MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 1536 September, 1995

C.B.C.L. Paper No. 121

Face Recognition From One Example View

David Beymer and Tomaso Poggio

email: beymer@ai.mit.edu, tp@ai.mit.edu

Abstract

To create a pose-invariant face recognizer, one strategy is the view-based approach, which uses a set of example views at different poses. But what if we only have one example view available, such as a scanned passport photo – can we still recognize faces under different poses? Given one example view at a known pose, it is still possible to use the view-based approach by exploiting prior knowledge of faces to generate *virtual views*, or views of the face as seen from different poses. To represent prior knowledge, we use 2D example views of prototype faces under different rotations. We will develop example-based techniques for applying the rotation seen in the prototypes to essentially "rotate" the single real view which is available. Next, the combined set of one real and multiple virtual views is used as example views in a view-based, pose-invariant face recognizer. Our experiments suggest that for expressing prior knowledge of faces, 2D example-based approaches should be considered alongside the more standard 3D modeling techniques.

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. D. Beymer is supported by a Howard Hughes Doctoral Fellowship from the Hughes Aircraft Company.

1 Introduction

Existing work in face recognition has demonstrated good recognition performance on frontal, expressionless views of faces with controlled lighting (see Baron [4], Turk and Pentland [48], Bichsel [11], Brunelli and Poggio [14], and Gilbert and Yang [20]). One of the key remaining problems in face recognition is to handle the variability in appearance due to changes in pose, expression, and lighting conditions. There has been some recent work in this direction, such as pose-invariant recognizers (Pentland, *et. al.* [34], Beymer [10]) and deformable template approaches (Manjunath, *et. al.* [30]). In addition to recognition, richer models for faces have been studied for analyzing varying illumination (Hallinan [22]) and expression (Yacoob and Davis [55], Essa and Pentland [19]).

In this paper, we address the problem of recognizing faces under varying pose when only one example view per person is available. For example, perhaps just a driver's license photograph is available for each person in the database. If we wish to recognize new images of these people under a range of viewing directions, some of the new images will differ from the single view by a rotation in depth. Is recognition still possible?

There are a few potential approaches to the problem of face recognition from one example view. For example, the invariant features approach records features in the example view that do not change as pose-expression-lighting parameters change, features such as color or geometric invariants. While not yet applied to face recognition, this approach has been used for face detection under varying illumination (Sinha [45]) and for indexing of packaged grocery items using color (Swain and Ballard [46]).

In the flexible matching approach (von der Malsburg and collaborators [30][25]), the input image is deformed in 2D to match the example view. In [30], the deformation is driven by a matching of local "end-stop" features so that the resulting transformation between model and input is like a 2D warp rather than a global, rigid transform. This enables the deformation to match input and model views even though they may differ in expression or out-of-plane rotations. A deformation matching the input with a model view is evaluated by a cost functional that measures both the similarity of matched features and the geometrical distortion induced by the deformation. In this method, the difficulties include (a) constructing a generally valid cost functional, and (b) the computational expense of a non-convex optimization problem at run-time. However, since this matching mechanism is quite general (it does not take into consideration any prior model of human facial expression or 3D structure), it may be used for a variety of objects.

Generic 3D models of the human face can be used to predict the appearance of a face under different pose-expression-lighting parameters. For synthesizing images of faces, 3D facial models have been explored in the computer graphics, computer vision, and model-based image coding communities (Aitchison and Craw[1], Kang, Chen, and Hsu[24], Essa and Pentland [19], Akimoto, Suennaga, and Wallace[3], Waters and Terzopoulos[47], Aizawa, Harashima, and Saito[2]). In the 3D technique, face shape is represented either by a polygonal model or by a more complicated multilayer mesh that simulates tissue. Once a 2D face image is texture mapped onto the 3D model, the face can be treated as a traditional 3D object in computer graphics, undergoing 3D rotations or changes in light source position. Faces are texture mapped onto the 3D model either by specifying corresponding facial features in both the image and 3D model or by recording both 3D depth and color image data simultaneously by using specialized equipment like the Cyberware scanner. Prior knowledge for expression has been added to the 3D model by embedding muscle forces that deform the 3D model in a way that mimics human facial muscles.

A generic 3D model could also be applied to our scenario of pose-invariant face recognition from one example view. The single view of each person could be texture mapped onto a 3D model, and then the 3D model could be rotated to novel poses. Applying this strategy to face recognition, to our knowledge, has not yet been explored.

While 3D models are one method for using prior knowledge of faces to synthesize new views from just one view, in this paper we investigate representing this prior face knowledge in an example-based manner, using 2D views of prototype faces. Since we address the problem of recognition under varying pose, the views of prototype faces will sample different rotations out of the image plane. In principle, though, different expressions and lightings can be modeled by sampling the prototype views under those parameters. Given one view of a person, we will propose two methods for using the information in the prototype views to synthesize new views of the person, views from different rotations in our case. Following Poggio and Vetter [39], we call these synthesized views *virtual views*.

Our motivation for using the example-based approach is its potential for being a simple alternative to the more complicated 3D model-based approach. Using an example-based approach to bypass 3D models for 3D object recognition was first explored in the linear combinations approach to recognition (Ullman and Basri [49], Poggio [35]). In linear combinations, one can show that a 2D view of an object under rigid 3D transformation can be written as a linear combination of a small set of 2D example views, where the 2D view representation is a vector of (x, y) locations of a set of feature points. This is valid for a range of viewpoints in which a number of feature points are visible in all views and thus can be brought into correspondence for the view representation. This suggests an object may be represented using a set of 2D views instead of a 3D model.

Poggio and Vetter [39] have discussed this linear combinations approach in the case where only one example view is available for an object, laying the groundwork for virtual views. Normally, with just one view, 3D recognition is not possible. However, any method for generating additional object views would enable a recognition system to use the the linear combinations approach. This motivated Poggio and Vetter to introduce the idea of using prior knowledge of object class to generate virtual views. Two types of prior knowledge were explored, knowledge of 3D object symmetry and example images of prototypical objects of the same class. In the former, the mirror reflection of the single example can be generated, and the latter leads to the idea of linear classes, which we will explain and use later in this paper.

In this paper, after discussing methods for generating virtual views, we evaluate their usefulness in a view-based, pose-invariant face recognizer. Given only one real example view per person, we will synthesize a set of rotated virtual views, views that cover up/down and left/right rotations. The combined set of one real and multiple virtual views will be used as example views in a view-based face recognizer. Recognition performance will be reported on a separate test set of faces that cover a range of rotations both in and out of the image plane.

Independent from our work, Lando and Edelman [26] have recently investigated the same overall question – generalization from a single view in face recognition – using a similar example-based technique for representing prior knowledge of faces. In addition, Maurer and von der Malsburg [31] have investigated a technique for transforming their "jet" features across rotations in depth. Their technique is more 3D than ours, as it uses a local planarity assumption and knowledge of local surface normals.

2 Vectorized image representation

Our example-based techniques for generating virtual views use a *vectorized* face representation, which is an ordered vector of image measurements taken at a set of facial feature points. These features can run the gamut from sparse features with semantic meaning, such as the corners of the eyes and mouth, to pixel level features that are defined by the local grey level structure of the image. By an ordered vector, we mean that the facial features have been enumerated $f_1, f_2, \ldots, f_n$, and that the vector representation first contains measurements from f_1, then f_2, etc. The measurements at a given feature will include its (x, y) location – a measure of face "shape" – and local image color or intensity – a measure of face "texture". The key part of this vectorized representation is that the facial features $f_1, f_2, \ldots, f_n$ are effectively put into correspondence across the face images being "vectorized". For example, if f_1 is the outer corner of the left eye, then the first three elements of our vector representation will refer to the $(x_1, y_1, \text{intensity-patch}(x_1, y_1))$ measurements of that feature point for any face being vectorized.

2.1 Shape

Given the locations of features $f_1, f_2, \ldots, f_n$, shape is represented by a vector $\mathbf{y}$ of length $2n$ consisting of the concatenation of the x and y coordinate values

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

In our notation, if an image being vectorized has an identifying subscript (e.g. i_a), then the vector $\mathbf{y}$ will carry the same subscript, $\mathbf{y}_a$. The coordinate system used for measuring x and y will be one normalized by using the eye locations to fix interocular distance and remove head tilt. By factoring out the 2D aspects of pose, the remaining variability in shape vectors will be caused by expressions, rotations out of the image plane, and the natural variation in the configuration of features seen across people.

This vectorized representation for 2D shape has been widely used, including network-based object recognition (Poggio and Edelman [37]), the linear combinations approach to recognition (Ullman and Basri [49], Poggio [35]), active shape models (Cootes and Taylor [15], Cootes, *et al.* [16]) and face recognition (Craw and Cameron [17][18]). In these shape vectors, a sparse set of feature points, on the order of 10's of features, are either manually placed on the object or located using a feature finder. For a face, example feature points may include the inner and outer corners of the eyes, the corners of the mouth, and points along the eyebrows and sides of the face.

In this paper we use a dense representation of one feature per pixel, a representation originally suggested to us by the object recognition work of Shashua [43]. Compared to a sparser representation, the pixelwise representation increases the difficulty of finding correspondences. However, we have found that a standard optical flow algorithm [7], preceded by normalization based on the eye locations, can do a good job at automatically computing dense pixelwise correspondences. After defining one image as a "reference" image, the (x, y) locations of feature points of a new image are computed by finding optical flow between the two images. Thus the shape vector of the new image, really a "relative" shape, is described by a flow or a vector field of correspondences relative to a standard reference shape. Our face vectorizer (see Beymer [9]), which uses optical flow as a subroutine, is also used to automatically compute the vectorized representation.

Optical flow matches features in the two frames using the local grey level structure of the images. As opposed to a feature finder, where the "semantics" of features is determined in advance by the particular set of features sought by the feature finder, the reference image provides shape "semantics" in the relative representation. For example, to find the corner of the left eye in a relative shape, one follows the vector field starting from the left eye corner pixel in the reference image.

Correspondence with respect to a reference shape, as computed by optical flow, can be expressed in our vector notation as the difference between two vectorized shapes. Let us chose a face shape $\mathbf{y}_{std}$ to be the reference. Then the shape of an arbitrary face $\mathbf{y}_a$ is represented by the geometrical difference $\mathbf{y}_a - \mathbf{y}_{std}$, which we shall abbreviate $\mathbf{y}_{a-std}$. This is still a vector of length $2n$, but now it is a vector field of correspondences between images i_a and i_{std}. In addition, we keep track of the reference frame by using a superscript, so we add the superscript std to the shape $\mathbf{y}^{std}_{a-std}$. The utility of keeping track of the reference image will become more apparent when describing operations on shapes. Fig. 1 shows the shape

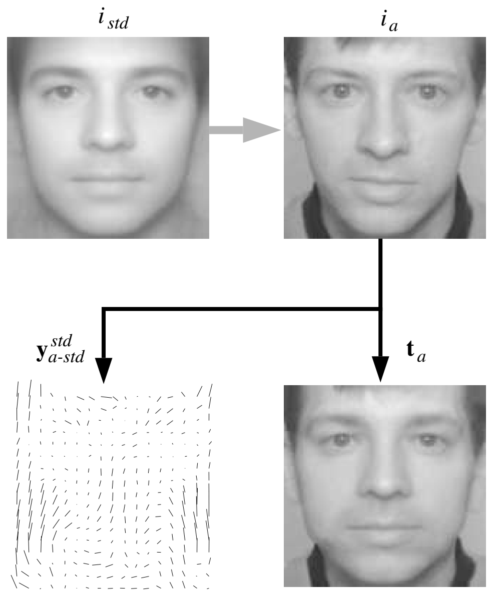


Figure 1: Our vectorized representation for image i_a with respect to the reference image i_{std} at standard shape. First, pixelwise correspondence is computed between i_{std} and i_a, as indicated by the grey arrow. Shape $\mathbf{y}^{std}_{a-std}$ is a vector field that specifies a corresponding pixel 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. In this figure, image i_{std} is the mean grey level image of 55 example faces that have been warped to the standard reference shape $\mathbf{y}_{std}$.

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. This relative shape representation has been used by Beymer, Shashua, and Poggio [8] in an example-based approach to image analysis and synthesis.

2.2 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 a common reference shape. This strategy for representing texture has been used, for example, in the face recognition works of Craw and Cameron [17], and Shackleton and Welsh [42]. 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. 1 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 [28], Wolberg [54]), radial basis functions (Reisfeld, Arad, and Yeshurun [40]), 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.

For our vectorized representation, we have chosen a dense, pixelwise set of features. What are some of the tradeoffs with respect to a sparser set of features? Texture processing is simplified over the sparse case since we avoid texture mapping and sparse data interpolation techniques, instead employing a simple 2D warping algorithm. Additionally, though, using a pixelwise representation makes the vectorized representation very simple conceptually: we can think of three measurements being made per feature $(x, y, I(x,y))$. The price we pay for this simplicity is a difficult correspondence problem. In section 5 we describe three correspondence techniques we explored for computing the vectorized image representation.

3 Prior knowledge of object class: prototype views

In our example-based approach for generating virtual views, prior knowledge of face transformations such as changes in rotation or expression are represented by 2D views of prototypical faces. Let there be N prototype faces $p_j, 1 \leq j \leq N$, where the prototypes are chosen to be representative of the variation in the class of faces. Unlike non-prototype faces – for which we only have a single example view – many views are available for each prototype p_j.

Given a single real view of a novel face at a known pose, we wish to transform the face to produce a rotated virtual view. Call the known pose of the real view the *standard* pose and the pose of the desired virtual view the *virtual* pose. Images of the prototype faces are then collected for both the standard and virtual poses. As shown in Fig. 2, let

$$\begin{aligned} i_{p_j} &= \text{set of } N \text{ prototype views at standard pose,} \\ i_{p_j,r} &= \text{set of } N \text{ prototype views at virtual pose,} \end{aligned}$$

where $1 \leq j \leq N$. Since we wish to synthesize many virtual views from the same standard pose, sets of prototype views at the virtual pose will be acquired for all the desired virtual views.

The techniques we explore for generating virtual views work with the vectorized image representation introduced in the previous section. That is, the prototype images i_{p_j} and $i_{p_j,r}$ have been vectorized, producing shape vectors $\mathbf{y}_{p_j}$ and $\mathbf{y}_{p_j,r}$ and texture vectors $\mathbf{t}_{p_j}$ and $\mathbf{t}_{p_j,r}$. The specific techniques we used to vectorize images will be discussed in section 5.

In the vectorized image representation, a set of images are brought into correspondence by locating a common set of feature points across all images. Since the set of prototype views contain a variety of both people and viewpoints, our definition of the vectorized representation implies that correspondence needs to be computed

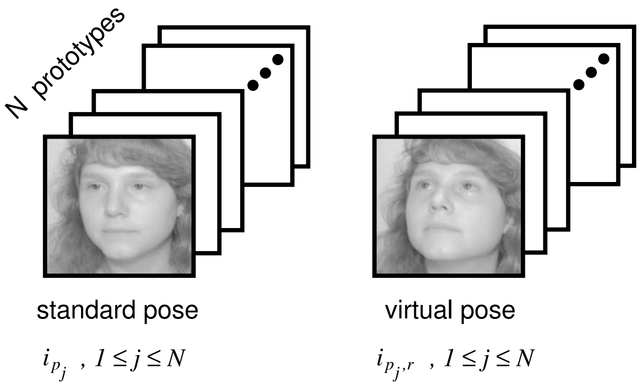


Figure 2: To represent prior knowledge of a facial transform (rotation upwards in the figure), views of N prototype faces are collected at the standard and virtual poses.

across different viewpoints as well as different people. However, the two techniques for generating virtual views, parallel deformation and linear classes, have different requirements in terms of correspondence across viewpoint. Parallel deformation requires these correspondences, so the prototype views are vectorized as one large set. On the other hand, linear classes does not require correspondence across viewpoints, so the set of images is partitioned by viewpoint and separate vectorizations defined for each viewpoint. In this latter case, vectorization is simply handling correspondence across the different prototypes at a fixed pose.

4 Virtual views synthesis techniques

In this section we explore two techniques for generating virtual views of a "novel" face for which just one view is available at standard pose:

1. *Linear classes.* Using multiple prototype objects, first write the novel face as a linear combination of prototypes at the standard pose, yielding a set of linear prototype coefficients. Then synthesize the novel face at the virtual pose by taking the linear combination of prototype objects at the virtual pose using the same set of coefficients. Using this approach, as discussed in Poggio [36] and Poggio and Vetter [39], it is possible to "learn" a direct mapping from standard pose to a particular virtual pose.

2. *Parallel deformation.* Using just one prototype object, measure the 2D deformation of object features going from the standard to virtual view. Then map this 2D deformation onto the novel object and use the deformation to distort, or warp, the novel image from the standard pose to the virtual one. The technique has been explored previously by Brunelli and Poggio [38] within the context of an "example-based" approach to computer graphics and by researchers in performance-driven animation (Williams [52][53], Patterson, Litwinowicz, and Greene [33]).

For notation, let i_n be the single real view of the novel face in standard pose. The virtual view will be denoted $i_{n,r}$.

4.1 Linear Classes

Because the theory of linear classes begins with a modeling assumption in 3D, let us generalize the 2D vectorized image representation to a 3D object representation. Recall that the 2D image vectorization is based on establishing feature correspondence across a set of 2D images. In 3D, this simply becomes finding a set of corresponding 3D points for a set of objects. The feature points are distributed over the face in 3D and thus may not all be visible from any one single view. Two measurements are made at each 3D feature point:

1. *Shape.* The (x, y, z) coordinates of the feature point. If there are n feature points, the vector $\mathbf{Y}$ will be a vector of length $3n$ consisting of the x, y, and z coordinate values.

2. *Texture.* If we assume that the 3D object is Lambertian and fix the lighting direction $\vec{l} = (l_x, l_y, l_z)$, we can measure the intensity of light reflected from each feature point, independent of viewpoint. At the ith feature point, the intensity $\mathbf{T}[i]$ is given by

$$\mathbf{T}[i] = \rho[i]\,(\vec{\eta}[i] \cdot \vec{l}), \quad (1)$$

where $\rho[i]$ is the albedo, or local surface reflectance, of feature i and $\vec{\eta}[i]$ is the local surface normal at feature i.[1]

The texture vector $\mathbf{T}$ is not an image; one can think of it as a texture that is mapped onto the 3D shape $\mathbf{Y}$ given a particular set of lighting conditions $\vec{l}$. One helpful way to visualize of the texture vector $\mathbf{T}$ is a sampling of image intensities in a cylindrical coordinate system that covers feature points over the entire face. This is similar to that produced by the Cyberware scanner.

Consider the relationship between 3D vectorized shape $\mathbf{Y}$ and texture $\mathbf{T}$ and their counterpart 2D versions $\mathbf{y}$ and $\mathbf{t}$. The projection process of going from 3D shape $\mathbf{Y}$ to 2D shape $\mathbf{y}$ consists of a 3D rotation, occlusion of a set of non-visible feature points, and orthographic projection. Mathematically, we model this using a matrix L

$$\mathbf{y} = L\mathbf{Y}, \quad (2)$$

[1] Vetter and Poggio [51] have explored the implications of 3D linear combinations of shape on image grey levels, or texture. If an object is a linear combination of prototype objects in 3D, then so are the surface normals. Thus, under Lambertian shading with constant albedo over each object, the grey level image of the novel object should be the same linear combination of the grey level prototype images.

Also, it is not strictly necessary for the object to be Lambertian; equation (1) could be a different functional form of ρ, $\vec{\eta}$, and $\vec{l}$. What is necessary is that $\mathbf{T}[i]$ is independent of lighting and viewing direction, which may be achieved by fixing the light source and assuming that the object is Lambertian.

where matrix L is the product of a 3D rotation matrix R, an occlusion matrix D that simply drops the coordinates of the occluded points, and orthographic projection O

$$L = ODR.$$

Note that L is a linear projection operator.

Creating a 2D texture vector $\mathbf{t}$ at a particular viewpoint $\vec{v}$ involves in some sense "projecting" the 3D texture $\mathbf{T}$. This is done by selecting the feature points that are visible in the standard shape at viewpoint $\vec{v}$

$$\mathbf{t} = D\mathbf{T}, \tag{3}$$

where D is a matrix that drops points occluded in the given viewpoint. Thus, viewpoint is handled in D; the lighting conditions are fixed in $\mathbf{T}$. Like operator L, D is a linear operator.

The idea of linear classes is based on the assumption that the space of 3D object vectorizations for objects of a given class is linearly spanned by a set of prototype vectorizations. That is, the shape $\mathbf{Y}$ and texture $\mathbf{T}$ of a class member can be written as

$$\mathbf{Y} = \sum_{j=1}^{N} \alpha_j \mathbf{Y}_{p_j} \quad \text{and} \quad \mathbf{T} = \sum_{j=1}^{N} \beta_j \mathbf{T}_{p_j} \tag{4}$$

for some set of α_j and β_j coefficients.

While the virtual views methods based on linear classes do not actually compute the 3D vectorized representation, the real view i_n is related to the destination virtual view $i_{n,r}$ through the 3D vectorization of the novel object. First, a 2D image analysis of i_n at standard pose estimates the α_j and β_j in equation (4) by using the prototype views i_{p_j}. Then the virtual view $i_{n,r}$ can be synthesized using the linear coefficients and the prototype views $i_{p_j,r}$. Let us now examine these steps in detail for the shape and texture of the novel face.

4.1.1 Virtual shape

Given the vectorized shape of the novel person $\mathbf{y}_n$ and the prototype vectorizations $\mathbf{y}_{p_j}$ and $\mathbf{y}_{p_j,r}, 1 \leq j \leq N$, linear classes can be used to synthesize vectorized shape $\mathbf{y}_{n,r}$ at the virtual pose. This idea was first developed by Poggio and Vetter [39].

In linear classes, we assume that the novel 3D shape $\mathbf{Y}_n$ can be written as a linear combination of the prototype shapes $\mathbf{Y}_{p_j}$

$$\mathbf{Y}_n = \textstyle\sum_{j=1}^{N} \alpha_j \mathbf{Y}_{p_j}. \tag{5}$$

If the linear class assumption holds and the set of 2D views $\mathbf{y}_{p_j}$ are linearly independent, then we can solve for the α_j's at the standard view

$$\mathbf{y}_n = \textstyle\sum_{j=1}^{N} \alpha_j \mathbf{y}_{p_j} \tag{6}$$

and use the prototype coefficients α_j to synthesize the virtual shape

$$\mathbf{y}_{n,r} = \textstyle\sum_{j=1}^{N} \alpha_j \mathbf{y}_{p_j,r}. \tag{7}$$

This is true under orthographic projection. The mathematical details are provided in Appendix B.

While this may seem to imply that we can perform a 3D analysis based on one 2D view of an object, the linear class assumption cannot be verified using 2D views. Thus, from just the 2D analysis, the technique can be "fooled" into thinking that it has found a good set of linear coefficients when in fact equation (5) is poorly approximated. That is, the technique will be fooled when the actual 3D shape of the novel person is different from the 3D interpolated prototype shape in the right hand side of equation (5).

In solving equations (6) and (7), the linear class approach can be interpreted as creating a direct mapping from standard to virtual pose. That is, we can derive a function that maps from $\mathbf{y}$'s in standard pose to $\mathbf{y}$'s in the virtual pose. Let Y be a matrix where column j is $\mathbf{y}_{p_j}$, and let Y_r be a matrix where column j is $\mathbf{y}_{p_j,r}$. Then if we solve for equation (6) using linear least squares and plug the resulting α's into equation (7), then

$$\mathbf{y}_{n,r} = Y_r Y^{\dagger} \mathbf{y}_n, \tag{8}$$

where $Y^{\dagger}$ is the pseudoinverse $(Y^t Y)^{-1} Y^t$.

Another way to formulate the solution as a direct mapping is to train a network to learn the association between standard and virtual pose (see Poggio and Vetter [39]). The (input, output) pairs presented to the network during training would be the prototype pairs $(\mathbf{y}_{p_j}, \mathbf{y}_{p_j,r})$. A potential architecture for such a network is suggested by the fact that equation (8) can be implemented by a single layer linear network. The weights between the input and output layers are given simply by the matrix $Y_r Y^{\dagger}$.

4.1.2 Virtual texture

In addition to generating the shape component of virtual views, the prototypes can also be used to generate the texture of virtual views. Given the texture of a novel face $\mathbf{t}_n$ and the prototype textures $\mathbf{t}_{p_j}$ and $\mathbf{t}_{p_j,r}, 1 \leq j \leq N$, the concept of linear classes can be used to synthesize the virtual texture $\mathbf{t}_{n,r}$. This synthesized grey level texture is then warped or texture mapped onto the virtual shape to create a finished virtual view. The ideas presented in this section were developed by the authors and also independently by Vetter and Poggio [51].

To generate the virtual texture $\mathbf{t}_{n,r}$, we propose using the same linear class idea of approximation at the standard view and reconstruction at the virtual view. Similarly to the shape case, this relies on the assumption that the space of grey level textures $\mathbf{T}$ is linearly spanned by a set of prototype textures. The validity of this assumption is borne out by recent successful face recognition systems (e.g. eigenfaces, Pentland, *et al.* [34]). First, assume that the novel texture $\mathbf{T}_n$ can be written as a linear combination of the prototype textures $\mathbf{T}_{p_j}$

$$\mathbf{T}_n = \textstyle\sum_{j=1}^{N} \beta_j \mathbf{T}_{p_j}. \tag{9}$$

The analog of linear classes for texture, presented in Appendix B, says that if this assumption holds and the 2D textures $\mathbf{t}_{p_j}$ are linearly independent, then we should be able to decompose the real texture $\mathbf{t}_n$ in terms of the example textures $\mathbf{t}_{p_j}$

$$\mathbf{t}_n = \sum_{j=1}^{N} \beta_j \mathbf{t}_{p_j} \tag{10}$$

and use the same set of coefficients to reconstruct the texture of the virtual view

$$\mathbf{t}_{n,r} = \sum_{j=1}^{N} \beta_j \mathbf{t}_{p_j,r}. \tag{11}$$

Note that the texture $\mathbf{T}$ and hence the β_j coefficients are dependent on the lighting conditions. Thus, by computing different views $\mathbf{t}$ using the D operator, we are effectively rotating the camera around the object. The geometry between object and light source is kept fixed.

We have synthesized textures for rotations of 10 to 15 degrees between standard and virtual poses with reasonable results; see section 5 for example $\mathbf{t}_{n,r}$ images and section 6 for recognition experiments. In terms of computing $\mathbf{t}_{n,r}$ from $\mathbf{t}_n$, we can use the same linear solution technique as for shape (equation (8)).

4.2 Parallel deformation

While the linear class idea does not require the $\mathbf{y}$ vectors to be in correspondence between the standard and virtual views, if we add such "cross view" correspondence then the linear class idea can be interpreted as finding a 2D deformation from $\mathbf{y}_n$ to $\mathbf{y}_{n,r}$. Having shape vectors in cross view correspondence simply means that the $\mathbf{y}$ vectors in both poses refer to the *same* set of facial feature points. The advantage of computing this 2D deformation is that the texture of the virtual view can be generated by texture mapping directly from the original view i_n. This avoids the need for additional techniques to synthesize virtual texture at the virtual view.

To see the deformation interpretation, subtract equation (6) from (7) and move $\mathbf{y}_n$ to the other side, yielding

$$\mathbf{y}_{n,r} = \mathbf{y}_n + \sum_{j=1}^{N} \alpha_j (\mathbf{y}_{p_j,r} - \mathbf{y}_{p_j}). \tag{12}$$

Bringing shape vectors from the different poses together in the same equation is legal because we have added cross view correspondence. The quantity $\mathbf{\Delta y}_j = \mathbf{y}_{p_j,r} - \mathbf{y}_{p_j}$ is a 2D warp that specifies how prototype j's feature points move under the prototype transformation. Equation (12) modifies the shape $\mathbf{y}_n$ by a linear combination of these prototype deformations. The coefficients of this linear combination, the α_j's, are given by $Y^{\dagger}\mathbf{y}_n$, the solution to the approximation equation (6).

Consider as a special case the deformation approach with just one prototype. In this case, the novel face is deformed in a manner that imitates the deformation seen in the prototype. This is similar to performance-driven animation (Williams [52]), and Poggio and Brunelli [38], who call it *parallel deformation*, have suggested it as a computer graphics tool for animating objects when provided with just one view. Specializing equation (12) gives

$$\mathbf{y}_{n,r} = \mathbf{y}_n + (\mathbf{y}_{p,r} - \mathbf{y}_p), \tag{13}$$

where we have dropped the j subscripts on the prototype variable p. The deformation $\mathbf{\Delta y} = \mathbf{y}_{p,r} - \mathbf{y}_p$ essentially represents the prototype transform and is the same 2D warping as in the multiple prototypes case.

By looking at the one prototype case through specializing the original equations (6) and (7), we get $\mathbf{y}_n = \mathbf{y}_p$ and $\mathbf{y}_{n,r} = \mathbf{y}_{p,r}$. This seems to say that the virtual shape $\mathbf{y}_{n,r}$ is simply that of the prototype at virtual pose, so why should equation (13) give us anything different? However, the specialized equations, which approximate the novel shape by prototype shape, are likely to be poor approximations. Thus, we should really add error terms, writing $\mathbf{y}_n = \mathbf{y}_p + \mathbf{y}_{error_1}$ and $\mathbf{y}_{n,r} = \mathbf{y}_{p,r} + \mathbf{y}_{error_2}$. The error terms are likely to be highly correlated, so by subtracting the equations – as is done by parallel deformation – we cancel out the error terms to some degree.

4.3 Comparing linear classes and parallel deformation

What are some of the relative advantages of linear classes and parallel deformation? First, consider some of the advantages of linear classes over parallel deformation. Parallel deformation works well when the 3D shape of the prototype matches the 3D shape of the novel person. If the two 3D shapes differ enough, the virtual view generated by parallel deformation will appear geometrically distorted. Linear classes, on the other hand, effectively tries to construct a prototype that matches the novel shape by taking the proper linear combination of example prototypes. Another advantage of linear classes is that correspondence is not required between standard and virtual poses. Thus, linear classes may be able to cover a wider range of rotations out of the image plane as compared to parallel deformation.

One advantage of parallel deformation over linear classes is its ability to preserve peculiarities of texture such as moles or birthmarks. Parallel deformation will preserve such marks since it samples texture from the original real view of the novel person's face. For linear classes, it is most likely that a random mark on a person's face will be outside the linear texture space of the prototypes, so it will not be reconstructed in the virtual view.

5 Generating virtual views

In our approach to recognizing faces using just one example view per person, we first expand the example set by generating virtual views of each person's face. The full set of views that we would ultimately like to have for our view-based face recognizer are the set of 15 example views shown in Fig. 3 and originally used in the view-based recognizer of Beymer [10]. These views evenly sample the two rotation angles out of the image plane.

While Fig. 3 shows 15 real views, in virtual views we assume that only view **m4** is available and we synthesize the remaining 14 example views. For the single real view, an off-center view was favored over, say, a frontal view because of the recognition results for bilaterally symmetric objects of Poggio and Vetter [39]. When the single real view is from a nondegenerate pose (i.e. mirror reflection is not equal to original view), then the mirror reflection immediately provides a second view that can be used for recognition. The choice of an off-center view is also supported by the psychophysical experiments of Schyns and Bülthoff [41]. They found that when humans

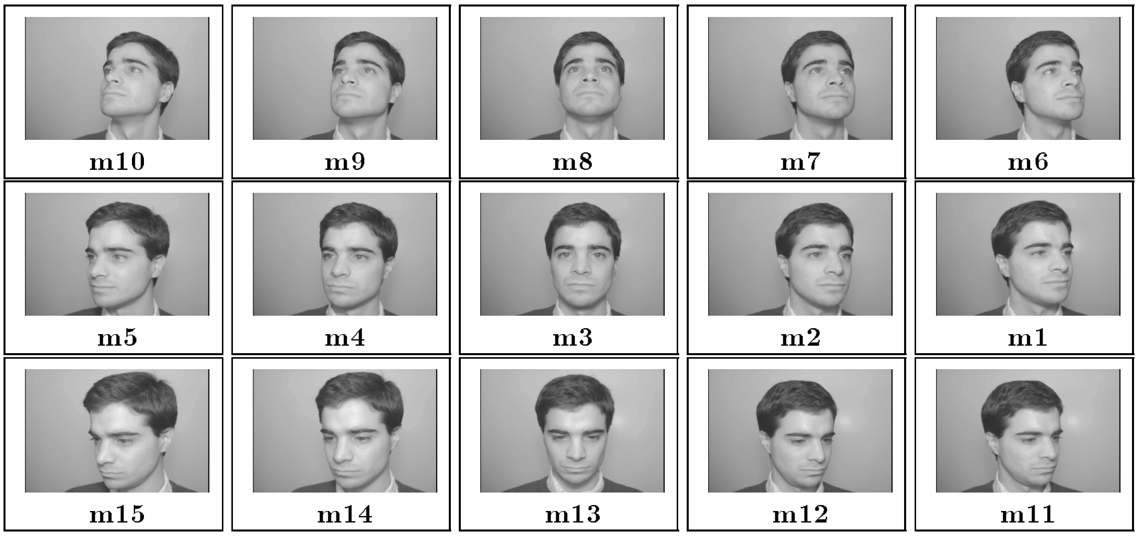


Figure 3: The view-based face recognizer uses 15 views to model a person's face. For virtual views, we assume that only one real view, view **m4**, is available and we synthesize the remaining 14.

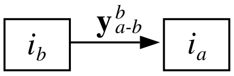


Figure 4: In relative shape, $\mathbf{y}^b_{a-b}$ denotes feature correspondence between i_b and i_a using i_b as a reference.

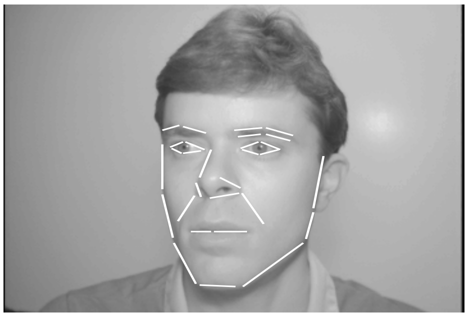


Figure 5: Manually specified line segments drive Beier and Neely's pixelwise correspondence technique.

are trained on just one pose and tested on many, recognition performance is better when the single training view is an off-center one as opposed to a frontal pose.

In completing the set of 15 example views, the 8 views neighboring **m4** will be generated using our virtual views techniques. Using the terminology of the theory section, view **m4** is the standard pose and each of the neighboring views are virtual poses. The remaining 6 views, the right two columns of Fig. 3, will be generated by assuming bilateral symmetry of the face and taking the mirror reflection of the left two columns.

But before describing our implementation of parallel deformation and linear classes, we need to define some operators on shape.

5.1 Shape operators

5.1.1 Vectorizing face images

Computing the vectorized representation is really a feature correspondence problem. The difficulty of this correspondence task depends on the difference between the two image arguments. Finding pixelwise correspondence between the images of two dissimilar people is inherently more difficult than dealing with two poses of the same person, both situations of which are encountered in virtual views. Thus, we have looked at three ways to vectorize faces, a manual method, optical flow, and a new automatic technique that we call an image vectorizer.

The pixelwise correspondence algorithms discussed in this section compute a relative shape $\mathbf{y}^b_{a-b}$, i.e. the shape $\mathbf{y}_a$ of image i_a with respect to a reference image i_b. This computation will be denoted using the **vect** operator

$$\mathbf{y}^b_{a-b} = \mathbf{vect}(i_a, i_b).$$

Pictorially, we visualize the shape $\mathbf{y}^b_{a-b}$ in Fig. 4 by drawing an arrow from i_b to i_a. Of course, given this relative shape $\mathbf{y}^b_{a-b}$, our original "absolute" definition of shape $\mathbf{y}^b_a$ can be computed by simply adding the shape $\mathbf{y}^b_b$, which is simply the x and y coordinate values of each pixel in i_b.

The **manual technique** is borrowed from Beier and Neely's morphing technique in computer graphics [5]. In their technique, a sparse set of corresponding line segment features manually placed on images i_a and i_b drive pixelwise correspondence between the two images (see Fig. 5). Points on the line segments are mapped exactly, and points in between are mapped using a weighted combination of the displacement fields generated by the line segment correspondences. This is one method we used for computing interperson correspondence – correspondence across different people. While this technique always works, it is manual. Ideally, we would like something automatic, which leads us to the next two techniques.

The **optical flow** technique uses the gradient-based, hierarchical method of Bergen and Hingorani [7] (also see Lucas and Kanade [29], Bergen, *et al.* [6]). Before applying optical flow, face images are brought into rough registration using the eyes, which were located manually. Optical flow is useful for computing correspondence among different rotated views of the same prototype. It works for interperson correspondence when the two people are similar enough in grey-level appearance, but this does not happen frequently enough to be useful.

Finally, our **image vectorizer** is a new method for computing pixelwise correspondence between an input and an "average" face shape $\mathbf{y}_{std}$. Beymer [9] provides the details; here we only set the problem up and sketch the solution. To model grey level face texture, the vectorizer uses a set of N shape free prototypes $\mathbf{t}_{p_j}$, the same set as described before for texture representation. "Vectorizing" an input image i_a means simultaneously solving for (1) an optical flow $\mathbf{y}^{std}_{a-std}$ that converts the input to the shape free representation, and (2) a set of linear prototype coefficients β_j used to construct a model image resembling the input. This is done by iteratively solving

$$i_a(\mathbf{x} + \mathbf{y}^{std}_{a-std}(\mathbf{x})) = \sum_{j=1}^{N} \beta_j \mathbf{t}_{p_j}$$

by alternating between the operations of optical flow and projection onto the prototypes until a stable solution is found. In this equation, $\mathbf{x}$ is a 2D point (x, y) in average shape $\mathbf{y}_{std}$. The iterative processing of shape and texture is similar to the active shape models of Cootes and Taylor [15], Cootes, *et al.* [16], and Lanitis, Taylor, and Cootes [27]. Jones and Poggio [23] also describe a related system that uses linear combinations of prototype shapes to analyze line drawings.

Correspondence between two arbitrary images can thus be found by vectorizing both, as now both images are in correspondence with the average shape. After vectorizing both images, one flow is inverted (which can be done with negation followed by a forward warp), and the two flows are then concatenated. This is an automatic technique for finding interperson correspondence.

5.1.2 Warping and shape manipulation operators

2D warping operations move pixel values back and forth between the reference shape $\mathbf{y}_b$ and the destination shape $\mathbf{y}_a$. A forward warp, **fwarp**, pushes pixels in the reference frame forward along the flow vectors to the destination shape. For example, back in Fig. 4, we can write $i_a = \mathbf{fwarp}(i_b, \mathbf{y}^b_{a-b})$. In general, we can push pixels along any arbitrary flow x, yielding the more general form of $i_{b+x} = \mathbf{fwarp}(i_b, \mathbf{y}^b_x)$. Note that the subscript of the image argument must match the superscript of the shape argument, implying that the image must be in the reference frame of the shape. Inversely, a backwards warp, **bwarp**, uses the flow as an index into the destination shape, bringing pixels in the destination shape back to the reference. In Fig. 4, we can write $i_b = \mathbf{bwarp}(i_a, \mathbf{y}^b_{a-b})$.

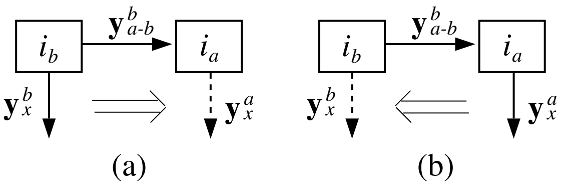


Figure 6: (a) To change the reference frame of flow $\mathbf{y}^b_x$ from i_a to i_b, the x and y components are forward warped along $\mathbf{y}^b_{a-b}$, producing the dotted flow $\mathbf{y}^a_x$. In (b), backwards warping is used to compute the inverse.

Between the forward and backward warping operations, implementing **bwarp** is the easier of the two. Each pixel in the reference image samples the destination image by following its flow vector. Since the destination location is usually between pixels, bilinear interpolation is used to produce a grey level value.

Forward warping is solved using the idea of four corner mapping (see Wolberg[54]). Basically, we invert the forward warping and then apply the backward warping algorithm. To invert the forward warping, repeat the following for every square source patch of four adjacent pixels in the source. Map the source patch to a quadrilateral in the destination image. Then for each destination pixel inside this quadrilateral, we estimate its position inside the quadrilateral treating the sides of the quadrilateral as a warped coordinate system. This position is used to map to a location in the original source patch.

In addition to warping operations, shapes can be combined using binary operations such as addition and subtraction. In adding and subtracting shapes, the reference frames of both shapes must be the same, and the subscripts of the shape arguments are added/subtracted to yield the subscripts of the results: $\mathbf{y}^b_{u\pm v} = \mathbf{y}^b_u \pm \mathbf{y}^b_v$.

The reference frame of a shape $\mathbf{y}^b_x$ can be changed from i_b to i_a by applying a forward warp with the shape $\mathbf{y}^b_{a-b}$. Shown pictorially in Fig. 6(a), the operation consists of separate 2D forward warps on the x and y components of $\mathbf{y}^b_x$ interpreted for the moment as images instead of vectors. Instead of pushing grey level pixels in the forward warp, we push the x and y components of the shape. The operation in Fig. 6(a) is denoted $\mathbf{y}^a_x = \mathbf{fwarp\text{-}vect}(\mathbf{y}^b_x, \mathbf{y}^b_{a-b})$. The inverse operation, shown in Fig. 6(b), is computed using two backwards warps instead of forward ones: $\mathbf{y}^b_x = \mathbf{bwarp\text{-}vect}(\mathbf{y}^a_x, \mathbf{y}^b_{a-b})$.

Finally, two flows fields $\mathbf{y}^a_{b-a}$ and $\mathbf{y}^b_{c-b}$ can be concatenated or composed to produce pixelwise correspondences between i_a and i_c, $\mathbf{y}^a_{c-a}$. Concatenation is shown pictorially in Fig. 7 and is denoted $\mathbf{y}^a_{c-a} = \mathbf{concat}(\mathbf{y}^a_{b-a}, \mathbf{y}^b_{c-b})$. The basic idea behind implementing this operator is to put both shapes in the same reference frame and then add. This is done by first computing $\mathbf{y}^a_{c-b} = \mathbf{bwarp\text{-}vect}(\mathbf{y}^b_{c-b}, \mathbf{y}^a_{b-a})$ followed by $\mathbf{y}^a_{c-a} = \mathbf{y}^a_{b-a} + \mathbf{y}^a_{c-b}$.

Having finished our primer on shape operators, we now describe how parallel deformation and linear classes were used to expand the example set with virtual views. Recognition results with these virtual views are summa-

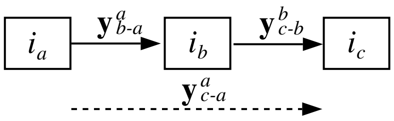


Figure 7: In flow concatentation, the flows $\mathbf{y}^a_{b-a}$ and $\mathbf{y}^b_{c-b}$ are composed to produce the dotted flow $\mathbf{y}^a_{c-a}$.

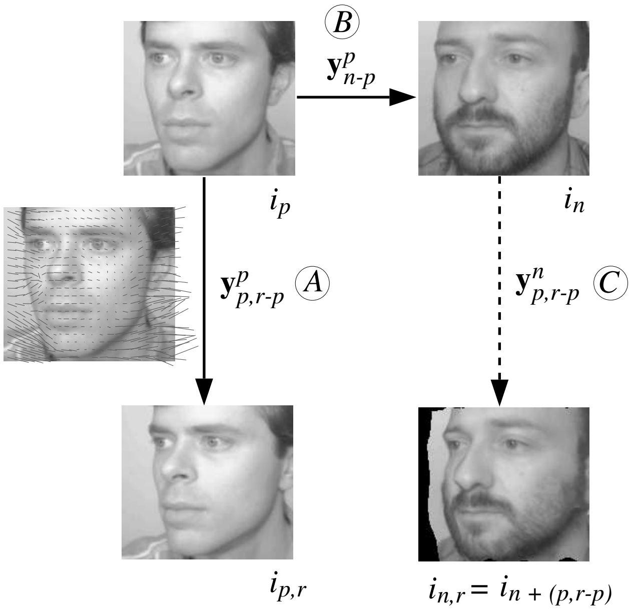


Figure 8: In parallel deformation, (A) the prototype flow $\mathbf{y}^p_{p,r-p}$ is first measured between $i_{p,r}$ and i_p, (B) the flow is mapped onto the novel face i_n, and (C) the novel face is 2D warped to the virtual view.

rized in the next section.

5.2 Parallel deformation

The goal of parallel deformation is to map a facial transformation observed on a prototype face onto a novel, non-prototype face. There are three steps in implementing parallel deformation: (a) recording the deformation $\mathbf{y}_{p,r} - \mathbf{y}_p$ on the prototype face, (b) mapping this deformation onto the novel face, and (c) 2D warping the novel face using the deformation. We now go over these steps in more detail, using as an example the prototype views and single novel view in Fig. 8.

First, we collect prototype views i_p and $i_{p,r}$ and compute the prototype deformation

$$\mathbf{y}^p_{p,r-p} = \mathbf{vect}(i_{p,r}, i_p)$$

using optical flow. Shown overlayed on the reference image on the left of Fig. 8, this 2D deformation specifies how to forward warp i_p to $i_{p,r}$ and represents our "prior knowledge" of face rotation. To assist the correspondence calculation, a sequence of four frames from standard to virtual pose is used instead of just two frames.

Figure 9: The prototypes used for parallel deformation. Standard poses are shown.

Pairwise optical flows are computed and concatenated to get the composite flow from first to last frame.

Next, the 2D rotation deformation is mapped onto the novel person's face by changing the reference frame of $\mathbf{y}^p_{p,r-p}$ from i_p to i_n. First, interperson correspondences between i_p and i_n are computed

$$\mathbf{y}^p_{n-p} = \mathbf{vect}(i_n, i_p)$$

and used to change the reference frame

$$\mathbf{y}^n_{p,r-p} = \mathbf{fwarp\text{-}vect}(\mathbf{y}^p_{p,r-p}, \mathbf{y}^p_{n-p}).$$

The flow $\mathbf{y}^n_{p,r-p}$ is the 2D rotation deformation mapped onto the novel person's standard view. As the interperson correspondences are difficult to compute, we evaluated two techniques for establishing feature correspondence: labeling features manually on both faces, and using our face vectorizer (see section 5.1.1 and Beymer [9]) to automatically locate features. More will be said about our use of these two approaches shortly.

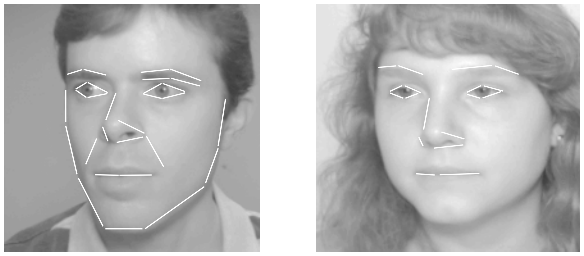
Finally, the texture from the original real view i_n is 2D warped onto the rotated face shape, producing the final virtual view

$$i_{n,r} = i_{n+(p,r-p)} = \mathbf{fwarp}(i_n, \mathbf{y}^n_{p,r-p}).$$

Referring to our running example in Fig. 8, the final virtual view is shown in the lower right.

In this procedure for parallel deformation, there are two main parameters that one may vary:

1. *The prototype.* As mentioned previously, the accuracy of virtual views generated by parallel deformation depends on the degree to which the 3D shape of the prototype matches the 3D shape of the novel face. Thus, one would expect different recognition results from different prototypes. We have experimented with virtual views generated using the three different prototypes shown in Fig. 9. In general, given a particular novel person, it is best to have a variety of prototypes to choose from and to try to select the one that is closest to the novel person in terms of shape.
2. *Approach for interperson correspondence.* In both the manual and automatic approaches, interperson correspondences are driven by the line segment features shown in Fig. 10. The automatic segments shown on the right were located using our face vectorizer from Beymer [9]. The manual segments on the left include some additional features not returned by the vectorizer, especially around the



example manual segments | example automatic segments

Figure 10: Parallel deformation requires correspondences between the prototype and novel person. These correspondences are driven by the segment features shown in the figure. The features on the left were manually located, and the features on the right were automatically located using the vectorizer.

sides of the face. Given these sets of correspondences, the interpolation method from Beier and Neely [5] (see section 5.1.1) is used to interpolate the correspondences to define a dense, pixelwise mapping from the prototype to novel face.

Figures 11 and 12 show example virtual views generated using prototype A with the real view in the center. Manual interperson correspondences were used in Fig. 11 and the image vectorizer in Fig. 12. To compare views generated from the different prototypes, Fig. 13 shows virtual views generated from all three prototypes. For comparison purposes, the real view of each novel person is shown on the right.

5.3 Linear Classes

We use the linear class idea to analyze the novel texture in terms of the prototypes at the standard view and reconstruct at the virtual view. In the analysis step at the standard view, we decompose the shape free texture of the novel view $\mathbf{t}_n$ in terms of the N shape free prototype views $\mathbf{t}_{p_j}$

$$\mathbf{t}_n = \sum_{j=1}^{N} \beta_j \mathbf{t}_{p_j}, \tag{14}$$

which results in a set of β_j prototype coefficients. But before solving this equation for the β_j, the novel view i_n and prototype views i_{p_j} must be vectorized to produce the geometrically normalized textures $\mathbf{t}_n$ and $\mathbf{t}_{p_j}, 1 \leq j \leq N$. Since the $\mathbf{t}_{p_j}$'s can be put into correspondence manually in an off-line step (using the interpolation technique of Beier and Neely [5]), the primary difficulty of this step is in converting i_n into its shape free representation $\mathbf{t}_n$. Since i_n is an **m4** view of the face, this step means finding correspondence between i_n and view **m4**'s standard face shape. Let this standard shape be denoted as $\mathbf{y}_{std}$.

Our image vectorizer (Beymer [9]) is used to solve for the correspondences $\mathbf{y}_{n-std}^{std}$ between i_n and standard shape $\mathbf{y}_{std}$. These correspondences can then be used to geometrically standardize i_n

$$\mathbf{t}_n(\mathbf{x}) = i_n(\mathbf{x} + \mathbf{y}_{n-std}^{std}(\mathbf{x})),$$

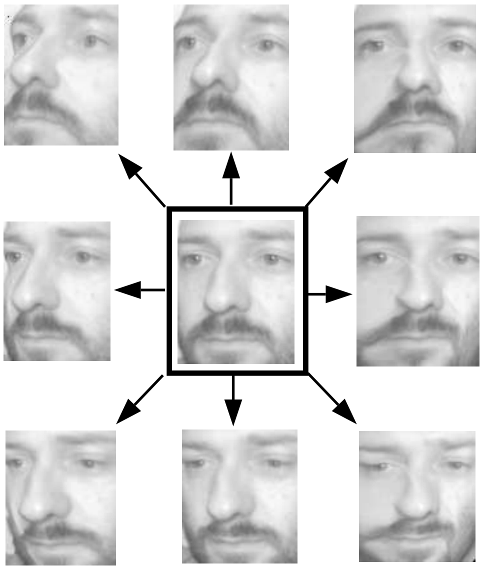


Figure 11: Example virtual views using parallel deformation. Prototype A was used, and interperson correspondence $\mathbf{y}_{n-p}^{p}$ was specified manually.

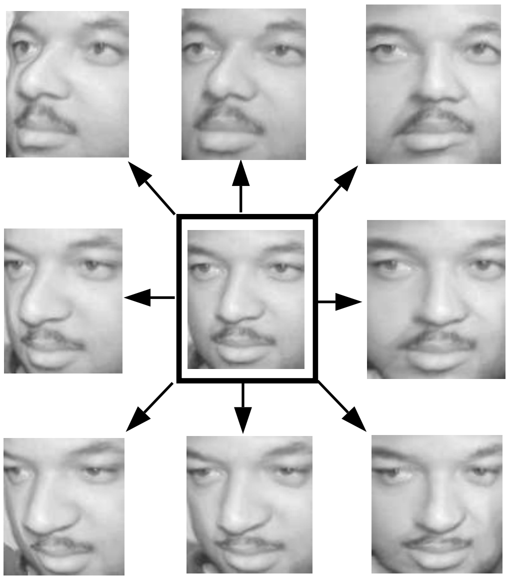


Figure 12: Example virtual views using parallel deformation. Prototype A was used, and interperson correspondence $\mathbf{y}_{n-p}^{p}$ was computed automatically using the image vectorizer.

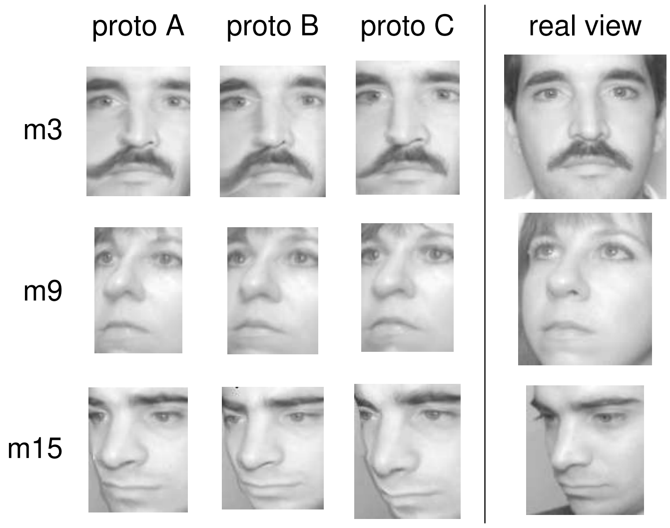


Figure 13: Example virtual views as the prototype person is varied. The corresponding real view of each novel person is shown on the right for comparison.

where $\mathbf{x}$ is an arbitrary 2D point (x, y) in standard shape. Fig. 14 on the left shows an example view i_n with some features automatically located by the vectorizer. The right side of the figure shows templates $\mathbf{t}_n$ of the eyes, nose, and mouth that have been geometrically normalized using the correspondences $\mathbf{y}_{n-std}^{std}$.

Next, the texture $\mathbf{t}_n$ is decomposed as a linear combination of the prototype textures, following equation (14). First, combine the β_j terms into a column vector β and define a matrix T of the prototype textures, where the jth column of T is $\mathbf{t}_{p_j}$. Then equation (14) can be rewritten as

$$\mathbf{t}_n = T\beta.$$

This can be solved using linear least squares, yielding

$$\beta = T^{\dagger}\mathbf{t}_n,$$

where $T^{\dagger}$ is the pseudoinverse $(T^tT)^{-1}T^t$.

The synthesis step assumes that the textural decomposition at the virtual view is the same as that at the standard view. Thus, we can synthesize the virtual texture

$$\mathbf{t}_{n,r} = \sum_{j=1}^{N} \beta_j \mathbf{t}_{p_j,r},$$

where $\mathbf{t}_{p_j,r}$ are the shape free prototypes that have been warped to the standard shape of the virtual view. As with the $\mathbf{t}_{p_j}$'s, the $\mathbf{t}_{p_j,r}$'s are put into correspondence manually in an off-line step. If we define a matrix T_r such that column j is $\mathbf{t}_{p_j,r}$, the analysis and synthesis steps can be written as a linear mapping from $\mathbf{t}_n$ to $\mathbf{t}_{n,r}$

$$\mathbf{t}_{n,r} = T_r T^{\dagger}\mathbf{t}_n.$$

This linear mapping was previously discussed in section 4.1 for generating virtual shapes.

Fig. 15 shows a set of virtual views generated using the analysis of Fig. 14. Note that the prototype views must be of the same set of people across all nine views. We used a prototype set of 55 people, so we had to specify manual correspondence (see Fig. 5) for 9 views of each person to set up the shape free views. When generating the virtual views for a particular person, we would, of course, remove him from the prototype set if he were initially present, following a cross validation methodology.

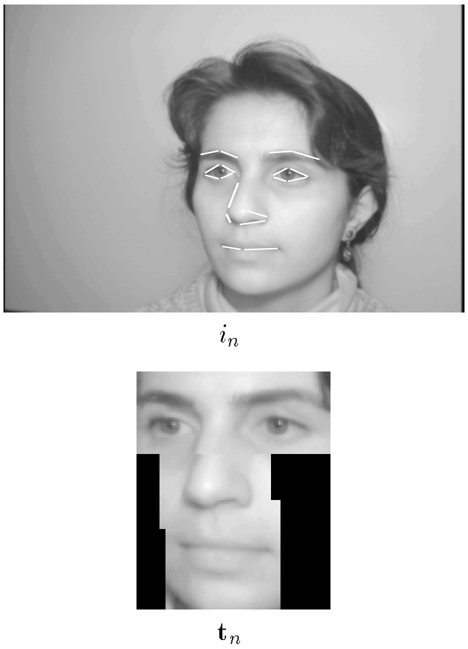


Figure 14: Using correspondences from our face vectorizer, we can geometrically normalize input i_n, producing the "shape free" texture $\mathbf{t}_n$.

Notice from Fig. 15 that by using the shape free textural representation, the virtual views in this experiment are decoupled from shape and hence all views are in the standard shape of the virtual pose. The only difference between the views of different people at a fixed pose will be their texture.

6 Experimental results

In this section we report the recognition rates obtained when virtual views were used in our view-based recognizer [10].

6.1 View-based recognizer

In our view-based face recognizer [10], the 15 example views of Fig. 3 are stored for each person to handle pose invariance. To recognize an input view, our recognizer uses a strategy of registering the input with the example views followed by template matching. To drive the registration step in the recognizer, a person- and pose-invariant feature finder first locates the irises and a nose lobe feature. Similar in flavor to the recognizer, the feature finder is template-based, using a large set of eyes-nose templates from a variety of "exemplar" people and the 15 example poses.

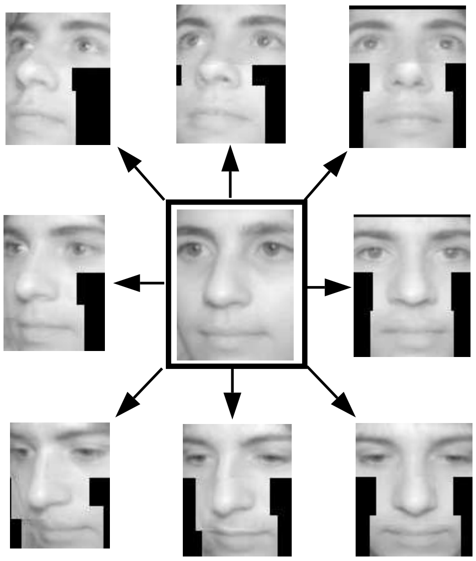


Figure 15: Example virtual views for linear classes.

After feature detection, the input is repetitively matched against all example views of all people. Matching the input against a particular example view consists of two steps, a geometrical registration step and correlation. In the registration step, first an affine transform is applied to the input to bring the iris and nose lobe features into correspondence with the same points on the example view. While this brings the two views into coarse alignment, small pose or expressional differences may remain. To bring the input and example into closer correspondence, optical flow is computed between the two and a 2D warp driven by the flow brings the two into pixelwise correspondence. Lastly, normalized correlation with example templates of the eyes, nose, and mouth is used to evaluate the match. The best match from the data base is reported as the identified person.

6.2 Recognition results

To test the recognizer, a set of 10 testing views per person were taken to randomly sample poses within the overall range of poses in Fig. 3. Roughly half of the test views include an image-plane rotation, so all three rotational degrees of freedom are tested. There are 62 people in the database, including 44 males and 18 females, people from different races, and an age range from the 20s to the 40s. Lighting for all views is frontal and facial expression is neutral.

Table 1 shows recognition rates for parallel deformation for the different prototypes and for manual vs. automatic features. As with the experiments with real views in Beymer [10], the recognition rates were recorded for a forced choice scenario – the recognizer always reports the best match. In the template-based recognizer, template scale was fixed at an intermediate scale (interocular distance = 30 pixels) and preprocessing was fixed at dx+dy (the sum of separate correlations on the x and y components of the gradient). These parameters had yielded the best recognition rates for real views in Beymer [10]. The results were fairly consistent, with a mean recognition rate of 84.1% and a standard deviation of only 0.6%. Automatic feature correspondence on average was as good as the manual correspondences, which was a good result for the face vectorizer. In the manual case, though, it is important to note that the manual step is at "model-building" time; the face recognizer at run time is still completely automatic.

interperson correspondence	prototype		
	A	B	C
manual	84.5%	83.9%	83.9%
auto	85.2%	84.0%	83.4%

Table 1: Recognition rates for parallel deformation for the different prototypes and for manual vs. automatic features.

Fig. 16 summarizes our experiments with using real and virtual views in the recognizer. Starting on the right, we repeat the result from Beymer [10] where we use 15 real views per person. This recognition rate of 98.7% presents a "best case" scenario for virtual views. The real views case is followed by parallel deformation, which gives a recognition rate of 85.2% for prototype A and automatic interperson correspondences. Next, linear classes on texture yields a recognition rate of 73.5%. To put these two recognition numbers in context, we compare them to a "base" case that uses only two example views per person, the real view **m4** plus its mirror reflection. A recognition rate of 70% was obtained for this two view case, thus establishing a lower bound for virtual views. Parallel deformation at 85% falls midway between the benchmark cases of 70% (one view + mirror reflect.) and 98%, (15 views) so it shows that virtual views do benefit pose-invariant face recognition.

In addition, the leftmost bar in Fig. 16 (one view) gives the recognition rate when only the view **m4** is used. This shows how much using mirror reflection helps in the single real view case: without the view generated by mirror reflection, the recognition rate is roughly cut in half from 70% to 32%. This low recognition rate is caused by winnowing of example views based on the coarse pose estimate (looking left vs. looking right) of the input. If the input view is "looking right", then the system does not even try to match against the **m4** example view, which is "looking left". In this (one view) case, 62% of the inputs are rejected, and 6% of the inputs give rise to substitution errors.

Linear classes for virtual texture was a disappointment, however, only yielding a recognition rate a few percentage points higher than the base case of 70%. This may have been due to the factoring out of shape information. We also noticed that the linear reconstruction has a "smoothing" effect, reproducing the lower frequency components of the face better than the higher frequency ones. One difference in the experimental test conditions with respect to parallel deformation was that correlation was performed on the original grey levels instead of dx+dy; empirically we obtained much worse performance after applying a differential operator.

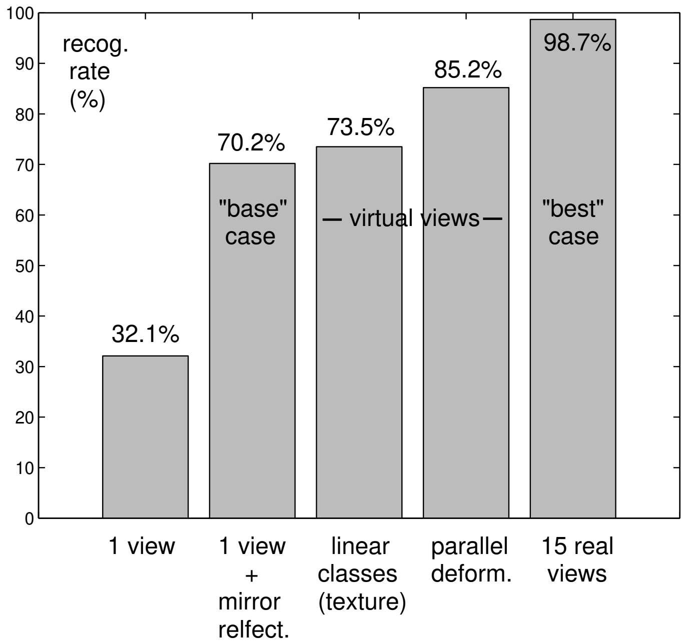


Figure 16: Face recognition performance for real and virtual views.

7 Discussion

7.1 Evaluation of recognition rate

While the recognition rate using virtual views, ranging from 85% for parallel deformation to 73% for linear classes, is much lower than the 98% rate for the multiple views case, this was expected since virtual views use much less information. One way to evaluate these rates is to use human performance as a benchmark. To test human performance, one would provide a subject with a set of training images of previously unknown people, using only one image per person. After studying the training images, the subject would be asked to identify new images of the people under a variety of poses. Moses, Ullman, and Edelman [32] have performed this experiment using testing views at a variety of poses and lighting conditions. While high recognition rates were observed in the subjects (97%), the subjects were only asked to discriminate between three different people. Bruce [12] performs a similar experiment where the subject is asked whether a face had appeared during training, and detection rates go down to either 76% or 60%, depending on the amount of pose/expression difference between the testing and training views. Schyns and Bülthoff [41] obtain a low recognition rate, but their results are difficult to compare since their stimuli are Gouraud shaded 3D faces that exclude texture information. Lando and Edelman [26] have recently performed computational experiments to replicate earlier psychophysical results in [32]. A recognition rate of only 76% was reported, but the authors suggest that this may be improved by using a two-stage classifier instead of a single-stage one.

Direct comparison of our results to related face recognition systems is difficult because of differences in example and testing views. The closest systems are those of Lando and Edelman [26] and Maurer and von der Malsburg [31]. Both systems explore a view transformation method that effectively generates new views from a single view. The view representation, in contrast to our template-based approach, is feature-based: Lando and Edelman use difference of Gaussian features, and Maurer and von der Malsburg use a set of Gabor filters at a variety of scales and rotations (called "jets"). The prior knowledge Lando and Edelman used to transform faces is similar to ours, views of prototype faces at standard and virtual views. They average the transformation in feature space over the prototypes and apply this average transformation to a novel object to produce a "virtual" set of features. As mentioned above, they report a recognition rate of 76%. Maurer and von der Malsburg transform their Gabor jet features by approximating the facial surface at each feature point as a plane and then estimating how the Gabor jet changes as the plane rotates in 3D. They apply this technique to rotating faces about 45° between frontal and half-profile views. They report a recognition rate of 53% on a subset of 90 people from the FERET database.

Two other comparable results are from Manjunath, *et al.* [30], who obtain 86% on a database of 86 people, and Pentland, *et al.* [34], whose extrapolation experiment with view-based eigenspaces yields 83% on a database of 21 people. In both cases, the system is trained on a set of views (vs. just one for ours) and recognition performance is tested on views from outside the pose-expression space of the training set. One difference in example views is that they include hair and we do not. In the future, the new Army FERET database should provide a common benchmark for comparing recognition algorithms.

7.2 Difficulties with virtual views generation

Since we know that the view-based approach performs well with real example views, making the virtual views closer in appearance to the "true" rotated views would obviously improve recognition performance. What difficulties do we encounter in generating "true" virtual views? First, the parallel deformation approach for shape essentially approximates the 3D shape of the novel person with the 3D shape of the prototype. If the two 3D shapes are different, the virtual view will not be "true" even though it may still appear to be a valid face. The resulting shape is a mixture of the novel and prototype shapes. Using multiple prototypes and the linear class approach may provide a better shape approximation.

In addition, for parallel deformation we have problems with areas that are visible in the virtual view but not in the standard view. For example, for the **m4** pose, the underside of the nose is often not visible. How can one predict how that region appears for upward looking virtual views? Possible ways to address this problem include using additional real views or having the recognizer exclude those regions during matching.

7.3 Transformations besides rotation

While the theory and recognition experiments in this paper revolve around generating rotated virtual views, one may also wish to generate virtual views for different lighting conditions or expressions. This would be useful for building a view-based face recognizer that handles those kinds of variation in the input. Here we suggest ways to generate these views.

7.3.1 Lighting

For changes in lighting conditions, the prototype faces are fixed in pose but the position of the light source is changed between the standard and virtual views. Unfortunately, changing the direction of the light source violates an assumption made for linear classes that the lighting conditions are fixed. That assumption had allowed us to ignore the fact that surface albedo and the local surface normal are confounded in the Lambertian model for image intensity.

However, the idea of parallel deformation can still be applied. Parallel deformation assumes that the 3D shape of the prototype is similar to the 3D shape of the novel person. Thus, corresponding points on the two faces should have the same local surface normal. The following analysis focuses on the image brightness of the same feature point on both the prototype and novel face. The two feature points may have been brought into correspondence through a vectorization procedure. Let

$$
\begin{aligned}
\eta &= \text{surface normal for both the prototype} \\
& \text{and novel faces} \\
l_{std} &= \text{light source direction for standard lighting} \\
l_{virtual} &= \text{light source direction for virtual lighting} \\
\rho_{proto} &= \text{albedo for the prototype face} \\
\rho_{nov} &= \text{albedo for the novel face}
\end{aligned}
$$

The prior knowledge of the lighting transformation can be represented by the ratio of the prototype image intensities under the two lighting directions

$$\frac{\rho_{proto}(\eta \cdot l_{virtual})}{\rho_{proto}(\eta \cdot l_{std})}.$$

Simply by multiplying by the image intensity of the novel person $\rho_{nov}(\eta \cdot l_{std})$ and cancelling terms, one can get

$$\rho_{nov}(\eta \cdot l_{virtual}),$$

which is the image intensity of the novel feature point under the virtual lighting. Overall, the novel face texture is modulated by the changes in the prototype lighting, an approach that has been explored by Brunelli [13].

7.3.2 Expression

In this case, the prototypes are fixed in pose and lighting but differ in expression, with the standard view being, say, a neutral expression and the virtual view being a smile, frown, etc. When generating virtual views, we need to capture both nonrigid shape deformations and the subtle texture changes such as the darkening effect of dimples or winkles. Thus, virtual views generation techniques for both shape and texture are required.

Predicting virtual expressions, however, seems more difficult than the rotation or lighting case. This is because the way a person smiles or frowns is probably decoupled from how to decompose his neutral face as a linear combination of the prototypes. To the extent that they are decoupled, the approaches we have suggested for generating virtual shapes and textures will be an approximation. Our problems show up mathematically in the nonrigidness of the transformation; the linear class idea for shape assumes a rigid 3D transform. The implication of these problems is that the expense of multiple prototypes is probably not justified; one is probably better off using just one or a few prototypes. In earlier work aimed primarily at computer graphics [8], we demonstrated parallel deformation for transformations from neutral to smiling expressions.

7.4 Future work

For future work on our approach to virtual views, we plan to use multiple prototypes for generating virtual shape. Vetter and Poggio [51] have already done some work in applying the linear class idea to both shape and texture. It would be interesting to test some of their virtual views in a view-based recognizer. In the longer term, one can test the virtual views technique for face recognition under different lighting conditions or expressions.

For the problem of recognizing faces from just one example view, it should be possible to use the idea of linear classes without actually synthesizing virtual views. The basic idea is to compare faces based on sets of (α_j, β_j) coefficients from equations (5) and (9) rather than using correlation in an image space. According to linear classes, the (α_j, β_j) decomposition for a specific individual should be invariant to pose. As explained in section 4.1, linear classes is based on the assumption that the 3D shape vector of the input $\mathbf{Y}$ and the 3D texture vector $\mathbf{T}$ are linear combinations of the shapes and textures of prototype faces. Under certain conditions, the linear coefficients (α_j, β_j) of the 3D decomposition are computable from an arbitrary 2D view. Thus, the coefficients should be invariant to pose since they are derived from a 3D representation. It follows that the (α_j, β_j) coefficients should themselves be an effective representation for faces. The coefficients of the unidentified input view $(\alpha_j, \beta_j)^{input}$ can be directly matched against the data base coefficients of each person at standard pose $(\alpha_j, \beta_j)^{std}$. Note that the linear coefficients are not a true invariant because the recognizer at run-time needs to have an estimate of the out-of-plane image rotation of the input.

8 Conclusion

In this paper we have addressed the problem of recognizing faces under different poses when only one example view of each person is available. Given one real view at a known pose, we use prior knowledge of faces to generate *virtual views*, views of the face as seen from different poses. Rather than using a more traditional 3D modeling approach, prior knowledge of faces is expressed in the form of 2D views of rotating prototype faces. Given the 2D prototype views and a single real view of a novel person, we demonstrated two techniques for effectively rotating the novel face in depth. First, in parallel deformation, a facial transformation observed on a prototype face in mapped onto a novel face and used to warp the novel view. Second, in linear classes, the single novel view is decomposed as a linear combination of prototype views at the same pose. Then these same linear coefficients are used to synthesize a virtual view of the novel person by taking a linear combination of the prototype views at virtual pose. We demonstrated this for the grey level, or textural, component of the face.

To evaluate virtual views, they were then used as example views in a view-based, pose-invariant face recognizer. On a database of 62 people with 10 test views per person, a recognition rate of 85% was achieved in experiments with parallel deformation, which is well above the base recognition rate of 70% when only one real view (plus its mirror reflection) is used. Also, our recognition rate is similar to other face recognition experiments where extrapolation from the pose-expression range of the example views is tested. Overall, for the problem of generating new views of an object from just one view, these results demonstrate that the 2D example-based technique, similarly to 3D object models, may be a viable method for representing knowledge of object classes.

References

[1] Andrew C. Aitchison and Ian Craw. Synthetic images of faces – an approach to model-based face recognition. In *Proc. British Machine Vision Conference*, pages 226–232, 1991.

[2] 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.

[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] Robert J. Baron. Mechanisms of human facial recognition. *International Journal of Man Machine Studies*, 15:137–178, 1981.

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

[6] 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.

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

[8] 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.

[9] David Beymer. Vectorizing face images by interleaving shape and texture computations. A.I. Memo No. 1537, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1995.

[10] David J. Beymer. Face recognition under varying pose. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 756–761, Seattle, WA, 1994.

[11] Martin Bichsel. *Strategies of Robust Object Recognition for the Automatic Identification of Human Faces*. PhD thesis, ETH, Zurich, 1991.

[12] Vicki Bruce. Changing faces: Visual and non-visual coding processes in face recognition. *British Journal of Psychology*, 73:105–116, 1982.

[13] Roberto Brunelli. Estimation of pose and illuminant direction for face processing. A.I. Memo No. 1499, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1994.

[14] Roberto Brunelli and Tomaso Poggio. Face recognition: Features versus templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(10):1042–1052, 1993.

[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, 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.

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

[18] 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.

[19] 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.

[20] Jeffrey M. Gilbert and Woody Yang. A real-time face recognition system using custom VLSI hardware. In *IEEE Workshop on Computer Architectures for Machine Perception*, pages 58–66, December 1993.

[21] F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7:219–269, 1995.

[22] Peter W. Hallinan. A low-dimensional representation of human faces for arbitrary lighting conditions. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 995–999, Seattle, WA, 1994.

[23] 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.

[24] Chii-Yuan Kang, Yung-Sheng Chen, and Wen-Hsing Hsu. Mapping a lifelike 2.5D human face via an automatic approach. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 611–612, New York, NY, June 1993.

[25] Martin Lades, Jan C. Vorbruggen, Joachim Buhmann, Jorg Lange, Christoph v.d. Malsburg, Rolf P. Wurtz, and Wolfgang Konen. Distortion invariant object recognition in the dynamic link architecture. *IEEE Transactions on Computers*, 42(3), March 1993.

[26] Maria Lando and Shimon Edelman. Generalization from a single view in face recognition. In *Proceedings, International Workshop on Automatic Face- and Gesture-Recognition*, pages 80–85, Zurich, 1995.

[27] 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.

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

[29] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings IJCAI*, pages 674–679, Vancouver, 1981.

[30] B.S. Manjunath, R. Chellappa, and C. von der Malsburg. A feature based approach to face recognition. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 373–378, 1992.

[31] Thomas Maurer and Christoph von der Malsburg. Single-view based recognition of faces rotated in depth. In *Proceedings, International Workshop on Automatic Face- and Gesture-Recognition*, pages 248–253, Zurich, 1995.

[32] Yael Moses, Shimon Ullman, and Shimon Edelman. Generalization to novel images in upright and inverted faces. Technical Report CC93-14, The Weizmann Institute of Science, 1993.

[33] Elizabeth C. Patterson, Peter C. Litwinowicz, and Ned Greene. Facial animation by spatial mapping. In *Computer Animation '91*, pages 31–44. Springer-Verlag, Tokyo, 1991.

[34] 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.

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

[36] T. Poggio. 3D object recognition and prototypes: one 2D view may be sufficient. Technical Report 9107–02, I.R.S.T., Povo, Italy, July 1991.

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

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

[39] 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.

[40] 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.

[41] Phillipe G. Schyns and Heinrich H. Bülthoff. Conditions for viewpoint invariant face recognition. A.I. Memo No. 1432, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1993.

[42] 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.

[43] A. Shashua. Correspondence and affine shape from two orthographic views: Motion and Recognition. A.I. Memo No. 1327, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, December 1991.

[44] A. Shashua. *Geometry and Photometry in 3D visual recognition.* PhD thesis, M.I.T Artificial Intelligence Laboratory, AI-TR-1401, November 1992.

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

[46] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[47] 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.

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

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

[50] Thomas Vetter, Anya Hurlbert, and Tomaso Poggio. View-based models of 3D object recognition: Invariance to imaging transformations. *Cerebral Cortex*, 3:261–269, May/June 1995.

[51] Thomas Vetter and Tomaso Poggio. Linear object classes and image synthesis from a single example

image. A.I. Memo No. 1531, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1995.

[52] Lance Williams. Performance-driven facial animation. In *SIGGRAPH '90 Proceedings*, pages 235–242, Dallas, TX, August 1990.

[53] Lance Williams. Living pictures. In *Models and Techniques in Computer Animation*, pages 2–12. Springer-Verlag, Tokyo, 1993.

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

[55] Yaser Yacoob and Larry Davis. Computing spatio-temporal representations of human faces. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 70–75, Seattle, WA, 1994.

A Appendix: how general is the linear class assumption?

Following the basic suggestion in Poggio (1991) and Poggio and Vetter (1992) we consider the problem of creating virtual views within the framework of learning-from-examples techniques. In this metaphor, a learning module such as a Regularization Network is trained with a set of input-output examples of objects of the same "nice" class (see Vetter et al., 1995) to learn a certain class-specific transformation. For a rotation transformation "frontal" views are associated with the "rotated" views of the same faces. For the "smile" transformation "serious" views are associated with "smiling" views. One can think of the frontal view as the input to the network and the rotated view as the corresponding output during the training phase.

One may be inclined to believe that this approach may be more powerful than the simple linear technique described in the the text and justified under the linear class assumption of Poggio and Vetter. Though this is true, the behavior of a generic network trained as described above is still severely restricted. It turns out that under rather general conditions the output of a very large class of learning-from-examples techniques produces virtual views that are contained in the linear space spanned by the (output) examples.

One way to express the result is the following. *If the transformed view of an object cannot be represented as a linear combination of transformed views of prototypes (in smaller number than view dimensionality) then no learning module (under quite weak conditions) can learn that transformations from example pairs.*

The basic result is implied by Girosi, Jones and Poggio (1995) and by Beymer, Shashua and Poggio (1993). We reproduce it here for completeness.

The simplest version of a regularization network approximates a vector field $\mathbf{y}(\mathbf{x})$ as

$$\mathbf{y}(\mathbf{x}) = \sum_{i=1}^{N} \mathbf{c}_i G(\mathbf{x} - \mathbf{x}_i) \tag{15}$$

which we rewrite in matrix terms as

$$\mathbf{y}(\mathbf{x}) = \mathbf{C}\mathbf{g}(\mathbf{x}), \tag{16}$$

where $\mathbf{g}$ is the vector with elements $g_i = G(\mathbf{x} - \mathbf{x}_i)$. Defining as $\mathbf{G}$ is the matrix of the chosen basis function ($G_{i,j} = G(\mathbf{x}_i - \mathbf{x}_j)$) evaluated at the examples we obtain

$$(G)\mathbf{c}_m = \mathbf{y}_m \tag{17}$$

and also

$$\mathbf{C} = \mathbf{Y}\mathbf{G}^{\dagger}. \tag{18}$$

It follows that the vector field is approximated as the linear combination of example fields, that is

$$\mathbf{y}(\mathbf{x}) = \mathbf{Y}\mathbf{G}^{\dagger}\mathbf{g}(\mathbf{x}) \tag{19}$$

that is

$$\mathbf{y}(\mathbf{x}) = \sum_{l=1}^{N} \mathbf{b}_l(\mathbf{x})\mathbf{y}_l, \tag{20}$$

where the $\mathbf{b}_l$ depend on the chosen G, according to

$$\mathbf{b}(\mathbf{x}) = \mathbf{Y}\mathbf{G}^{\dagger}\mathbf{g}(\mathbf{x}). \tag{21}$$

Thus *for any choice of the regularization network* the output (vector) image is always a linear combination of example (vector) images with coefficients $\mathbf{b}$ that depend (nonlinearly) on the desired input value. This is true for a large class of networks trained with the L^2 error criterion, including many types of Neural Networks (the observation is by F. Girosi).

Thus, the virtual views that can be generated by a large class of learning techniques are always contained in the linear subspace of the examples. This non trivial observation *means that the latter property is rather general and does not depend on the linear class assumption of Poggio and Vetter.*

Notice that for these results to hold we assume correspondence between all input vectors and all output vectors separately. Strictly speaking correspondence is not needed between input and output vectors (this observation is due to T. Vetter).

Let us define separately the shape component of our images and the texture components. The shape is a vector $\mathbf{y}$ of all $x_1, y_1, \cdots, x_n, y_n$ describing the image position of each of n pixels. We consider separately the vector of corresponding textures for the same n pixels as $\mathbf{t}$ consisting of the grey values $I_1, \cdots, I_n$. We could also consider the extended vector $\mathbf{E}$ obtained by concatenating $\mathbf{y}$ and $\mathbf{t}$. The previous result show that that the output rotated shape image obtained for a new frontal image $\mathbf{y}_r$ is in the linear space spanned by the examples. Thus

$$\mathbf{y}_r = \sum c_i \mathbf{y}_r^i. \tag{22}$$

Of course in general the c are not identical to the coefficients of the input representation (as in the linear class case described in the text) and will be a nonlinear function of them. In the linear class case the c can be learned by a simple linear network without hidden layers (see main text).

Since the linear class argument can also be extended to the texture component $\mathbf{t}$ (see main text and Vetter and Poggio, 1995), the same observations stated here for shape can also be applied to texture.

One could impose linear class conditions on $\mathbf{E}$: the dimensionality of the space will however be quite larger. In general, one should keep separate the shape and the texture components and to span them with independent basis.

Notice (see also Beymer, Poggio and Shashua, 1993) that in many situations it may be advantageous to transform the output representation from

$$\mathbf{y}_r = \sum_n c_i \mathbf{y}_r^i \tag{23}$$

to

$$\mathbf{y}_r = \sum_q c_i^* \mathbf{y^*}_r^i, \tag{24}$$

where the $\mathbf{y}^*$ are the basis of a KL decomposition and $q << n$.

B Appendix: linear classes

As explained in section 4.1, linear classes is a technique for synthesizing new views of an object using views of prototypical objects belonging to the same object class. The basic idea is to decompose the novel object as a linear combination of the prototype objects. This decomposition is performed separately for the shape and texture of the novel object. In this appendix, we explain the mathematical detail behind the linear class approach for shape and texture. Please refer to sections 3 and 4.1 for definitions of the example prototype images, mathematical operators, etc.

B.1 Shape (Poggio and Vetter, 1992)

In this section, we reformulate the description of linear classes for shape that originally appeared in Poggio and Vetter [39]. The development here makes explicit the fact that the vectorized $\mathbf{y}$ vectors need not be in correspondence between the standard and virtual poses.

Linear classes begins with the assumption that a novel object is a linear combination of a set of prototype objects in 3D

$$\mathbf{Y}_n = \textstyle\sum_{j=1}^N \alpha_j \mathbf{Y}_{p_j}. \tag{25}$$

From this assumption, it is easy to see that any 2D view of the novel object will be the same linear combination of the corresponding 2D views of the prototypes. That is, the 3D linear decomposition is the same as the 2D linear decomposition. Using equation (2) which relates 3D and 2D shape vectors, let $\mathbf{y}_{n,r}$ be a 2D view of a novel object

$$\mathbf{y}_{n,r} = L\mathbf{Y}_n \tag{26}$$

and let $\mathbf{y}_{p_j,r}$ be 2D views of the prototypes

$$\mathbf{y}_{p_j,r} = L\mathbf{Y}_{p_j} \qquad 1 \le j \le N. \tag{27}$$

Apply the operator L to both sides of equation (25)

$$L\mathbf{Y}_n = L(\textstyle\sum_{j=1}^N \alpha_j \mathbf{Y}_{p_j}). \tag{28}$$

We can bring L inside the sum since L is linear

$$L\mathbf{Y}_n = \textstyle\sum_{j=1}^N \alpha_j L\mathbf{Y}_{p_j}. \tag{29}$$

Substituting equations (26) and (27) yields

$$\mathbf{y}_{n,r} = \textstyle\sum_{j=1}^N \alpha_j \mathbf{y}_{p_j,r}.$$

Thus, the 2D linear decomposition uses the same set of linear coefficients as with the 3D vectorization.

Next, we show that under certain assumptions, the novel object can be analyzed at standard pose and the virtual view synthesized at virtual pose using a single set of linear coefficients. Again, assume that a novel object is a linear combination of a set of prototype objects in 3D

$$\mathbf{Y}_n = \textstyle\sum_{j=1}^N \alpha_j \mathbf{Y}_{p_j}. \tag{30}$$

Say that we have 2D views of the prototypes at standard pose $\mathbf{y}_{p_j}$, 2D views of the prototypes at virtual pose $\mathbf{y}_{p_j,r}$, and a 2D view of the novel object $\mathbf{y}_n$ at standard pose. Additionally, assume that the 2D views $\mathbf{y}_{p_j}$ are linearly independent. Project both sides of equation (30) using the rotation for standard pose, yielding

$$\mathbf{y}_n = \textstyle\sum_{j=1}^N \alpha_j \mathbf{y}_{p_j}.$$

A unique solution for the α_j exist since the $\mathbf{y}_{p_j}$ are linearly independent. Now, since we have solved for the same set of coefficients in the 3D linear class assumption, the decomposition at virtual pose must use the same coefficients

$$\mathbf{y}_{n,r} = \textstyle\sum_{j=1}^N \alpha_j \mathbf{y}_{p_j,r}.$$

That is, we can recover the α_j's from the view at standard pose and use the α_j's to generate the virtual view of the novel object.

B.2 Texture

Virtually the same argument can be applied to the geometrically normalized texture vectors $\mathbf{t}$. The idea of applying linear classes to texture was thought of by the authors and independently by Vetter and Poggio [51].

With the texture case, assume that a novel object texture $\mathbf{T}_n$ is a linear combination of a set of prototype textures

$$\mathbf{T}_n = \textstyle\sum_{j=1}^N \beta_j \mathbf{T}_{p_j}. \tag{31}$$

As with shape, we show that the 3D linear decomposition is the same as the 2D linear decomposition. Using equation (3) which relates 3D and 2D texture vectors, let $\mathbf{t}_{n,r}$ be a 2D texture of a novel object

$$\mathbf{t}_{n,r} = D\mathbf{T}_n \tag{32}$$

and let $\mathbf{t}_{p_j,r}$ be 2D textures of the prototypes

$$\mathbf{t}_{p_j,r} = D\mathbf{T}_{p_j} \qquad 1 \le j \le N. \tag{33}$$

Apply the operator D to both sides of equation (31)

$$D\mathbf{T}_n = D(\textstyle\sum_{j=1}^N \beta_j \mathbf{T}_{p_j}). \tag{34}$$

We can bring D inside the sum since D is linear

$$D\mathbf{T}_n = \textstyle\sum_{j=1}^N \beta_j D\mathbf{T}_{p_j}. \tag{35}$$

Substituting equations (32) and (33) yields

$$\mathbf{t}_{n,r} = \sum_{j=1}^{N} \beta_j \mathbf{t}_{p_j,r}.$$

Thus, as with shape, the 2D linear decomposition for texture uses the same set of linear coefficients as with the 3D vectorization.

Next, we show that under certain linear independence assumptions, the novel object texture can be analyzed at standard pose and the virtual view synthesized at virtual pose using a single set of linear coefficients. Again, assume that a novel object texture $\mathbf{T}$ is a linear combination of a set of prototype objects

$$\mathbf{T}_n = \sum_{j=1}^{N} \beta_j \mathbf{T}_{p_j}. \tag{36}$$

Say that we have 2D textures of the prototypes at standard pose $\mathbf{t}_{p_j}$, the 2D prototype textures at virtual pose $\mathbf{t}_{p_j,r}$, and a 2D texture of the novel object at standard pose $\mathbf{t}_n$. Additionally, assume that the 2D textures $\mathbf{t}_{p_j}$ are linearly independent. Project both sides of equation (36) using the rotation for standard pose, yielding

$$\mathbf{t}_n = \sum_{j=1}^{N} \beta_j \mathbf{t}_{p_j}.$$

A unique solution for the β_j exist since the $\mathbf{t}_{p_j}$ are linearly independent. Now, since we have solved for the same set of coefficients in the 3D linear class assumption, the decomposition at virtual pose must use the same coefficients

$$\mathbf{t}_{n,r} = \sum_{j=1}^{N} \beta_j \mathbf{t}_{p_j,r}.$$

That is, we can recover the β_j's from the view at standard pose and use the β_j's to generate the virtual view of the novel object.