MIT Media Laboratory Perceptual Computing Section Technical Report 422
Also appears as MIT thesis for the degree of
Bachelor of Science in Electrical Engineering and Computer Science
Supervised by Alex P. Pentland

Eigenheads for Reconstruction

Kenneth B. Russell
Vision and Modeling Group
MIT Media Laboratory
20 Ames Street; Cambridge, MA 02139
kbrussel@media.mit.edu

Abstract

A framework is presented for recovering the 3D structure and visual appearance of a human head from sparse data obtained from a real-time tracking system. An eigenvector decomposition of CyberWare-scanned heads is used to code incoming information. Modular eigenspaces are used to decorrelate eigenfeatures (eyes, nose, and mouth) from the rest of the head data. We observe that the modular eigenspace encoding often does not perform as well as a single eigenspace, and offer reasons for this based on experimental evidence.

This work was supported in part by British Telecom and Texas Instruments.

Acknowledgments

I thank my parents for their constant love and support, and for giving me the opportunity to attend MIT.

Thanks to Tony Jebara and Prof. Sandy Pentland for supervising this work.

I am deeply indebted and offer sincere thanks to Tom Minka, for his unfailing help and advice, for helping me to understand the optimal linear estimator, for showing me how to appreciate the research process, for reviewing this document, and in general for being the world's most patient mentor.

Chapter 1

Introduction

1.1 Abstract

This document presents a framework for recovering the 3D structure and visual appearance of a human head from sparse data obtained from a real-time tracking system. An eigenvector decomposition of CyberWare-scanned heads is used to code incoming information. Modular eigenspaces are used to decorrelate eigenfeatures (eyes, nose, and mouth) from the rest of the head data. We observe that the modular eigenspace encoding often does not perform as well as a single eigenspace, and offer reasons for this based on experimental evidence.

1.2 Description of the Project

The primary goal of this project was to create a system which, when presented with examples of 3D models of human heads, could then take incomplete data from a head tracking system (such as that described in [1]) and produce a new 3D model which best approximates the user's head.

The 3D head models used as examples were obtained using CyberWare scanners (see Figure 1.1 for an example). These scanners work by revolving an arm containing a laser range finder 360 degrees around a subject's head, measuring the distance to the head, as well as the color, at many points (typically 512 measurements vertically and longitudinally). The output is a cylindrical coordinate depth map, as well as an associated texture map, of the user's head; these can be combined to reconstruct a 3D model. When viewed as 2D images instead of 3D models, these scans look like an "unwrapped" version of the head.

FLIRT [1] (Face Localization for Invariant Recognition and Tracking) uses a single video camera connected to a Silicon Graphics Indy workstation to find certain facial features such as the eyes and corners of the mouth. The three-dimensional alignment and (sparse) structure of the head is estimated using a

Figure 1.1: A computer model generated with a CyberWare scanner. The first two images are the range and texture data, respectively.

Kalman filter as described in [6]. Using this positional information, an "average head" model is aligned to the user's head and the video image of the user is projected onto the model. The texture corresponding to the face region is then unwrapped to create a sparse image with the same "squashed" appearance as in the example CyberWare scans. This data is used as input to the reconstruction system.

An eigenvector decomposition of the example heads was used as the coding mechanism for the reconstruction pipeline. This technique is efficient; encoding an incoming head requires just taking the dot product with the eigenvectors. It has also been shown to be effective for both recognition ([2], [3]) and reconstruction ([4]). Modular eigenspaces were used to semi-independently code regions of the face such as the eyes, nose, and mouth at higher resolution. A linear estimator based on the example heads was used to fill in the data which the tracking system could not supply.

1.3 Prior Work

Principal component analysis was proposed for the problem of face recognition in [2]. Modular approaches to the eigenface technique were described in [3]. A technique for recovering 3D information from a 2D image, using a statistical model built from scans of peoples' heads, was published in [4].

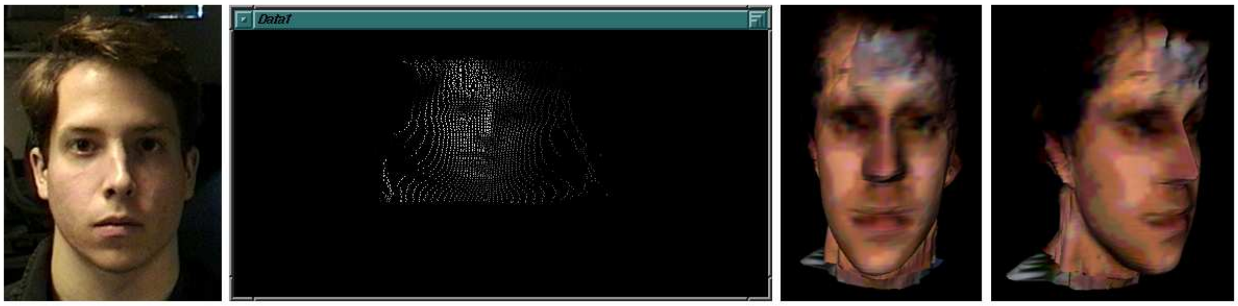

Figure 1.2: The run-time reconstruction sequence. FLIRT captures the video image and the data corresponding to the face region is unmapped into cylindrical coordinates, creating a sparse texture map. Two views of the reconstructed model from this data are shown.

1.4 Contributions

The contributions of this work are twofold. First, this system may be unique in using modular eigenspaces for reconstruction of head models. Second, rather than operating on photographs as in [4], it uses incomplete data from a real-time tracking system as input to the reconstruction mechanism.

1.5 Organization

The rest of this document is organized as follows. Chapter 2 describes the precise pipeline, both preprocessing and runtime, of the reconstruction mechanism. Chapter 3 describes the 3D head alignment and resampling algorithms. Chapter 4 derives the optimal linear estimator used to fill in missing data from the tracker. Chapter 5 describes the mathematical formulation of the modular eigenspace technique. Chapter 6 details the experiments used to test the system. Finally, Chapter 7 draws conclusions based on experiences with the system. Descriptions of most of the software written for the reconstruction mechanism are given in Appendix A.

Chapter 2

Description of the reconstruction pipeline

Most of the work done in the reconstruction pipeline is preprocessing to correct the alignment of the example heads, build the eigenspaces, and create the linear estimators to fill in data not obtained from FLIRT. Each step in the pipeline is detailed below, and relevant programs from the software catalog (Appendix A) are listed. The sequence of operations is summarized in Figure 2.1.

1. Head models are aligned in 3D (and, in FLIRT's version of the alignment code, color corrected) so that eyes, nose, and mouth line up as best as possible. This eliminates the need to encode information about the orientation of the user's head in the eigenspace. The merits of this "view-based" approach are discussed in [3]. (**align, alignToFLIRT**)

2. The aligned models are converted back into cylindrical-coordinate range and texture files, which now have the property that the eyes, nose, and mouth of each of the heads are in approximately the same 2D location in all of the data files. (**iv2float**)

3. The range map of the canonical head (the one to which the other models were aligned) is used as the basis to create masks for the modular eigenspaces — see Figure 2.2. These masks highlight separately the eyes, nose, mouth, and hair of the canonical head. In addition, for comparison, a mask highlighting the entire head region is generated (corresponding to a single, rather than modular, eigenspace). All of these masks crop out information above and below the region of the head, so that the shoulder blades and any invalid information above the top of the head are removed.

4. The masks are smoothed using a diffusion algorithm to eliminate hard boundaries around the highlighted regions, to avoid hard-edge artifacts in

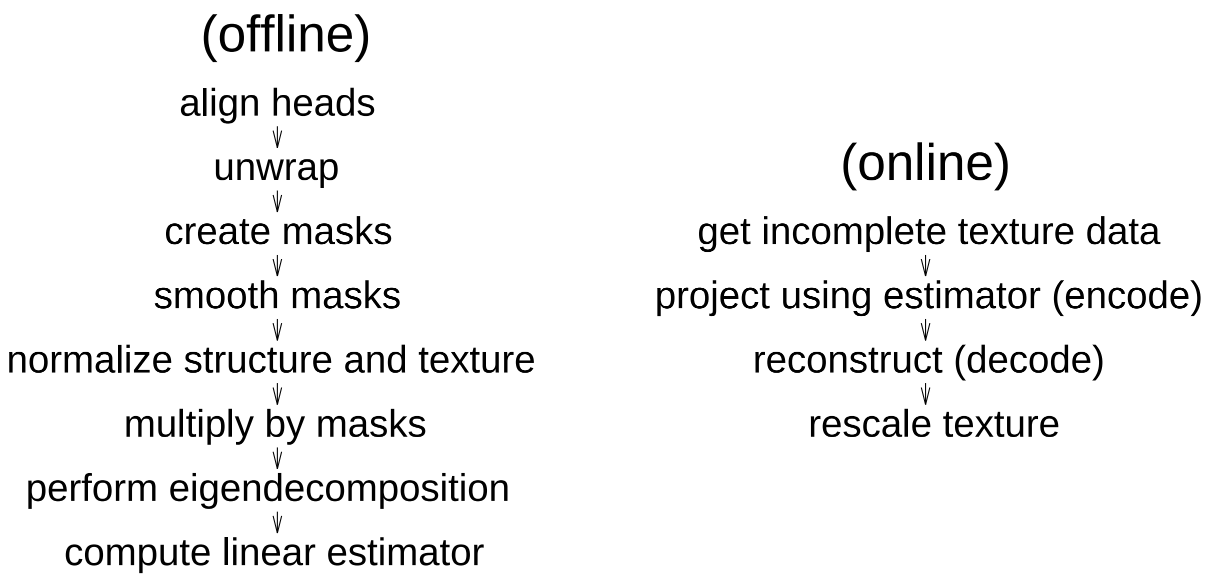

Figure 2.1: Overview of the reconstruction pipeline.

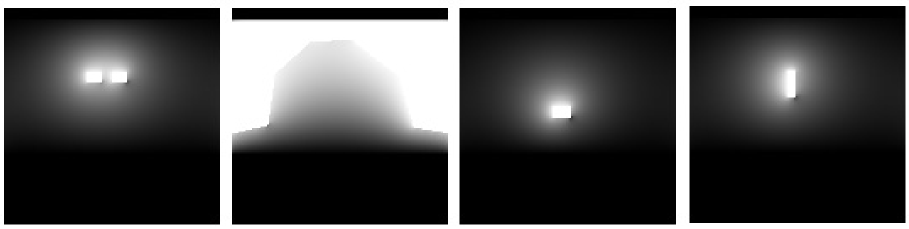

Figure 2.2: The attenuation masks for the eyes, hair, mouth, and nose.

the modular eigenspaces. (**diffuse**)

5. The intensities of the texture components of the aligned head files are scaled down by a dataset-dependent amount, to avoid having the power of the texture component (which is three times as large as the structure component, per file) overwhelm the structure component. In other words, when the eigenspace is presented with some structure information and some texture information for a head, it should weight both equally when representing the head, as opposed to essentially discarding the structural information. (**lambdaCalc, combiner**)

6. Each of the head files is multiplied by each of the masks in turn to create a set of input heads per mask. (**combiner**)

7. An eigenvector decomposition is performed on each mask's head set. These are the modular eigenspaces used for reconstruction. (**eigenDecomp**)

8. For each modular eigenspace, the matrix of eigenvectors is reduced to the dimensions that will be acquired from FLIRT, and the linear estimator computed; see Chapter 4. The result is the matrix used for coding the incoming head data at run time. At this point the (offline) preprocessing stage is completed. (**estimatorCalc**)

9. At run time, the texture image of the user is acquired from FLIRT and its dot product is taken with each of the vectors in each of the modular eigenspaces' estimators. The resulting coefficients are used to weight the full eigenvectors when taking the weighted sum of eigenvectors for reconstruction. The result is divided by the sum of the masks, and the texture component is scaled back up. The output is a head file containing structure and texture information approximating the user's head. (**reconTrack**)

Chapter 3

Alignment and resampling

The original head alignment program from the FLIRT system used one translation, one rotation, and a vertical stretch as its degrees of freedom when aligning the four points (eyes, nose, and mouth) of the input heads to the canonical head. There were two most noticeable problems with the output of this program. First, some output heads' radii were significantly larger than others, causing some heads to look too large. Second, the resampling algorithm in this program (which creates a grid-aligned range map from the rotated data) caused the rotated heads to have quantization artifacts; these appeared as visible creases in the output heads.

To alleviate these problems, a new alignment program was written, which used solely rigid transformations (one translation and one rotation) to align these four points on the heads. The algorithm for this alignment is derived in [5], and can be summarized as follows:

1. Let $\{p_i\}$ $(i = 1 \ldots N)$ be the set of 3D points on the head which we wish to align, and $\{p'_i\}$ be the corresponding points on the canonical head. ($\{p_i\}$ and $\{p'_i\}$ here are $3 \times N$ matrices, with each column a data point.) We wish to find R and T, the rotation and translation which will transform each p_i,

$$Rp_i + T$$

to minimize the least-squared error between $\{p_i\}$ and $\{p'_i\}$,

$$\sum_{i=1}^{N} ||p'_i - (Rp_i + T)||^2$$

2. Subtract off the centroids of the data sets to obtain $\{q_i\}$ and $\{q'_i\}$, respectively. T is calculated as the centroid of $\{p'_i\}$ minus the centroid of $\{p_i\}$.

3. Calculate the 3×3 matrix

$$H \equiv \sum_{i=1}^{N} q_i q_i'^{\mathrm{T}}$$

4. Find the SVD of H, $H = U \Lambda V^{\mathrm{T}}$.

5. Calculate $X = V U^{\mathrm{T}}$.

6. Calculate $\det(X)$, the determinant of X. If $\det(X) = +1$, then $R = X$. (If $\det(X) = -1$, the algorithm fails.)

Once we have rotated the incoming head to align to the canonical head, we must resample it along a "grid" to obtain its cylindrical coordinate depth and texture maps. For this reason a general resampling algorithm was implemented which casts rays from the outside of the model towards the vertical axis — it can be thought of as a "virtual CyberWare scanner". The first version of the algorithm was implemented using the general-purpose ray picking capability of the Open Inventor 3D graphics library, and took approximately 40 minutes to resample a rotated head at 128x128 resolution.

The fact that the rays are all being cast towards the center of the model imposes some constraints on the problem which can be used to improve the performance of the algorithm. The 3D space can be divided up into "bins", with the restriction that no ray may intersect more than one bin. In this case the geometric constraint allows subdivision of space into wedges, that is, by θ and y: see Figure 3.1. By putting the triangles comprising the model into bins, we can restrict the number of triangles with which we need to perform intersection tests when actually executing the ray cast.

In this case, it is easy to determine which bin or bins a particular triangle falls in. The vertical span of the triangle determines the range of bins in the y dimension; its angular span determines the range of bins in the θ dimension. Note that this causes a triangle to be placed in any bin with which it *might* have an intersection; this inaccuracy is acceptable, since it will never cause an intersection to be missed, and speeds up the binning process.

Once all of the triangles of the model have been binned, rays are cast into the model by first determining which bin the ray will intersect, and then intersecting the ray with all triangles in that bin. The intersection point with the largest radius (distance from the vertical axis) is returned as the result of the ray cast. There is a tradeoff between the size of the bins and the speed of the ray casting process; if the bins are made smaller (for example, the number of bins is equal to the number of rays being cast) then the number of triangle intersections required per ray will decrease, but the startup time for the binning process will increase. Informal testing has shown that allowing four rays to intersect with each bin (two in the θ direction and two in the y direction) offers a good compromise.

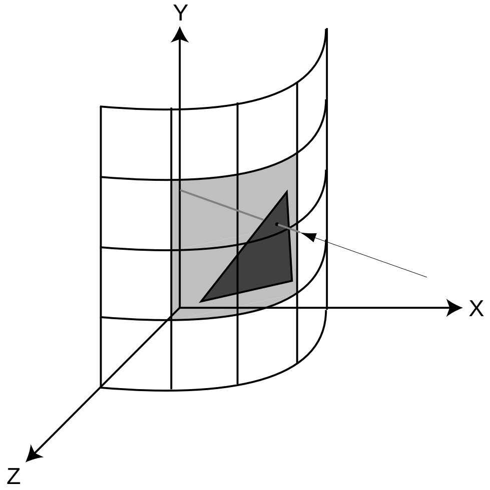

Figure 3.1: The spatial subdivision scheme. The triangle falls into the four shaded bins, which divide space radially. The ray being cast is only intersected with triangles in the bin it intersects.

This binning scheme decreased the running time of the resampling algorithm from 40 minutes to under 30 seconds, a factor of 100 speed increase. The resampled heads are free from the original quantization artifacts.

Tests done with the output heads from the new alignment program showed that it did not line up the eyes and mouth as well as FLIRT's version, because of the lack of a vertical stretch. For this reason, the new resampling algorithm was coupled with the original alignment program (which, in the meantime, had had the "oversized head" problem fixed) to generate the aligned heads for later work.

Chapter 4

Derivation of the optimal linear estimator

The input data coming from FLIRT, which is to be used as the basis for the reconstruction, is incomplete; it contains only some of the dimensions of a complete head vector. In order to project this incomplete vector onto an eigenspace, we must estimate its unknown dimensions. We restrict the estimator to be linear, and here derive the form of the optimal linear estimator. The following notation will be used:

> $\mathbf{x}_o$: observation data (known)
> $\mathbf{x}_m$: missing data (unknown, but we have this in the example scanned heads)
> $\mathbf{x}_c = \begin{bmatrix} \mathbf{x}_o \\ \mathbf{x}_m \end{bmatrix}$: complete input vector (i.e., a head)
> $\mathbf{K}$: covariance matrix of data
> $\mathbf{V}_c \mathbf{\Lambda}_c \mathbf{V}_c^{\mathrm{T}}$: eigenvector decomposition of $\mathbf{K}$
> $\mathbf{K}_{\mathbf{x}_m \mathbf{x}_o}$ or $\mathbf{K}_{mo} = E[\mathbf{x}_m \mathbf{x}_o^{\mathrm{T}}]$: cross-covariance matrix

First, we show that without any assumptions on the probability density function (PDF) of x_o, the minimum mean squared error criterion

$$E[(x_m - \underbrace{f(x_o)}_{\hat{x}_m})^2 | x_o] \text{ minimized}$$

leads to the conditional expectation as the optimal linear estimator. x_o is given (observed). $\hat{x}_m$ is a function of x_o, and is the output of the estimator. We want to find the value of $\hat{x}_m$ which minimizes the squared error over all choices of x_m for a given x_o.

Proof:

$$\frac{dE}{df} = E[-2(x_m - \hat{x}_m)|x_o] = 0$$
$$E[x_m|x_o] - E[\hat{x}_m|x_o] = 0$$
$$\hat{x}_m = E[x_m|x_o]$$

Now we derive the scalar form of the optimal linear estimator for x_m given x_o. We seek a to minimize the new criterion $E[(x_m - ax_o)^2]$.

$$\frac{dE}{da} = E[2(x_m - ax_o)(-x_o)] = 0$$
$$-E[x_m x_o] + aE[x_o^2] = 0$$
$$a = \frac{E[x_m x_o]}{E[x_o^2]}$$

Now we consider the vector case, where $\mathbf{x}_o$ and $\mathbf{x}_m$ are vectors, and $\mathbf{A}$ is a matrix. We seek to estimate the missing part of the vector, $\mathbf{x}_m$, given $\mathbf{x}_o$. The new criterion to be minimized is $E[(\mathbf{x}_m - \mathbf{A}\mathbf{x}_o)^{\mathrm{T}}(\mathbf{x}_m - \mathbf{A}\mathbf{x}_o)]$. Note that $\frac{d}{d\mathbf{A}}[\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{y}] = \mathbf{x}\mathbf{y}^{\mathrm{T}}$.

$$\frac{d}{d\mathbf{A}} E[(\mathbf{x}_m^{\mathrm{T}}\mathbf{x}_m - \mathbf{x}_m^{\mathrm{T}}\mathbf{A}\mathbf{x}_o - \mathbf{x}_o^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{x}_m + \mathbf{x}_o^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x}_o)] = 0 \tag{4.1}$$

First we will simplify the last term using the product rule, grouping the $\mathbf{A}^{\mathrm{T}}$ with $\mathbf{x}_o^{\mathrm{T}}$ and the $\mathbf{A}$ with $\mathbf{x}_o$, using the derivative rule above, and using the identity $\frac{d}{d\mathbf{A}}\mathbf{x}\mathbf{A}^{\mathrm{T}}\mathbf{y} = (\frac{d}{d\mathbf{A}}\mathbf{x}\mathbf{A}\mathbf{y})^{\mathrm{T}}$. Substituting variables,

$$\frac{d}{d\mathbf{A}}[\mathbf{x}_o^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x}_o] = \frac{d}{d\mathbf{A}}[\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x}_o + \mathbf{x}_o^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{z}] \tag{4.2}$$

It is obvious that the first term in Equation 4.2 reduces to $\mathbf{A}\mathbf{x}_o\mathbf{x}_o^{\mathrm{T}}$. Simplifying the second term,

$$\frac{d}{d\mathbf{A}}\mathbf{x}_o^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{z} = (\frac{d}{d\mathbf{A}}\mathbf{x}_o^{\mathrm{T}}\mathbf{A}\mathbf{z})^{\mathrm{T}} = (\mathbf{x}_o\mathbf{z}^{\mathrm{T}})^{\mathrm{T}} = \mathbf{z}\mathbf{x}_o^{\mathrm{T}} = \mathbf{A}\mathbf{x}_o\mathbf{x}_o^{\mathrm{T}} \tag{4.3}$$

Substituting back into Equation 4.1,

$$-2E[\mathbf{x}_m\mathbf{x}_o^{\mathrm{T}}] + 2E[\mathbf{A}\mathbf{x}_o\mathbf{x}_o^{\mathrm{T}}] = 0$$
$$\mathbf{K}_{mo} - \mathbf{A}\mathbf{K}_{oo} = 0$$
$$\mathbf{A} = \mathbf{K}_{mo}\mathbf{K}_{oo}^{-1} \tag{4.4}$$

We can simply replace m with c in the above equations to find the matrix which takes in the observed data and outputs the entire (missing plus observed data) head vector.

Now we will rewrite this equation to simplify it, by using the eigenvector decomposition of the covariance matrix $\mathbf{K}_c$. Our goal is to represent it in terms of the eigenvalues and eigenvectors of the covariance matrix (already found in the eigenspace computations) and the eigenvectors of an on-diagonal block of

$\mathbf{K}_c$. Recall from the definition of a covariance matrix and the ordering we imposed on our $\mathbf{x}_c$ at the start:

$$\mathbf{K}_c = \begin{bmatrix} \mathbf{K}_{oo} & \mathbf{K}_{om} \\ \mathbf{K}_{mo} & \mathbf{K}_{mm} \end{bmatrix}$$

In the equations below, $\mathbf{S}$ is a *selector matrix*, which selects particular rows or columns of $\mathbf{K}$. The selector matrix has the form

$$\mathbf{S} = \left[\begin{array}{ccc|c} 1 & & & \\ & \ddots & & 0 \\ & & 1 & \end{array}\right]$$

$$\begin{aligned}
\mathbf{K}_{oo} &= \mathbf{S}\mathbf{K}_c\mathbf{S}^{\mathrm{T}} \\
&= \underbrace{\mathbf{S}\mathbf{V}_c}_{\mathbf{V}_o}\mathbf{\Lambda}_c\underbrace{\mathbf{V}_c^{\mathrm{T}}\mathbf{S}^{\mathrm{T}}}_{\mathbf{V}_o^{\mathrm{T}}} \\
&= \mathbf{V}_o\mathbf{\Lambda}_c\mathbf{V}_o^{\mathrm{T}}
\end{aligned}$$

Now we consider the $\mathbf{K}_{co}$ cross-covariance matrix (replacing $\mathbf{K}_{mo}$ in equation 4.4).

$$\begin{aligned}
\mathbf{K}_{co} &= \mathbf{K}_c\mathbf{S}^{\mathrm{T}} \\
&= \mathbf{V}_c\mathbf{\Lambda}_c\mathbf{V}_c^{\mathrm{T}}\mathbf{S}^{\mathrm{T}} \\
&= \mathbf{V}_c\mathbf{\Lambda}_c\mathbf{V}_o^{\mathrm{T}}
\end{aligned}$$

Combining the decompositions of $\mathbf{K}_{oo}^{-1}$ and $\mathbf{K}_{co}$,

$$\mathbf{K}_{co}\mathbf{K}_{oo}^{-1} = \mathbf{V}_c\mathbf{\Lambda}_c\mathbf{V}_o^{\mathrm{T}}(\mathbf{V}_o\mathbf{\Lambda}_c\mathbf{V}_o^{\mathrm{T}})^{-1}$$

Performing a change of variables,

$$\begin{aligned}
\mathbf{B} &= (\mathbf{\Lambda}_c^{1/2}\mathbf{V}_o^{\mathrm{T}})^{\mathrm{T}} = \mathbf{V}_o\mathbf{\Lambda}_c^{1/2} \\
\Rightarrow \mathbf{K}_{co}\mathbf{K}_{oo}^{-1} &= \mathbf{V}_c\mathbf{\Lambda}_c^{1/2}\mathbf{B}^{\mathrm{T}}(\mathbf{B}\mathbf{B}^{\mathrm{T}})^{-1} \\
&= \mathbf{V}_c\mathbf{\Lambda}_c^{1/2}\mathbf{B}^{+} \\
&= \underbrace{\mathbf{V}_c}_{\text{decode}}\underbrace{\mathbf{\Lambda}_c^{1/2}(\mathbf{V}_o\mathbf{\Lambda}_c^{1/2})^{+}}_{\text{encode}} \qquad (4.5)
\end{aligned}$$

The optimal linear estimator can thus be thought of as separate encoding and decoding phases. The decoding phase is, as before, simply taking the sum of scaled versions of the eigenvectors of the head data set. The encoding phase involves taking the dot product of the reduced input vector (containing only those dimensions actually observed by the tracking system) with the matrix above, which involves the pseudoinverse of the reduced eigenvector matrix, $\mathbf{V}_o$.

Note that if the observed dimensions of the input head change over time, this pseudoinverse must be recomputed. However, FLIRT does not store "memory" of past observations, but always provides texture data at the same (x, y) coordinates in the texture map, so this computation may be done off line. Even if it had to be moved into the reconstruction loop, it would not slow down the system unacceptably (the reconstruction step takes roughly ten seconds, which is roughly the time for the computation of this pseudoinverse).

Chapter 5

Details of the modular eigenspace algorithm

There are many ways to look at an eigenvector or eigenfunction analysis of a data set or linear system: it decorrelates the data as much as possible; it finds the fundamental modes of the system; it minimizes mean-squared error for reconstructing the original data points or vectors. However, since we are performing coding of the input data, we will consider it a coding problem.

All of the elements in our data set, the head models, can be considered points or vectors in a high-dimensional space by concatenating the rows of the range and texture maps to form one long vector per head; each vector contains roughly 65,000 dimensions for a subsampled 128×128 head. We perform an eigenvector decomposition on this data set (a matrix containing these vectors as its columns) by finding the covariance matrix of this data set and then finding the eigenvectors of that covariance matrix. (Actually, we perform an well-known equivalent operation which prevents the need to compute the full 65,000 by 65,000 covariance matrix; this is described in, for example, [2], and is equivalent to the singular value decomposition.)

The eigenvectors of a covariance matrix can be chosen orthonormal, because a covariance matrix is by definition symmetric [7, p. 273]. Let $\mathbf{A}$ be the matrix of normalized eigenvectors of the covariance matrix of a data set, where each row is one eigenvector. Then we can consider the projection of a new column vector $\mathbf{x}$ onto the subspace spanned by these eigenvectors, $\mathbf{A}^{\mathrm{T}}\mathbf{A}x$, as a *coding* operation ($\mathbf{Ax}$) combined with a *decoding* operation (multiplication by $\mathbf{A}^{\mathrm{T}}$). Assuming we have used all of the eigenvectors of the covariance matrix, the following is true for all $\mathbf{x}$, where $\mathbf{x}$ is an element of the original data set:

$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \mathbf{x} \tag{5.1}$$

That is, the eigenspace projection of $\mathbf{x}$ is an identity operation (modulo roundoff error). We want to verify that the modular eigenspace operation holds in a similar case.

Let $\mathbf{M}_1$ be a diagonal matrix representing mask 1 (for example, the mask highlighting the region around the eyes), $\mathbf{A}_1$ be the matrix of eigenvectors (as rows) of the data set of input heads multiplied by $\mathbf{M}_1$, and $\mathbf{M}_2$ and $\mathbf{A}_2$ be another mask and its eigenvectors (for example, the mouth mask). Then assuming we have kept all the eigenvectors of the two modular eigenspaces, the equation which we wish to verify for all $\mathbf{x}$ in the original data set is

$$(\mathbf{M}_1 + \mathbf{M}_2)^{-1}(\mathbf{A}_1^{\mathrm{T}}\mathbf{A}_1\mathbf{M}_1\mathbf{x} + \mathbf{A}_2^{\mathrm{T}}\mathbf{A}_2\mathbf{M}_2\mathbf{x}) = \mathbf{x} \tag{5.2}$$

Because of the identity in Equation 5.1, the projection of $\mathbf{M}_1\mathbf{x}$ into eigenspace 1 is an identity operation; the same is true for $\mathbf{M}_2\mathbf{x}$ into eigenspace 2. Therefore this equation simplifies in this restricted case to

$$(\mathbf{M}_1 + \mathbf{M}_2)^{-1}(\mathbf{M}_1\mathbf{x} + \mathbf{M}_2\mathbf{x}) = \mathbf{x}$$
$$(\mathbf{M}_1 + \mathbf{M}_2)^{-1}(\mathbf{M}_1 + \mathbf{M}_2) = \mathbf{I}$$

We have implemented the above formula; that is, the sum of the projections of an input vector onto the modular eigenspace is divided by the sum of the contributions of the masks. However, when $\mathbf{x}$ is not in the original data set, or if we have dropped higher-frequency eigenvectors from the eigenspace, the projection of $\mathbf{x}$ will not be an identity operation. Since the pseudoinverse of $\mathbf{M}_1 + \mathbf{M}_2$ will in general not be equal to the inverse of $\mathbf{A}_1^{\mathrm{T}}\mathbf{A}_1\mathbf{M}_1 + \mathbf{A}_2^{\mathrm{T}}\mathbf{A}_2\mathbf{M}_2$, please note that this equation does not hold in general. It does hold when the modular eigenspaces are orthogonal to each other and the nonzero entries on the diagonal are unity, because it reduces to the case of a projection of $\mathbf{x}$ onto an orthonormal basis by taking its dot product with the basis vectors. We can enforce this orthogonality constraint by making the masks orthogonal (i.e., no diffusion, and no overlapping regions).

The rationale behind using modular eigenspaces is that they manually delineate which regions of the face (eyes, nose, mouth) are approximately decorrelated. Separate, specialized eigenspaces can code high-resolution versions of these various features, and they can be combined independently, providing more reconstruction parameters. However, we found that when the modular eigenspaces were not orthogonal, the reconstructions sometimes had errors including large variations in the head shape (Figure 6.3). From experiments described in Chapter 6, we found that forcing the modular eigenspaces to be orthogonal caused other stability problems. We are considering techniques to reduce the modular eigenspace reconstruction error by interpolating nonlinearly among the modular eigenspaces' projections. Our current work involves using a search technique to automatically find masks which minimize cross-validation error on a given data set.

Chapter 6

Experiments

Two basic forms of experiments were performed. The first involved the reconstruction of heads which were used as data points for computing the eigenspaces. In this situation it was expected that when using all of the eigenvectors for reconstruction, the single eigenspaces would have less reconstruction error than the modular eigenspaces (see Chapter 5), because this is the criterion which an eigendecomposition optimizes. We wanted to see whether the modular eigenspaces would improve the truncation error (incurred when dropping the higher-frequency eigenvectors).

Our error metric was the Euclidean distance between an input scanned head and its eigenspace reconstruction. This metric has the disadvantage that it has no perceptual foundation; a reconstruction with a seemingly low error can be perceptually very different from the input. Heuristically, in the data presented here, a distance of a few hundred is relatively insignificant, while an error of a few thousand is very significant (reconstruction bears only passing resemblance to the input).

Reconstruction error (8 heads, 8 eigenvectors)

	mean	std. dev
single	0.05192	0.01838
modular	0.07502	0.02749

(As expected, these are certainly due to roundoff error.)

Reconstruction error (8 heads, 6 eigenvectors)

	mean	std. dev
single	1002	668
modular	921	691

The differences between the latter reconstructions may also be attributable to roundoff error. The modular eigenspaces' reconstructions seemed to have less error, but the perceptual differences between the reconstructions were not significant.

The other experiment involved cross-validation using the "leave-one-out" technique. An eigenspace (or set of eigenspaces, in the case of modular eigenspaces) was created from all but one of the available CyberWare scans; the remaining head was projected onto both the single and modular eigenspaces, and the reconstruction errors were compared. In this case we expected the modular eigenspaces to perform better than the single eigenspace, because there were separate reconstruction parameters for the eyes, nose, and mouth regions. However, this was not the case.

Cross-validation error (24 heads, 18 eigenvectors)

	mean	std. dev
single	4594	1935
modular	4754	2012

The reconstructed heads from the cross-validation experiments were compared visually. In cases where the modular eigenspace reconstruction error differed noticeably from the single eigenspace's, the visual discrepancies between the models were negligible. For the most part, all of the reconstructed heads looked like the mean or "average" face, and the differences between the modular and single eigenspaces' reconstructions were limited to small variations in the eye and mouth regions.

We hypothesized that the presence of the hair and back of the head in the eigenspace was causing the eigenspace to concentrate less on the face, which was the region of interest. For this reason a new set of masks (eyes, nose, and mouth) was created which cropped out everything except the facial region. Because of a lack of time, the above cross-validation and truncation error experiments were not run with these eigenspaces. Instead, data from FLIRT was used to visually compare the reconstructions. In the following experiments 8 input heads were used to compute the eigenspaces and all 8 eigenvectors were kept.

The first set of experiments used smoothed versions of the modular eigenspace masks (Figure 6.1). One of the CyberWare scanned heads not in the eigenspaces was decimated to the dimensions FLIRT ordinarily provides, and projected into both a single and the modular eigenspaces. The results of this are shown in Figure 6.2. Both heads look approximately the same, with only minor variations in the nose and mouth regions. An input head from FLIRT was also projected into these eigenspaces; the results are shown in Figure 6.3. The obvious differences may be attributable to the non-orthogonality of the modular eigenspaces and the fact that we were combining them improperly (Chapter 5).

To try to remedy the reconstruction error, a new set of orthogonal masks was generated (Figure 6.4). In this situation it was expected that the modular eigenspaces would code the eigenfeatures better than the single eigenspace for the entire head, although edges might be visible along the boundaries where one eigenspace stopped coding for a region and another one started. Again, one of the scanned heads not in the eigenspaces was reconstructed (Figure 6.5). This time the modular eigenspace did qualitatively better in the reconstruction; the eyes looked straight ahead instead of off to the side. Furthermore, no hard edges were visible in the modular reconstructed model. When input data from FLIRT was projected into this eigenspace, however, the result was not just visible edges, but fairly drastic protrusions of the eigenfeatures beyond the rest of the face (Figure 6.6). This is probably attributable to variations in the lighting of the user being tracked by FLIRT. An overhead light source was used, likely causing the nose to be illuminated more than the rest of the face, and causing shadows in the cheeks. This could cause the average energy of the vector being projected into the nose eigenspace to be disproportionately larger than that of the "rest of the face" vector. Since we estimated the structure of the head solely from the texture, these regions protruded further in the reconstruction. However, there were also color variations between the various eigenfeatures visible in the actual model; these did not disappear even when the structure information was removed from the eigenspace.

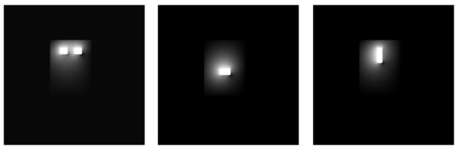

Figure 6.1: The cropped and diffused modular eigenspace masks.

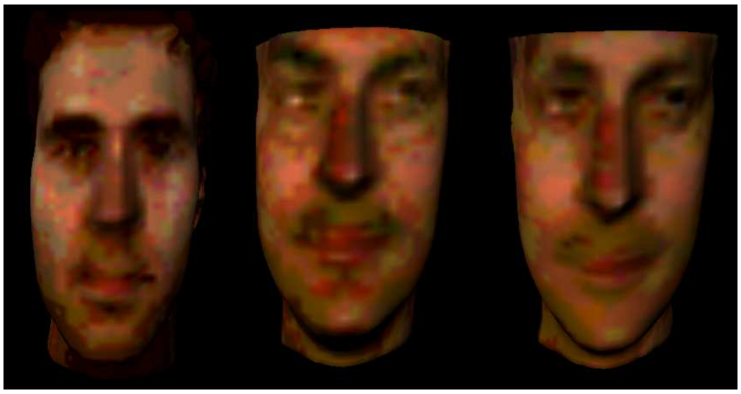

Figure 6.2: Cropped, diffused mask reconstruction: from left to right, original head, single eigenspace reconstruction, modular reconstruction.

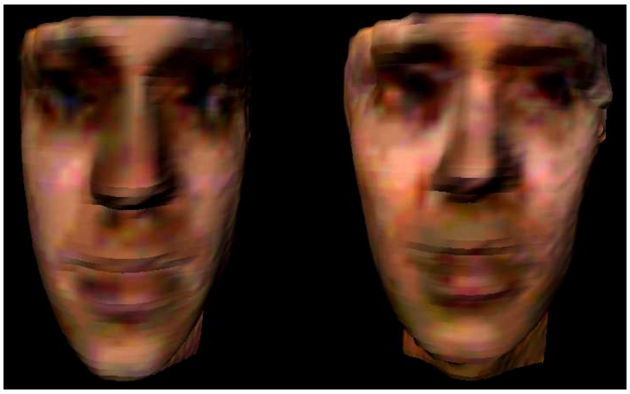

Figure 6.3: Cropped, diffused mask reconstruction from FLIRT: video image is as in Figure 1.2. Left is single eigenspace reconstruction, right is modular reconstruction.

Figure 6.4: The orthogonal modular eigenspace masks.

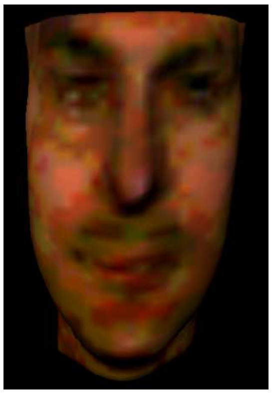

Figure 6.5: Orthogonal mask reconstruction. Note that the eyes look straight ahead, unlike the diffused mask versions.

Figure 6.6: Orthogonal mask reconstruction from FLIRT: video image is as in Figure 1.2. Lighting variations on the face may be the cause of the obvious borders around the eigenfeatures.

Chapter 7

Conclusions

7.1 Analysis of the system

When performing an eigenvector analysis, it is essential to normalize the input data to correct for variations which the eigenspace should not model; this is the reason, for example, for the alignment stage, which prevents the eigenvectors from attempting to model rotations of the head. We also found that if the input head from the tracking system was significantly brighter than the head models in the eigenspace, the reconstructions were severely deformed, probably because the reconstruction coefficients were being increased beyond their statistically valid range. The experiments performed with the tracking system while modifying the modular eigenspace masks (Chapter 6) indicate that more work is needed to stabilize the image being used as input to the reconstruction pipeline. For example, illumination correction might help the problem of protruding eigenfeatures; however, the complete solution to this problem would involve the full shape from shading computation described in [4], and would probably reduce the reconstruction step from "interactive time" (a few seconds) to an offline computation.

The theory behind the modular eigenspace technique (Chapter 5) and experiments done with various modular eigenspace masks (Chapter 6) indicate that with the current technique for combining the modular eigenspaces' contributions, the eigenspaces should be orthogonal to perform properly, but that such orthogonality leads to instability in the reconstruction (tendency to create hard edges in the reconstruction). The diffusion was an attempt to remedy this, but it appears that this causes too great an overlap among the modular eigenspaces, leading to unacceptably large reconstruction error. It may be possible to trade off accuracy for stability by, for example, only blurring the modular eigenspace masks by a very slight amount. Conversely, it may be possible to choose another (nonlinear) combination algorithm.

All of the cross-validation experiments that were performed indicate that we did not have enough data to create a reasonable statistical model. The visual differences between the heads reconstructed with a single eigenspace and with the modular eigenspaces were negligible. In general, as the data set shrinks, the eigenvectors of the data set look more and more like the input vectors rather than like their modes. We found that our eigenheads looked very much like the original data set, as opposed to deformation modes of the head (see [4] for a good example). At the time of this writing we had only 27 head scans to form our eigenspaces. For comparison, the shape from shading experiments done in [4] used a well-known database of 347 CyberWare-scanned air force pilots.

A large amount of time went into rewriting basic input/output code several times, to conform to FLIRT's file formats and to avoid forcing the existing system to change. In retrospect, the existing system should have been modified at the start of the project to use a standard matrix library, which would have allowed the reconstruction system to be implemented in a fraction of the time it took, allowing more development of the underlying algorithms, and leaving less opportunity for error in matrix-related functions. It is recommended that this course of action be taken rather than continuing further development with the existing code base for the reconstruction system.

7.2 Research technique

In the hope that it will be helpful, some of the decisions made during the course of this project relating to the order in which things were done are listed here.

This project was begun with implementation. Instead of having time allotted at first for a review of the literature in the field, an early demonstration of the graphics front end to the reconstruction system was needed. From that time iterative refinements were made to the file formats and other aspects of the pipeline, as deficiencies were uncovered where precision was being lost (for example, quantizing the modular eigenspace masks to integer values between 0 and 255). This kept the intensity of the work cycle constant, leaving little time for a higher-level review of the concepts and previous work.

Especially in a research setting, it is essential to keep up with the current literature, to know what others are doing, to know in which directions the field is going, and simply to know what is known. It seemed at the end of the project that the first thing that should have been done was carefully consider the previous work in the field (namely, [3] and [4]). Once these papers had been reviewed, it seemed that some of the problems we had encountered during implementation might have been avoided by earlier consideration of these works.

However, discussion with other people in the research group indicated that our lack of time was not an uncommon problem, and in fact it is the exception that a supervisor requires full review and/or implementation of prior work before beginning research, even when the student may wish it. In fact, it seems that a strict initial literature review dissuades some students from pursuing research, possibly because they do not feel they are making original contributions by doing so. A professor in the MIT EECS department once said in a class that he recommends that his graduate students read other papers in the field only after completing the implementation of their own system. Apparently this avoids the accidental appropriation of ideas, and increases the number of new approaches taken.

The overall point, however weak, is that it may not be a mistake to implement a system before performing a literature review. Implementation brings a kind of experience complementary to that acquired from reading. It is a mistake to avoid the literature review altogether; it is probably equally bad to have an implementation work properly "the first time", because of the possibility that the algorithm may, in fact, be flawed and its seeming to work properly is just pure chance. We learn best from our own mistakes. Fortunately, this project offered many opportunities for learning.

Appendix A

Catalog of the software

This section describes at a high level most of the software developed for the reconstruction pipeline. The individual programs are self documenting (with few exceptions, noted below); this means that for the most part the programs can be run with no command line arguments in order to get a usage message displayed. The organization of this section is roughly as follows: conversion utilities are listed first, followed by programs used to compute the eigenspaces, in the order they would be used; at the end are the testing and reconstruction programs.

float2pgm Converts FLIRT's .f file format to a PGM suitable for viewing in xv.

float2mat Converts FLIRT's .f file format for matrices into Matlab's .mat file format.

float2iv reads in a .f file of a CyberWare head and converts it into an Open Inventor file. Makes the following assumptions:

1. The first 1/4 (vertically) of the .f file represents the range data for the head.
2. The rest of the .f file represents the texture map of the head; values are uninterleaved, in red, green, and blue order, and range from 0 to 255.
3. Head is oriented vertically (i.e., top of head is at top of image)

range2pgm Converts a set of range data in standard CyberWare format into a PGM file suitable for viewing with xv.

mapIntensity Reads in a PGM image and maps the intensity values [minIn.. maxIn] to [minOut..maxOut], passing through unchanged all pixels outside of the [minIn..maxIn] range. (A more general version of **threshold**.)

threshold Reads in a PGM image and linearly maps intensity values 0..255 to threshold..(255-threshold).

diffuse Reads in a PGM image, diffuses it using a 2D Laplacian lattice, and writes out the result in .f format. Command line options turn on/off Open Inventor display window, and set alpha value and number of iterations. Usage documentation describes diffusion algorithm.

lambdaCalc Calculates Lambda, the ratio of structure to texture power for a given data set.

combiner Reads in two files, a .f file containing a CyberWare head (combined structure and texture), and a .f file containing a mask. Multiplies the structure and texture by the mask (so that higher intensities in the mask show up more brightly in the resulting file), multiplies the texture components by lambda (see **lambdaCalc**), and writes out the resulting head's data in .f format.

floatDeci Subsamples a .f file by a factor of n in the horizontal and vertical dimensions.

eigenDecomp Performs eigenvector decomposition on a set of input vectors. See comments at top of code for input/output file descriptions.

estimatorCalc Computes the optimal linear estimator for a given set of eigenvectors and a known set of dimensions. The estimator, when given these dimensions later, will use the complete set of eigenvectors to estimate the unknown dimensions.

align Aligns heads. Input is a data file containing information about each head, including four points to align to the first ("canonical") head. Output is head-aligned-[0...n].iv, which can be converted to .f format with **iv2float**.

iv2float Renders an Inventor model using a "virtual CyberWare scanner" into cylindrical coordinate depth and texture maps. Writes the result in .f format. Assumes dimensions of model are those which **float2iv** outputs (Y coordinate from 0 to 1; radius between 0 and 255).

alignToFLIRT Aligns heads based on alignment data from FLIRT's alignment program (canonicalCyber) but using Inventor-based ray caster to interpolate model.

blinker Cycles through Inventor models given on command line at a rate of half a second per model. Useful for verifying alignment of head models.

singleEigHeadTest, modularEigHeadTest Reconstruct original input heads based on either single or modular eigenspace decomposition. These programs have path names hard coded into them, and are not self documenting (take no command line arguments).

modularErrorTest Reads in a data file (the format of which is described in the usage) and reconstructs all the heads listed in it based on the coefficients in the file (acquired from **eigenDecomp**). Used to compare modular eigenspace reconstruction to single eigenspace reconstruction without leaving the test head out of the eigenspace. (Note: with all eigenvectors preserved, single eigenspace should always do better than modular.)

projectionErrorTest Projects a head onto an eigenspace or set of eigenspaces, reconstructs the head based on the acquired coefficients, and computes the error between the original head and the reconstructed one, printing this floating point number on standard output. Also outputs reconstructed head's data to "recon-test.f".

reconMan A manual reconstruction demo in which the user can specify (via a graphical user interface) the coefficients of reconstruction. Has pathnames hard-coded in, and is not self-documenting. See comments at top of code for running instructions.

reconTrack uses the input from the FLIRT system as the basis for model reconstruction. It projects the incoming head data onto an eigenspace or set of eigenspaces (using the estimators computed by **estimatorCalc** to fill in unknown data), and reconstructs the head based on the acquired coefficients.

Bibliography

[1] Jebara, Tony S. and Alex Pentland. "Parametrized Structure from Motion for 3D Adaptive Feedback Tracking of Faces," *MIT Media Laboratory, Perceptual Computing Technical Report #401.*

[2] Turk, Matthew and Alex Pentland. "Eigenfaces for Recognition," *Journal of Cognitive Neuroscience,* Vol. 3, No. 1, 1991.

[3] Moghaddam, Baback and Alex Pentland. "Face Recognition using View-Based and Modular Eigenspaces," *Automatic Systems for the Identification and Inspection of Humans, SPIE,* Vol. 2277, July 1994.

[4] Atick, Joseph J., Paul A. Griffin and A. Norman Redlich. "Statistical Approach to Shape from Shading: Reconstruction of Three-Dimensional Face Surfaces from Single Two-Dimensional Images," *Neural Computation,* Vol 8, 1996.

[5] Arun, K. S., T. S. Huang, and S. D. Blostein. "Least-Squares Fitting of Two 3-D Point Sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-9, No. 5, September 1987.

[6] Azarbayejani, Ali and Alex Pentland. "Recursive Estimation of Motion, Structure and Focal Length," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* June 1995.

[7] Strang, Gilbert. *Introduction to Linear Algebra.* Wellesley, MA: Wellesley-Cambridge Press, 1993.