a$ as a linear combination of the example textures. Of course, given a linear subspace such as our face space, one can choose among different sets of basis vectors that will span the same subspace. One popular method for choosing the basis set, the eigenimage approach, applies principal components analysis to the example set. Another potential basis set is simply the original set of images themselves. We now discuss the off-line texture processing required for the two basis sets of principal components and the original images.

Principal components analysis is a classical technique for reducing the dimensionality of a cluster of data points, where the data are assumed to be distributed in an ellipsoid pattern about a cluster center. If there is correlation in the data among the coordinate axes, then one can project the data points to a lower dimensional subspace without losing information. This corresponds to an ellipsoid with interesting variation along a number of directions that is less than the dimensionality of the data points. Principal components analysis finds the lower dimensional subspace inherent in the data points. It works by finding a set of directions $\mathbf{e}_i$ such that the variance in the data points is highest when projected onto those directions. These $\mathbf{e}_i$ directions are computed by finding the eigenvectors of the of the covariance matrix of the data points.

In our ellipsoid of n geometrically normalized textures $\mathbf{t}_{p_j}$, let $\mathbf{t}'_{p_j}$ be the set of textures with the mean $\mathbf{t}_{mean}$ subtracted off

$$\begin{aligned} \mathbf{t}_{mean} &= \frac{1}{n}\sum_{j=1}^{n} \mathbf{t}_{p_j} \\ \mathbf{t}'_{p_j} &= \mathbf{t}_{p_j} - \mathbf{t}_{mean}, \qquad 1 \le j \le n. \end{aligned}$$

If we let T be a matrix where the jth column is $\mathbf{t}'_{p_j}$

$$T = \left[\mathbf{t}'_{p_1}\ \mathbf{t}'_{p_2}\ \cdots\ \mathbf{t}'_{p_n}\right],$$

then the covariance matrix is defined as

$$\Sigma = TT^t.$$

Notice that T is a $m \times n$ matrix, where m is the number of pixels in vectorized texture vectors. Due to our pixelwise representation for shape, $m \gg n$ and thus Σ, which is a $m \times m$ matrix, is quite large and may be intractable for eigenanalysis. Fortunately, one can solve the smaller eigenvector problem for the $n \times n$ matrix T^tT. This is possible because an eigenvector $\mathbf{e}_i$ of T^tT

$$T^tT\ \mathbf{e}_i = \lambda_i \mathbf{e}_i$$

corresponds to an eigenvector $T\mathbf{e}_i$ of Σ. This can be seen by multiplying both sides of the above equation by matrix T

$$(TT^t)\, T\mathbf{e}_i = \lambda_i T\mathbf{e}_i.$$

Since the eigenvectors (or eigenimages) $\mathbf{e}_i$ with the larger eigenvalues λ_i explain the most variance in the example set, only a fraction of the eigenimages need to be retained for the basis set. In our implementation, we chose to use roughly half the eigenimages. Fig. 6 shows the mean face and the first 6 eigenimages from a principal components analysis applied to a group of 55 people.

Since the eigenimages are orthogonal (and can easily be normalized to be made orthonormal), analysis and reconstruction of new image textures during vectorization can be easily performed. Say that we retain N eigenimages, and let $\mathbf{t}_a$ be a geometrically normalized texture to analyze. Then the run-time vectorization procedure projects $\mathbf{t}_a$ onto the $\mathbf{e}_i$

$$\beta_i = \mathbf{e}_i \cdot (\mathbf{t}_a - \mathbf{t}_{mean}) \tag{2}$$

and can reconstruct $\mathbf{t}_a$, yielding $\widehat{\mathbf{t}}_a$

$$\widehat{\mathbf{t}}_a = \mathbf{t}_{mean} + \textstyle\sum_{i=1}^{N} \beta_i \mathbf{e}_i. \tag{3}$$

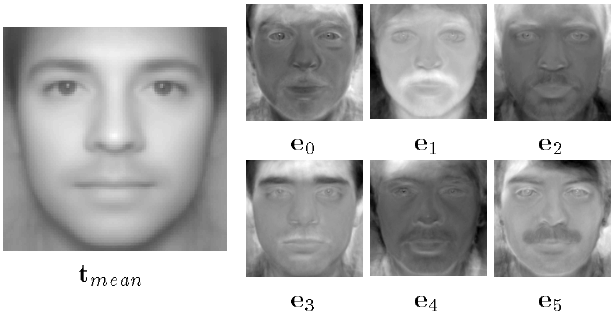

Figure 6: Mean image and eigenimages from applying principal components analysis to the geometrically normalized examples.

Another potential basis set is the original example textures themselves. That is, we approximate $\mathbf{t}_a$ by a linear combination of the n original image textures $\mathbf{t}_{p_i}$

$$\widehat{\mathbf{t}}_a = \textstyle\sum_{i=1}^{n} \beta_i \mathbf{t}_{p_i}. \tag{4}$$

While we do not need to solve this equation until on-line vectorization, previewing the solution will elucidate what needs to be done for off-line processing. Write equation (4) in matrix form

$$\widehat{\mathbf{t}}_a = T\ \beta, \tag{5}$$

where $\widehat{\mathbf{t}}_a$ is written as a column vector, T is a matrix where the ith column is $\mathbf{t}_{p_i}$, and β is a column vector of the β_i's. Solving this with linear least squares yields

$$\begin{aligned} \beta &= T^{\dagger}\ \mathbf{t}_a && (6) \\ &= (T^tT)^{-1}T^t\ \mathbf{t}_a && (7) \end{aligned}$$

where $T^{\dagger} = (T^tT)^{-1}T^t$ is the pseudoinverse of T. The pseudoinverse can be computed off-line since it depends only on the example textures $\mathbf{t}_{p_i}$. Thus, run-time vectorization performs texture analysis with the columns of $T^{\dagger}$ (equation (6)) and reconstruction with the columns of T (equation (5)). Fig. 7 shows some example images processed by the pseudoinverse where n was 40.

Note that for both basis sets, the linear coefficients are computed using a simple projection operation. Coding-wise at run-time, the only difference is whether one subtracts off the mean image $\mathbf{t}_{mean}$. In practice though, the eigenimage approach will require fewer projections since not all eigenimages are retained. Also, the orthogonality of the eigenimages may produce a more stable set of linear coefficients – consider what happens for the pseudoinverse approach when two example images are similar in texture. Yet another potential basis set, one that has the advantage of orthogonality, would be the result of applying Gram-Schmidt orthonormalization to the example set.

Most of our vectorization experiments have been with the eigenimage basis, so the notation in the next section uses this basis set.

3.2 Run-time vectorization

In this section we go over the details of the vectorization procedure. The inputs to the vectorizer are an image i_a

Figure 7: Example textures processed by the pseudoinverse $T^\dagger = (T^tT)^{-1}T^t$. When using the original set of image textures as a basis, texture analysis is performed by projection onto these images.

to vectorize and a texture model consisting of N eigenimages $\mathbf{e}_i$ and mean image $\mathbf{t}_{mean}$. In addition, the vectorizer takes as input a planar transform P that selects the face region from the image i_a and normalizes it for the effects of scale and image-plane rotation. The planar transform P can be a rough estimate from a coarse scale analysis. Since the faces in our test images were taken against a solid background, face detection is relatively easy and can be handled simply by correlating with a couple face templates. The vectorization procedure refines the estimate P, so the final outputs of the procedure are the vectorized shape $\mathbf{y}^{std}_{a-std}$, a set of β_i coefficients for computing $\widehat{\mathbf{t}}_a$, and a refined estimate of P.

As mentioned previously, the interconnectedness of the shape and texture steps makes the iteration converge. Fig. 8 depicts the convergence of the vectorization procedure from the perspective of texture. There are three sets of face images in the figure, sets of (1) all face images, (2) geometrically normalized face textures, and (3) the space of our texture model. The difference between the texture model space and the set of geometrically normalized faces depends on the prototype set of n example faces. The larger and more varied this set becomes, the smaller the difference becomes between sets (2) and (3). Here we assume that the texture model is not perfect, so the true $\mathbf{t}_a$ is slightly outside the texture model space.

The goal of the iteration is to make estimates of $\mathbf{t}_a$ and $\widehat{\mathbf{t}}_a$ converge to the true $\mathbf{t}_a$. The path for $\mathbf{t}_a$, the geometrically normalized version of i_a, is shown by the curve from i_a to the final $\mathbf{t}_a$. The path for $\widehat{\mathbf{t}}_a$ is shown by the curve from initial $\widehat{\mathbf{t}}_a$ to final $\widehat{\mathbf{t}}_a$. The texture and shape steps are depicted by the arrows jumping between the curves. The texture step, using the latest estimate of shape to produce $\mathbf{t}_a$, projects $\mathbf{t}_a$ into the texture model space. The shape step uses the latest $\widehat{\mathbf{t}}_a$ to find a new set of correspondences, thus updating shape and hence $\mathbf{t}_a$. As one moves along the $\mathbf{t}_a$ curve, one is getting better estimates of shape. As one moves along the $\widehat{\mathbf{t}}_a$ curve, the β_i coefficients in the texture model improve. Since the true $\mathbf{t}_a$ lies outside the texture model space, the iteration stops at final $\widehat{\mathbf{t}}_a$. This error can be made smaller by increasing the number of prototypes for the texture model.

We now look at one iteration step in detail.

3.2.1 One iteration

In examining one iteration of the texture and shape steps, we assume that the previous iteration has provided an estimate for $\mathbf{y}^{std}_{a-std}$ and the β_i coefficients. For the first iteration, an initial condition of $\mathbf{y}^{std}_{a-std} = \vec{\mathbf{0}}$ is used. No initial condition is needed for texture since the iteration starts with the texture step.

In the **texture step**, first the input image i_a is geometrically normalized using the shape estimate $\mathbf{y}^{std}_{a-std}$, producing $\mathbf{t}_a$

$$\mathbf{t}_a(\mathbf{x}) = i_a(\mathbf{x} + \mathbf{y}^{std}_{a-std}(\mathbf{x})), \tag{8}$$

where $\mathbf{x} = (x, y)$ is a pixel location in the standard shape. This is implemented as a backwards warp using the flow vectors pointing from the standard shape to the input. Bilinear interpolation is used to sample i_a at non-integral (x, y) locations. Next, $\mathbf{t}_a$ is projected onto the eigenimages $\mathbf{e}_i$ using equation (2) to update the linear coefficients β_i. These updated coefficients should enable the shape computation to synthesize an approximation $\widehat{\mathbf{t}}_a$ that is closer to the true $\mathbf{t}_a$.

In the **shape step**, first a reference image $\widehat{\mathbf{t}}_a$ is synthesized from the texture coefficients using equation (3). Since the reference image reconstructs the texture of the input, it should be well suited for finding shape correspondences. Next, optical flow is computed between $\widehat{\mathbf{t}}_a$, which is geometrically normalized, and i_a, which updates the pixelwise correspondences $\mathbf{y}^{std}_{a-std}$. For optical flow, we used the gradient-based hierarchical scheme of Bergen and Adelson [7], Bergen and Hingorani [9], and Bergen, *et al.* [8]. The new correspondences should provide better geometrical normalization in the next texture step.

Overall, iterating these steps until the representation stabilizes is equivalent to iteratively solving for the $\mathbf{y}^{std}_{a-std}$ and β_i which best satisfy

$$\mathbf{t}_a = \widehat{\mathbf{t}}_a,$$

or

$$i_a(\mathbf{x} + \mathbf{y}^{std}_{a-std}(\mathbf{x})) = \mathbf{t}_{mean} + \textstyle\sum_{i=1}^n \beta_i \mathbf{e}_i.$$

3.2.2 Adding a global transform

We introduce a planar transform P to select the image region containing the face and to normalize the face for the effects of scale and image-plane rotation. Let i'_a be the input image i_a resampled under the planar transform P

$$i'_a(\mathbf{x}) = i_a(P(\mathbf{x})). \tag{9}$$

It is this resampled image i'_a that will be geometrically normalized in the texture step and used for optical flow in the shape step.

Besides selecting the face, the transform P will also be used for selecting subimages around individual features such as the eyes, nose, and mouth. As will be explained

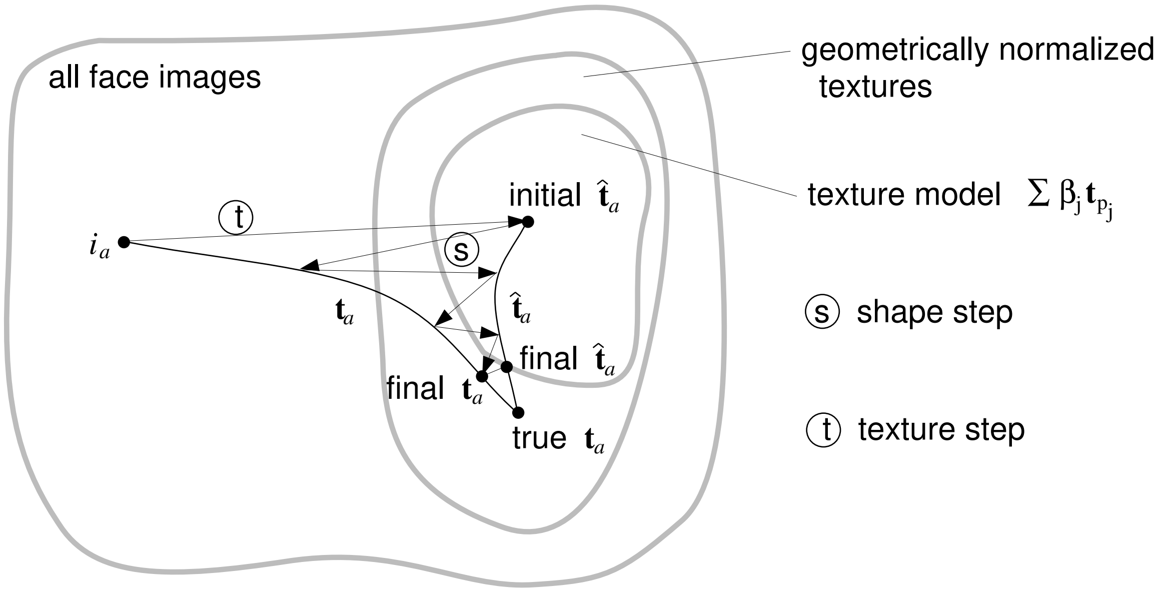

Figure 8: Convergence of the vectorization procedure with regards to texture. The texture and shape steps try to make $\widehat{\mathbf{t}}_a$ and $\mathbf{t}_a$ converge to the true $\mathbf{t}_a$.

in the next section on our hierarchical implementation, the vectorization procedure is applied in a coarse-to-fine strategy on a pyramid structure. Full face templates are vectorized at the coarser scales and individual feature templates are vectorized at the finer scales.

Transform P will be a similarity transform

$$P(\mathbf{x}) = s \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \mathbf{x} + \begin{pmatrix} t_x \\ t_y \end{pmatrix},$$

where the scale s, image-plane rotation θ, and 2D translation (t_x, t_y) are determined in one of two ways, depending on the region being vectorized.

1. *Two point correspondences.* Define anchor points $\mathbf{q}_{std,1}$ and $\mathbf{q}_{std,2}$ in standard shape, which can be done manually in off-line processing. Let $\mathbf{q}_{a,1}$ and $\mathbf{q}_{a,2}$ be estimates of the anchor point locations in the image i_a, estimates which need to be performed on-line. The similarity transform parameters are then determined such that

$$P(\mathbf{q}_{std,1}) = \mathbf{q}_{a,1}, \qquad P(\mathbf{q}_{std,2}) = \mathbf{q}_{a,2}. \qquad (10)$$

   This uses the full flexibility of the similarity transform and is used when the image region being vectorized contains two reliable feature points such as the eyes.

2. *Fixed s, θ, and one point correspondence.* In this case there is only one anchor point $\mathbf{q}_{std,1}$, and one solves for t_x and t_y such that

$$P(\mathbf{q}_{std,1}) = \mathbf{q}_{a,1}. \qquad (11)$$

   This is useful for vectorizing templates with less reliable features such as the nose and mouth. For these templates the eyes are vectorized first and used to fix the scale and rotation for the nose and mouth.

While the vectorizer assumes that a face finder has provided an initial estimate for P, we would like the vectorizer to be insensitive to a coarse or noisy estimate and to improve the estimate of P during vectorization. The similarity transform P can be updated during the iteration when our estimates change for the positions of the anchor points $\mathbf{q}_{a,i}$. This can be determined after the shape step computes a new estimate of the shape $\mathbf{y}^{std}_{a-std}$. We can tell that an anchor point estimate is off when there is nonzero flow at the anchor point

$$\|\mathbf{y}^{std}_{a-std}(\mathbf{q}_{std,i})\| > \text{threshold}.$$

The correspondences can be used to update the anchor point estimate

$$\mathbf{q}_{a,i} = P(\mathbf{q}_{std,i} + \mathbf{y}^{std}_{a-std}(\mathbf{q}_{std,i})).$$

Next, P can be updated using the new anchor point locations using equation (10) or (11) and i_a can be resampled again using equation (9) to produce a new i'_a.

3.2.3 Entire procedure

The basic vectorization procedure is now summarized. Lines 2(a) and (b) are the texture step, lines 2(c) and (d) are the shape step, and line 2(e) updates the similarity transform P.

procedure vectorize

1. initialization
   (a) Estimate P using a face detector. For example, a correlational face finder using averaged face templates can be used to estimate the translational component of P.
   (b) Resample i_a using the similarity transform P, producing i'_a (equation (9)).

(c) $\mathbf{y}^{std}_{a-std} = \vec{\mathbf{0}}$.

2. iteration: solve for $\mathbf{y}^{std}_{a-std}$, β_i, and P by iterating the following steps until the β_i stop changing.

   (a) Geometrically normalize i'_a using $\mathbf{y}^{std}_{a-std}$, producing $\mathbf{t}_a$

   $$\mathbf{t}_a(\mathbf{x}) = i'_a(\mathbf{x} + \mathbf{y}^{std}_{a-std}(\mathbf{x})).$$

   (b) Project $\mathbf{t}_a$ onto example set $\mathbf{e}_i$, computing the linear coefficients β_i

   $$\beta_i = \mathbf{e}_i \cdot (\mathbf{t}_a - \mathbf{t}_{mean}), \quad 1 \leq i \leq n.$$

   (c) Compute reference image $\widehat{\mathbf{t}}_a$ for correspondence by reconstructing the geometrically normalized input

   $$\widehat{\mathbf{t}}_a = \mathbf{t}_{mean} + \textstyle\sum_{i=1}^{n} \beta_i \mathbf{e}_i.$$

   (d) Compute the shape component using optical flow

   $$\mathbf{y}^{std}_{a-std} = \text{optical-flow}(i'_a, \widehat{\mathbf{t}}_a).$$

   (e) If the anchor points are misaligned, as indicated by optical flow, then:

      i. Update P with new anchor points.

      ii. Resample i_a using the similarity transform P, producing i'_a (eqn (9)).

      iii. $\mathbf{y}^{std}_{a-std} = \text{optical-flow}(i'_a, \widehat{\mathbf{t}}_a)$.

Fig. 9 shows snapshot images of i'_a, $\mathbf{t}_a$, and $\widehat{\mathbf{t}}_a$ during each iteration of an example vectorization. The iteration number is shown in the left column, and the starting input is shown in the upper left. We deliberately provided a poor initial alignment for the iteration to demonstrate the procedure's ability to estimate the similarity transform P. As the iteration proceeds, notice how (1) improvements in P lead to a better global alignment in i'_a, (2) the geometrically normalized image $\mathbf{t}_a$ improves, and (3) the image $\widehat{\mathbf{t}}_a$ becomes a more faithful reproduction of the input. The additional row for i'_a is given because when step 2(e) is executed in the last iteration, i'_a is updated.

3.3 Pose dependence from the example set

The example images we have used in the vectorizer so far have been from a frontal pose. What about other poses, poses involving rotations out of the image plane?

Because we are being careful about geometry and correspondence, the example views used to construct the vectorizer must be taken from the same out-of-plane image rotation. The resulting vectorizer will be tuned to that pose, and performance is expected to drop as an input view deviates from that pose. The only thing that makes the vectorizer pose-dependent, however, is the set of example views used to construct face space. The iteration step is general and should work for a variety of poses. Thus, even though we have chosen a frontal view as an example case, a vectorizer tuned for a different pose can be constructed simply by using example views from that pose.

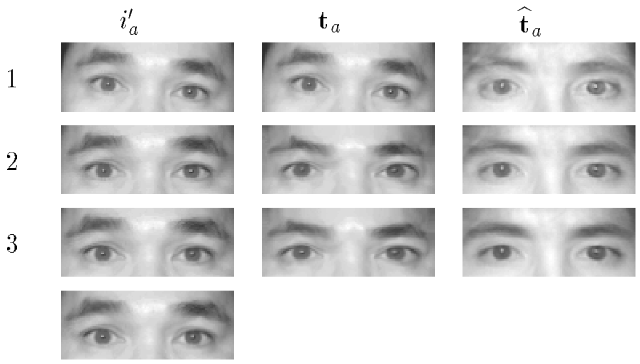

Figure 9: Snapshot images of i'_a, $\mathbf{t}_a$, and $\widehat{\mathbf{t}}_a$ during the three iterations of an example vectorization. See text for details.

In section 5.1 on applying the vectorizer to feature detection, we demonstrate two vectorizers, one tuned for a frontal pose, and one for an off-frontal pose. Later, in section 6.3, we suggest a multiple-pose vectorizer that connects different pose-specific vectorizers through interpolation.

4 Hierarchical implementation

For optimization purposes, the vectorization procedure is implemented using a coarse-to-fine strategy. Given an input image to vectorize, first the Gaussian pyramid (Burt and Adelson [14]) is computed to provide a multiresolution representation over 4 scales, the original image plus 3 reductions by 2. A face finder is then run over the coarsest level to provide an initial estimate for the similarity transform P. Next, the vectorizer is run at each pyramid level, working from the coarser to finer levels. As processing moves from a coarser level to a finer one, the coarse shape correspondences are used to initialize the similarity transform P for the vectorizer at the finer level.

4.1 Face finding at coarse resolution

For our test images, face detection is not a major problem since the subjects are shot against a uniform background. For the more general case of cluttered backgrounds, see the face detection work of Reisfeld and Yeshurun [32], Ben-Arie and Rao [6], Sung and Poggio [35], Sinha [34], and Moghaddam and Pentland [25]. For our test images, we found that normalized correlation using two face templates works well. The normalized correlation metric is

$$r = \frac{<\mathbf{TI}> - <\mathbf{T}><\mathbf{I}>}{\sigma(\mathbf{T})\sigma(\mathbf{I})},$$

where $\mathbf{T}$ is the template, $\mathbf{I}$ is the subportion of image being matched against, $<\mathbf{TI}>$ is normal correlation, $<>$ is the mean operator, and $\sigma()$ measures standard deviation. The templates are formed by averaging face grey levels over two populations, an average of all examples plus an average over people with beards. Before averaging, example face images are first warped to standard

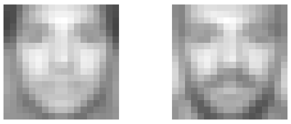

Figure 10: Face finding templates are grey level averages using two populations, all examples (left) plus people with beards (right).

shape. Our two face templates for a frontal pose are shown in Fig. 10. To provide some invariance to scale, regions with high correlation response to these templates are examined with secondary correlations where the scale parameter is both increased and decreased by 20%. The location/scale of correlation matches above a certain threshold are reported to the vectorizer.

4.2 Multiple templates at high resolution

When processing the different pyramid levels, we use a whole face template at the two coarser resolutions and templates around the eyes, nose, and mouth for the two finer resolutions. This template decomposition across scales is similar to Burt's pattern tree approach [13] for template matching on a pyramid representation. At a coarse scale, faces are small, so full face templates are needed to provide enough spatial support for texture analysis. At a finer scale, however, individual features – eyes, noses – cover enough area to provide spatial support for analysis, giving us the option to perform separate vectorizations. The advantage of decoupling the analysis of the eyes, nose, and mouth is that it should improve generalization to new faces not in the original example set. For example, if the eyes of a new face use one set of linear texture coefficients and the nose uses another, separate vectorization for the eyes and nose provides the extra flexibility we need. However, if new inputs always come from people in the original example set, then this extra flexibility is not required and keeping to whole-face templates should be a helpful constraint.

When vectorizing separate eyes, nose, and mouth templates at the finer two resolutions, the template of the eyes has a special status for determining the scale and image-plane rotation of the face. The eyes template is vectorized first, using 2 iris features as anchor points for the similarity transform P. Thus, the eyes vectorization estimates a normalizing similarity transform for the face. The scale and rotation parameters are then fixed for the nose and mouth vectorizations. Only one anchor point is used for the nose and mouth, allowing only the translation in P to change.

4.3 Example results

For the example case in Fig. 11, correspondences from the shape component are plotted over the four levels of the Gaussian pyramid. These segment features are generated by mapping the averaged line segments from Fig. 2 to the input image. To get a sense of the final shape/texture representation computed at the highest resolution, Fig. 12 displays the final output for the Fig. 11 example. For the eyes, nose and mouth templates, we show i'_a, the geometrically normalized templates $\mathbf{t}_a$, and the reconstruction of those templates $\widehat{\mathbf{t}}_a$ using the linear texture coefficients. No images of this person were used among the examples used to create the eigenspaces.

We have implemented the hierarchical vectorizer in C on an SGI Indy R4600 based machine. Once the example images are loaded, multilevel processing takes just a few seconds to execute.

Experimental results presented in the next section on applications will provide a more thorough analysis of the vectorizer.

5 Applications

Once the vectorized representation has been computed, how can one use it? The linear texture coefficients can be used as a low-dimensional feature vector for face recognition, which is the familiar eigenimage approach to face recognition [37][2][26]. Our application of the vectorizer, however, has focused on using the correspondences in the shape component. In this section we describe experimental results from applying these correspondences to two problems, locating facial features and the registration of two arbitrary faces.

5.1 Feature finding

After vectorizing an input image i_a, pixelwise correspondence in the shape component $\mathbf{y}^{std}_{a-std}$ provides a dense mapping from the standard shape to the image i_a. Even though this dense mapping does more than locate just a sparse set of features, we can sample the mapping to locate a discrete set of feature points in i_a. To accomplish this, first, during off-line example preparation, the feature points of interest are located manually with respect to the standard shape. Then after the run-time vectorization of i_a, the feature points can be located in i_a by following the pixelwise correspondences and then mapping under the similarity transform P. For a feature point $\mathbf{q}_{std}$ in standard shape, its corresponding location in i_a is

$$P(\mathbf{q}_{std} + \mathbf{y}^{std}_{a-std}(\mathbf{q}_{std})).$$

For example, the line segment features of Fig. 2 can be mapped to the input by mapping each endpoint, as shown for the test images in Fig. 13.

In order to evaluate these segment features located by the vectorizer, two vectorizers, one tuned for a frontal pose and one for a slightly rotated pose, were each tested on separate groups of 62 images. The test set consists of 62 people, 2 views per person – a frontal and slightly rotated pose – yielding a combined test set of 124 images. Example results from the rotated view vectorizer are shown in Fig. 14. Because the same views were used as example views to construct the vectorizers, a leave-6-out cross validation procedure was used to generate statistics. That is, the original group of 62 images from a given pose were divided into 11 randomly chosen groups (10 of 6 people, 1 of the remaining 2 people). Each group

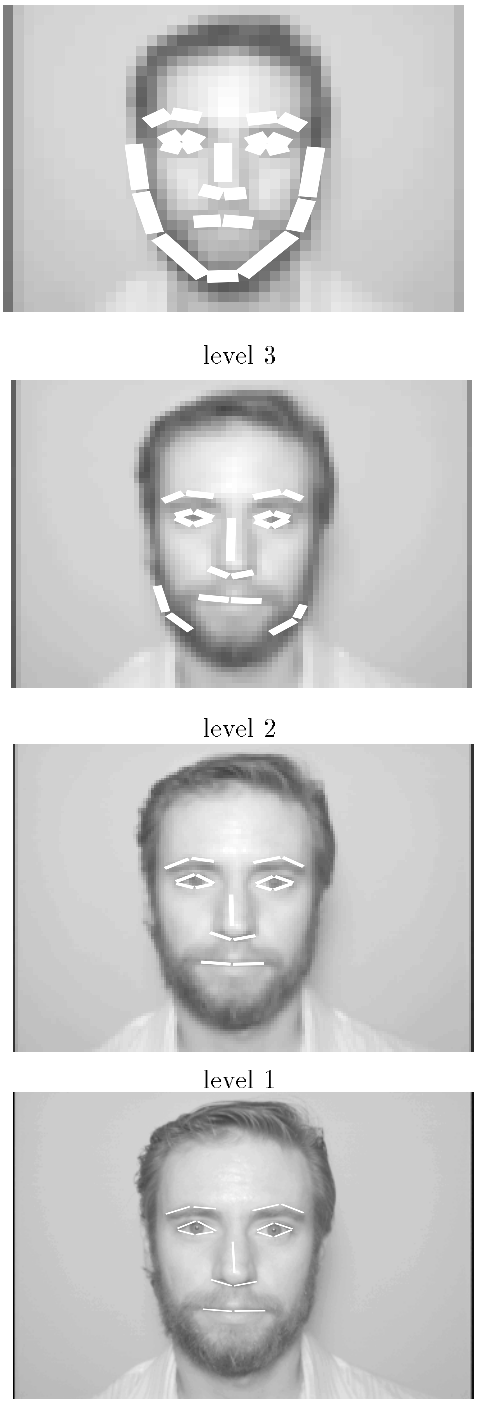

Figure 11: Evolution of the shape component during coarse-to-fine processing. The shape component is displayed through segment features which are generated by mapping the averaged line segments from Fig. 2 to the input image.

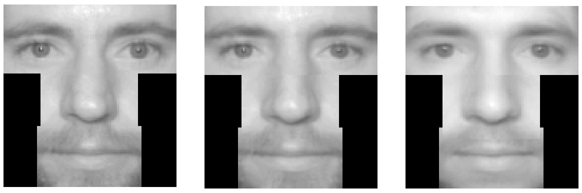

Figure 12: Final vectorization at the original image resolution.

Figure 13: Example features located by sampling the dense set of shape correspondences $\mathbf{y}^{std}_{a-std}$ found by the vectorizer.

of images is tested using a different vectorizer; the vectorizer for group G is constructed from an example set consisting of the original images minus the set G. This allows us to separate the people used as examples from those in the test set.

Qualitatively, the results were very good, with only one mouth feature being completely missed by the vectorizer (it was placed between the mouth and nose). To quantitatively evaluate the features, we compared the computed segment locations against manually located "ground truth" segments, the same segments used for off-line geometrical normalization. To report statistics by feature, the segments in Fig. 2 are grouped into 6 features: left eye (c_3, c_4, c_5, c_6), right eye (c_9, c_{10}, c_{11}, c_{12}), left eyebrow (c_1, c_2), right eyebrow (c_7, c_8), nose (n_1, n_2, n_3), and mouth (m_1, m_2).

Two different metrics were used to evaluate how close a computed segment came to its corresponding ground truth segment. Segments in the more richly textured areas (e.g. eye segments) have local grey level structure at both endpoints, so we expect both endpoints to be ac-

Figure 14: Example features located by the vectorizer.

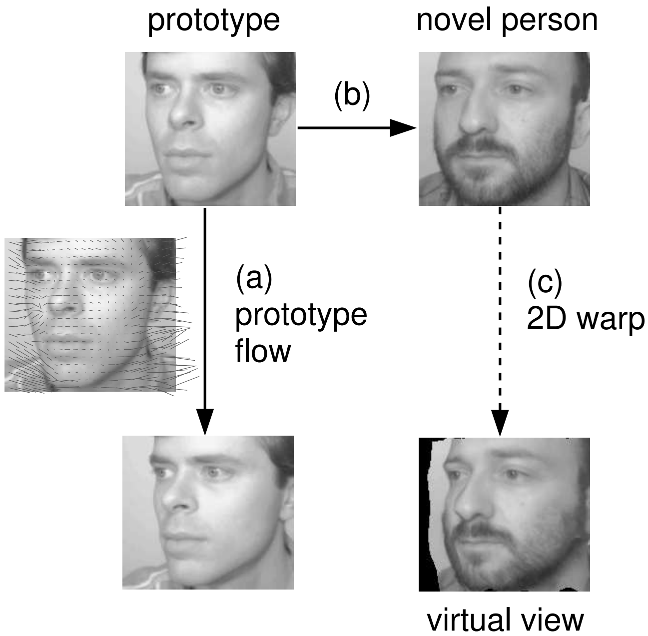

Figure 15: In parallel deformation, (a) a 2D deformation representing a transformation is measured by finding correspondence among prototype images. In this example, the transformation is rotation and optical flow was used to find a dense set of correspondences. Next, in (b), the flow is mapped onto the novel face, and (c) the novel face is 2D warped to a "virtual" view. Figure from [11].

curately placed. Thus, the "point" metric measures the two distances between corresponding segment endpoints. On the other hand, some segments are more edge-like, such as eyebrows and mouths. For the "edge" metric we measure the angle between segments and the perpendicular distance from the midpoint of the ground truth segment to the computed segment.

Next, the distances between the manual and computed segments were thresholded to evaluate the closeness of fit. A feature will be considered properly detected when *all* of its constituent segments are within threshold. Using a distance threshold of 10% of the interocular distance and an angle threshold of 20°, we compute detection rates and average distances between manual and computed segments (Table 1). The eyebrow and nose errors are more of a misalignment of a couple points rather than a complete miss (the mouth error was a complete miss).

In the next section we consider another application of the shape component computed by the vectorizer.

5.2 Registration of two arbitrary faces

Suppose that we have only one view of an individual's face and that we would like to synthesize other views, perhaps rotated views or views with different expressions. These new "virtual" views could be used, for example, to create an animation of the individual's face from just one view. For the task of face recognition, virtual views could be used as multiple example views in a view-based recognizer. In this section, we discuss how the shape component from the vectorizer can be used to synthesize virtual views. In addition, these virtual views are then evaluated by plugging them into a view-based, pose-invariant face recognizer.

To synthesize virtual views, we need to have prior knowledge of a facial transformation such as head rotation or expression change. A standard approach used in the computer graphics and computer vision communities for representing this prior knowledge is to use a 3D model of the face (Akimoto, Suennaga, and Wallace[3], Waters and Terzopoulos[36][39], Aizawa, Harashima, and Saito[1], Essa and Pentland [20]). After the single available 2D image is texture mapped onto a 3D polygonal or multilayer mesh model of the face, rotated views can be synthesized by rotating the 3D model and rendering. In addition, facial expressions have been modeled [36][39][20] by embedding muscle forces that deform the 3D model in a way that mimics human facial muscles. Mapping image data onto the 3D model is typically solved by locating corresponding points on both the 3D model and the image or by simultaneously acquiring both the 3D depth and image data using the Cyberware scanner.

We have investigated an alternative approach that uses example 2D views of prototype faces as a substitute for 3D models (Poggio and Vetter [30], Poggio and Brunelli [29], Beymer and Poggio [11]). In *parallel deformation*, one of the example-based techniques discussed in Beymer and Poggio [11], prior knowledge of a facial transformation such as a rotation or change in expression is extracted from views of a prototype face undergoing the transformation. Shown in Fig. 15, first a 2D deformation representing the transformation is measured

feature	detection rate	average distances		
		point metric	edge metric	
		endpt. dist. (pixels)	angle (degrees)	perpend. dist. (pixels)
left eye	100% (124/124)	1.24	-	-
right eye	100% (124/124)	1.23	-	-
left eyebrow	97% (121/124)	-	5.1°	1.06
right eyebrow	96% (119/124)	-	4.8°	1.06
nose	99% (123/124)	1.45	3.2°	0.66
mouth	99% (123/124)	-	2.2°	0.53

Table 1: Detection rates and average distances between computed and "ground truth" segments. Qualitatively, the eyebrow and nose errors were misalignments, while the mouth error did involve a complete miss.

real views virtual views

Figure 16: Example pairs of real and virtual views.

by finding correspondence between the prototype face images. We use the same gradient-based optical flow algorithm [9] used in the vectorizer to find a dense set of pixelwise correspondences. Next, the prototype flow is mapped onto the "novel" face, the individual for which we wish to generate virtual views. This step requires "interperson" correspondence between the prototype and novel faces. Finally, the prototype flow, now mapped onto the novel face, can be used to 2D warp the novel face to produce the virtual view.

The difficult part of parallel deformation is automatically finding a set of feature correspondences between the prototype and novel faces. We have used the vectorizer to automatically locate the set of facial features shown in Fig. 14 in both the prototype and novel faces. From this sparse set of correspondences, the interpolation technique from Beier and Neely [5] is used to generate a dense, pixelwise mapping between the two faces. We then used the dense set of correspondences to map rotation deformations from a single prototype to a group of 61 other faces for generating virtual views. Fig. 16 shows some example pairs of real and virtual views.

To evaluate these virtual views, they were used as example views in a view-based, pose-invariant face recognizer (see [11] for details). The problem is this: given one real view of each person, can we recognize the person under a variety of poses? Virtual views were used to generate a set of rotated example views to augment the single real view. Using a simple view-based approach that represents faces with templates of the eyes, nose, and mouth, we were able to get a recognition rate of 85% on a test set of 620 images (62 people, 10 views per person). To put this number in context, consider the recognition results from a "base" case of two views per person (the single real view plus its mirror reflection) and a "best" case of 15 real views per person. When tested on the same test set, we obtained recognition rates of 70% for the two views case and 98% for the 15 views case. Thus, adding virtual views to the recognizer increases the recognition by 15%, and the performance of virtual views is about midway between the base and best case scenarios.

6 Future work

In this section, first we discuss some shorter-term work for the existing vectorizer. This is followed by longer-term ideas for extending the vectorizer to use parameterized shape models and to handle multiple poses.

6.1 Existing vectorizer

So far the vectorizer has been tested on face images shot against a solid background. It would be nice to demonstrate the vectorizer working in cluttered environments. To accomplish this, both the face detection and vectorizer should be made more robust to the presense of false positive matches. To improve face detection, we would probably incorporate the learning approaches of Sung and Poggio [35] or Moghaddam and Pentland [25]. Both of these techniques model the space of grey level face images using principal components analysis. To judge the "faceness" of a image, they use a distance metric that includes two terms, "distance from face space" (see Turk and Pentland [37])

$$\|\mathbf{t}_a - \widehat{\mathbf{t}}_a\|$$

and the Mahalanobis distance

$$\sum_{i=1}^{N} \frac{\beta_i^2}{\lambda_i},$$

where the β_i are the eigenspace projection coefficients and λ_i are the eigenvalues from principal component analysis. This distance metric could be added to the vectorizer as a threshold test after the iteration step has converged.

Our current coarse-to-fine implementation does not exploit potential constraints that could be passed from the coarser to finer scales. The only information currently passed from a coarse level to the next finer level are feature locations used to initialize the similarity transform P. This could be expanded to help initialize the shape and texture components at the finer level as well.

6.2 Parameterized shape model

In the current vectorizer, shape is measured in a "data-driven" manner using optical flow. However, we can explicitly model shape by taking a linear combination of example shapes

$$\mathbf{y}_{a-std}^{std} = \sum_{i=1}^{n} \alpha_i \mathbf{y}_{p_i-std}^{std},$$

where the shape of the ith example image, $\mathbf{y}_{p_i-std}^{std}$, is the 2D warping used to geometrically normalize the image in the off-line preparation step. This technique for modeling shape is similar to the work of Cootes, *et al.* [17], Blake and Isard [12], Baumberg and Hogg [4], and Jones and Poggio [21]. The new shape step would, given i'_a and reference $\widehat{\mathbf{t}}_a$, try to find a set of coefficients α_i that minimizes the squared error of the approximation

$$i'_a(\mathbf{x} + \sum_{i=1}^{n} \alpha_i \mathbf{y}_{p_i-std}^{std}(\mathbf{x})) = \widehat{\mathbf{t}}_a.$$

This involves replacing the optical flow calculation with a model-based matching procedure; one can think of it as a parameterized "optical flow" calculation that computes a single set of linear coefficients instead of a flow vector at each point. One advantage of modeling shape is the extra constraint it provides, as some "illegal" warpings cannot even be represented. Additionally, compared to the raw flow, the linear shape coefficients should be more amenable for shape analysis tasks like expression analysis or face recognition using shape.

Given this new model for shape in the vectorizer, the set of α shape coefficients and β texture coefficients could be used as a low-dimensional representation for faces. An obvious application of this would be face recognition. Even without the modified vectorizer and the α coefficients, the β coefficients alone could be evaluated as a representation for a face recognizer.

6.3 Multiple poses

The straightforward way to handle different out-of-plane image rotations with the vectorizer is simply to use several vectorizers, each tuned to a different pose. However, if we provide pixelwise correspondence between the standard shapes of the different vectorizers, their operations can be linked together through image interpolation. The main idea is to interpolate among the $\widehat{\mathbf{t}}_a$ images of the different vectorizers to produce a new image that reconstructs both the grey levels *and the pose* of the input image (see Beymer, Shashua and Poggio [10] for examples of interpolation across different poses). Correspondence is then found between the input and this new interpolated image using optical flow. This correspondence, in turn, gives us correspondence between the input and the individual vectorizers, so the input can be warped to each one for a combined textural analysis. This procedure requires adding pose to the existing state variables of shape, texture, and similarity transform P. The output of this multi-pose vectorizer would be useful for pose estimation and pose-invariant face recognition.

7 Conclusion

In this paper, we first introduced a *vectorized* image representation, a feature-based representation where correspondence has been established with respect to a reference image. Two image measurements are made at the feature points. First, feature geometry, or shape, is represented by the (x, y) feature locations relative to the standard face shape. Second, grey levels, or texture, are represented by mapping image grey levels onto the standard face shape. Given this definition, primary focus of this paper is to explore an automatic technique for computing this vectorized representation for face images.

To design an algorithm for vectorizing images, or a "vectorizer", we observed that the two representations can be linked. That is, for textural analysis, the shape component can be used to geometrically normalize an image so that features are properly aligned. Conversely, for shape analysis, the textural analysis can be used to create a reference image that reconstructs a geometrically normalized version of the input. We can then compute shape by finding correspondence between the reference image, which is at standard shape, and the input. The main idea of our vectorizer is to exploit the natural feedback between the texture and shape computations by iterating back and forth between the two until the shape/texture representation converges. We have demonstrated an efficient implementation of the vectorizer using a hierarchical coarse-to-fine strategy.

Two applications of the shape component were explored, facial feature finding and the registration of two faces. In our feature finding experiments, eyes, nose, mouth, and eyebrow features were located in 124 test images of 62 people at two different poses, and only one mouth feature was missed by the system. In the second application, one wants to generate new views of a "novel" face given just one view. Prior knowledge of a facial transformation such as a rotation is represented by 2D example images of a "prototype" face undergoing the transformation. The problem here is to register the "novel" face with a prototype face. We showed how to perform this registration step using features located by the vectorized shape component.

References

[1] K. Aizawa, H. Harashima, and T. Saito. Model-based analysis synthesis image coding (MBASIC) system for a person's face. *Signal Processing: Image Communication*, 1:139–152, 1989.

[2] Shigeru Akamatsu, Tsutomu Sasaki, Hideo Fukamachi, Nobuhiko Masui, and Yasuhito Suenaga. An accurate and robust face identification scheme. In *Proceedings Int. Conf. on Pattern Recognition*, volume 2, pages 217–220, The Hague, The Netherlands, 1992.

[3] Takaaki Akimoto, Yasuhito Suenaga, and Richard S. Wallace. Automatic creation of 3D facial models. *IEEE Computer Graphics and Applications*, 13(5):16–22, 1993.

[4] Adam Baumberg and David Hogg. Learning flexible models from image sequences. In *Proceedings of the European Conference on Computer Vision*, pages 299–308, Stockholm, Sweden, 1994.

[5] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In *SIGGRAPH '92 Proceedings*, pages 35–42, Chicago, IL, 1992.

[6] Jezekiel Ben-Aire and K. Raghunath Rao. On the recognition of occluded shapes and generic faces using multiple-template expansion matching. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 214–219, New York, NY, 1993.

[7] James R. Bergen and Edward H. Adelson. Hierarchical, computationally efficient motion estimation algorithm. *J. Opt. Soc. Am. A*, 4(13):P35, 1987.

[8] James R. Bergen, P. Anandan, Keith J. Hanna, and Rajesh Hingorani. Hierarchical model-based motion estimation. In *Proceedings of the European Conference on Computer Vision*, pages 237–252, Santa Margherita Ligure, Italy, June 1992.

[9] J.R. Bergen and R. Hingorani. Hierarchical motion-based frame rate conversion. Technical report, David Sarnoff Research Center, Princeton, New Jersey, April 1990.

[10] D. Beymer, A. Shashua, and T. Poggio. Example based image analysis and synthesis. A.I. Memo No. 1431, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1993.

[11] David Beymer and Tomaso Poggio. Face recognition from one example view. A.I. Memo No. 1536, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1995.

[12] Andrew Blake and Michael Isard. 3D position, attitude and shape input using video tracking of hands and lips. In *SIGGRAPH '94 Proceedings*, pages 185–192, Orlando, FL, 1994.

[13] Peter J. Burt. Smart sensing within a pyramid vision machine. *Proceedings of the IEEE*, 76(8):1006–1015, August 1988.

[14] Peter J. Burt and Edward H. Adelson. The laplacian pyramid as a compact image code. *IEEE Trans. on Communications*, COM-31(4):532–540, April 1983.

[15] T.F. Cootes and C.J. Taylor. Active shape models - 'Smart snakes'. In David Hogg and Roger Boyle, editors, *Proc. British Machine Vision Conference*, pages 266–275. Springer Verlag, 1992.

[16] T.F. Cootes and C.J. Taylor. Using grey-level models to improve active shape model search. In *Proceedings Int. Conf. on Pattern Recognition*, volume 1, pages 63–67, Jerusalem, Israel, 1994.

[17] T.F. Cootes, C.J. Taylor, A. Lanitis, D.H. Cooper, and J. Graham. Building and using flexible models incorporating grey-level information. In *Proceedings of the International Conference on Computer Vision*, pages 242–246, Berlin, May 1993.

[18] Ian Craw and Peter Cameron. Parameterizing images for recognition and reconstruction. In *Proc. British Machine Vision Conference*, pages 367–370, 1991.

[19] Ian Craw and Peter Cameron. Face recognition by computer. In David Hogg and Roger Boyle, editors, *Proc. British Machine Vision Conference*, pages 498–507. Springer Verlag, 1992.

[20] Irfan A. Essa and Alex Pentland. A vision system for observing and extracting facial action parameters. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 76–83, Seattle, WA, 1994.

[21] Michael J. Jones and Tomaso Poggio. Model-based matching of line drawings by linear combinations of prototypes. In *Proceedings of the International Conference on Computer Vision*, pages 531–536, Boston, Massachusetts, June 1995.

[22] M. Kirby and L. Sirovich. Application of the Karhunen-Loeve procedure for the characterization of human faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1):103–108, 1990.

[23] A. Lanitis, C.J. Taylor, and T.F. Cootes. A unified approach to coding and interpreting face images. In *Proceedings of the International Conference on Computer Vision*, pages 368–373, Cambridge, MA, June 1995.

[24] Peter Litwinowicz and Lance Williams. Animating images with drawings. In *SIGGRAPH '94 Proceedings*, pages 409–412, Orlando, FL, 1994.

[25] Baback Moghaddam and Alex Pentland. Probabilistic visual learning for object detection. In *Proceedings of the International Conference on Computer Vision*, pages 786–793, Cambridge, MA, June 1995.

[26] Alex Pentland, Baback Moghaddam, and Thad Starner. View-based and modular eigenspaces for face recognition. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 84–91, Seattle, WA, 1994.

[27] T. Poggio. 3D object recognition: on a result by Basri and Ullman. Technical Report # 9005–03, IRST, Povo, Italy, 1990.

[28] T. Poggio and S. Edelman. A network that learns to recognize three-dimensional objects. *Nature*, 343(6255):263–266, January 1990.

[29] Tomaso Poggio and Roberto Brunelli. A novel approach to graphics. A.I. Memo No. 1354, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1992.

[30] Tomaso Poggio and Thomas Vetter. Recognition and structure from one 2D model view: Observations on prototypes, object classes, and symmetries. A.I. Memo No. 1347, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1992.

[31] Daniel Reisfeld, Nur Arad, and Yehezkel Yeshurun. Normalization of face images using few anchors. In *Proceedings Int. Conf. on Pattern Recognition*, volume 1, pages 761–763, Jerusalem, Israel, 1994.

[32] Daniel Reisfeld and Yehezkel Yeshurun. Robust detection of facial features by generalized symmetry. In *Proceedings Int. Conf. on Pattern Recognition*, volume 1, pages 117–120, The Hague, The Netherlands, 1992.

[33] M.A. Shackleton and W.J. Welsh. Classification of facial features for recognition. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 573–579, Lahaina, Maui, Hawaii, 1991.

[34] Pawan Sinha. Object recognition via image invariances. *Investigative Ophthalmology and Visual Science*, 35(4):1626, 1994.

[35] Kah-Kay Sung and Tomaso Poggio. Example-based learning for view-based human face detection. In *Proceedings Image Understanding Workshop*, volume II, pages 843–850, Monterey, CA, November 1994.

[36] Demetri Terzopoulos and Keith Waters. Analysis of facial images using physical and anatomical models. In *Proceedings of the International Conference on Computer Vision*, pages 727–732, Osaka, Japan, December 1990.

[37] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[38] Shimon Ullman and Ronen Basri. Recognition by linear combinations of models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):992–1006, 1991.

[39] Keith Waters and Demetri Terzopoulos. Modelling and animating faces using scanned data. *The Journal of Visualization and Computer Animation*, 2:123–128, 1991.

[40] George Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, California, 1990.

[41] Alan L. Yuille, Peter W. Hallinan, and David S. Cohen. Feature extraction from faces using deformable templates. *International Journal of Computer Vision*, 8(2):99–111, 1992.