Light Field Appearance Manifolds

by

Chris Mario Christoudias

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2004

Author ....
Department of Electrical Engineering and Computer Science
August 23, 2004

Certified by ....
Trevor Darrell
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Light Field Appearance Manifolds

by

Chris Mario Christoudias

Submitted to the Department of Electrical Engineering and Computer Science
on August 23, 2004, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

Statistical shape and texture appearance models are powerful image representations, but previously had been restricted to 2D or 3D shapes with smooth surfaces and lambertian reflectance. In this thesis we present a novel 4D appearance model using image-based rendering techniques, which can represent complex lighting conditions, structures, and surfaces. We construct a light field manifold capturing the multi-view appearance of an object class and extend previous direct search algorithms to match new light fields or 2D images of an object to a point on this manifold. When matching to a 2D image the reconstructed light field can be used to render unseen views of the object. Our technique differs from previous view-based active appearance models in that model coefficients between views are explicitly linked, and that we do not model any pose variation within the deformable model at a single view. It overcomes the limitations of polygonal based appearance models and uses light fields that are acquired in real-time.

Thesis Supervisor: Trevor Darrell
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

Firstly, I would like to thank my advisor, Trevor Darrell, for his endless support and guidance. Without him this thesis would not have been possible. In the short time I have been at MIT I have learned a lot from him. He is a very creative and gifted scientist and a caring person. I am very fortunate and grateful to be his student.

I am grateful to the members of the Vision Interface Group for their support. In particular, I would like to thank Greg Shakhnarovich, Ali Rahimi, Kevin Wilson and David Demirdjian for their advice and encouragement. I would like to thank Louis-Philippe Morency for taking me under his wing when I first came to MIT. I also would like to acknowledge him for all of his help throughout the work of this thesis[1]. He has always been there for me as a fellow co-worker and more importantly as a friend. I also would like to thank my good friend Neal Checka for all of his support.

I would like to thank my undergraduate advisor, Peter Meer, for introducing me to research and the field of computer vision. Without him I would not be where I am today.

I gratefully acknowledge the Kanellakis fellowship for their generous funding during my first year at MIT. I also want to thank the participants of our data collection.

Finally, I would like to thank my parents John and Aphrodite Christoudias, sister Tina Christoudias and my family for all of their love and support. They have given me so much to be grateful for and I love them deeply.

[1] Part of the work presented in this thesis is jointly published in the 8th European Conference on Computer Vision. See [7] for details.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Descriptive and compact appearance models have been of interest to many computer vision and graphics researchers for the last decade. These models aim to provide a rich, intuitive description of the appearance of an object class from a set of example images of prototype objects. Since these models are learned from examples, they provide a natural and powerful way of describing the appearance of objects of the same class. Appearance models have many applications in computer vision and graphics including computer animation, object recognition, segmentation and tracking. Given an image of a novel object outside of the model database, an appearance model is fit to the image and the parameters of the model are then used to fully describe the appearance of the imaged object. In computer graphics, these parameters may be used to synthesize the input object from unseen views, novel lighting conditions or with a different configuration, properties. In computer vision, these parameters provide useful information about the object (e.g. its pose, articulation, age, gender).

Multidimensional Morphable Models (MMM) [24], Active Appearance Models (AAM) [8], and their extensions have been applied to model a wide range of object appearance. These methods form a class of appearance models known as *shape and texture appearance models* or *deformable models*. Shape and texture appearance models vectorize each example image into shape and texture vectors and then jointly model the data variation in each of these vector spaces. Shape vectors define the geometry of each example object and are used to place the objects into correspon-

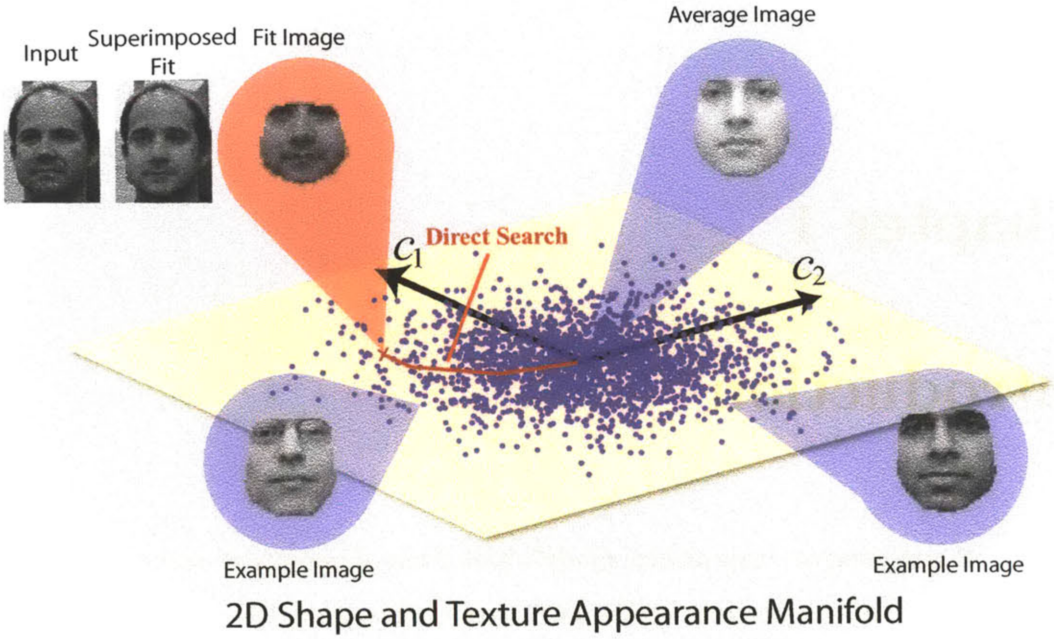


Figure 1-1: Two-dimensional shape and texture appearance manifold.

dence. Texture vectors are the "shape free" versions of each example image and are defined by warping each image to the model reference shape. To form the appearance model, principal axes of variation are computed in each vector space using Principal Components Analysis (PCA). Unlike pure intensity based models (e.g. Eigenfaces [36]), shape and texture appearance models provide a more flexible and compact representation of object appearance since they decouple the variation due to geometry and intensity. Conceptually, each prototype image is a point in the combined texture-shape vector space and lies on a low-dimensional appearance manifold that represents all valid shapes and textures of the object class (see Figure 1-1). MMMs and AAMs parameterize this manifold by a hyperplane using PCA, and the model parameters of an input image are computed by projecting the image onto this hyperplane.

As presented, MMMs and AAMs are able to faithfully model many linear object classes (e.g. frontal faces). They have difficulty to represent many important aspects of object appearance, however, including object pose variation, illumination and dynamics. Each of these aspects introduce their own set of challenges and over the

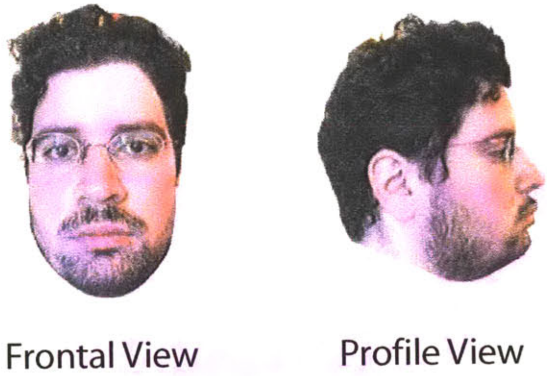


Figure 1-2: Pose variation can result in non-linear differences in appearance: parts of the object are visible in one view but absent from the other view.

years researchers have made progress toward extending deformable models to overcome these limitations [31, 9, 16, 11]. In this thesis we present a novel extension of deformable models that enables them to easily and naturally model object classes with complex surface reflectance and geometry across pose.

Traditional deformable models are only able to handle a small amount of pose variation, since large pose changes lead to non-linear differences in appearance (see Figure 1-2). It is possible to model large pose variation in a single 2D deformable model [31], but requires the use of non-linear models and is therefore complex to optimize. Alternatively, local-linear models can be fit to the different portions of pose space [9]. These models are then linked together such that model parameters can be easily translated from one local-linear model to another. Although this solution is practical and intuitive, it has difficulty modelling view-dependent textures as multiple poses are blended at a single model. Also, with object classes that exhibit a large degree of self-occlusion, such a solution would require a large number of local-linear models rendering it inefficient.

Large pose variation is easily modelled using 3D; a polygonal 3D appearance model was proposed by Blanz and Vetter [4]. With their approach the view is an external parameter of the model and does not need to be modelled as shape variation. However, this technique is based on a textured polygonal mesh which has difficultly representing fine structure, complex lighting conditions and non-lambertian surfaces.

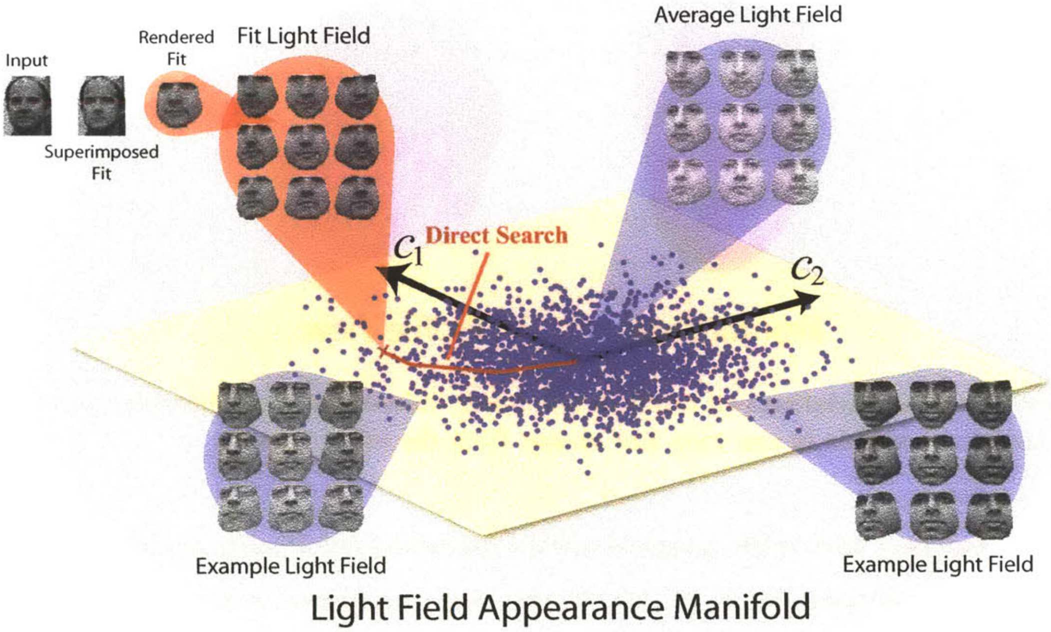


Figure 1-3: Light field appearance manifold. Each point on the manifold is a 4D light field representing the 3D shape and surface reflectance of an object. The light field of an object is constructed by computing its projection onto the shape-texture appearance manifold. A 2D input image is matched to a point on this manifold by interpolating the shape and texture of neighboring prototype light fields.

Due to the accuracy of the 3D surfaces needed with their approach, the face scans of each prototype subject cannot be captured in real-time and fine structure such as hair cannot be acquired.

In this thesis we propose a 4D deformable model using image-based rendering [25, 19] rather than rendering with a polygonal mesh. We use a light field representation, which does not require any depth information to render novel views of the scene. With light field rendering, each model prototype consists of a set of sample views of the plenoptic function [1]. Shape is defined for each prototype and a combined texture-shape PCA space computed. The resulting appearance manifold (see Figure 1-3) can be matched to a light field or 2D image of a novel object by searching over the combined texture-shape parameters on the manifold. When matching to an image, we automatically estimate the object's pose by performing gradient decent over the views of the light field.

We construct a light field appearance manifold in both the spirit of MMMs and AAMs. We first show how to construct this manifold using optical-flow shape features and match to a novel object using a matching algorithm analogous to Beymer and Poggio [3]. We also demonstrate how to define a light field appearance manifold using point shape features and we extend the direct search matching algorithm of [8] to light fields. Specifically, we construct a Jacobian matrix consisting of intensity gradient light fields. With this approach a 2D image is matched by rendering the Jacobian at the estimated object pose. Our approach can easily model complex scenes, lighting effects, and can be captured in real-time using camera arrays [41, 38].

1.1 Related Work

Statistical models based on linear manifolds of shape and/or texture variation have been widely applied to the modelling, tracking, and recognition of objects [3, 13, 24, 29]. In these methods small amounts of pose change are typically modelled implicitly as part of shape variation on the linear manifold. For representing objects with large amounts of rotation, nonlinear models have been proposed, but are complex to optimize [31]. An alternative approach to capturing pose variation is to use an explicit multi-view representation which builds a PCA model at several viewpoints. This approach has been used for pure intensity models [28] as well as shape and texture models [9]. A model of inter-view variation can be recovered using the approach in [9], and missing views could be reconstructed. However, in this approach pose change is encoded as shape variation, in contrast to 3D approaches where pose is an external parameter. Additionally, views were relatively sparse, and individual features were not matched across views.

Shape models with 3D features have the advantage that viewpoint change can be explicitly optimized while matching or rendering the model. Blanz and Vetter [4] showed how a morphable model could be created from 3D range scans of human heads. This approach represented objects as simply textured 3D shapes, and relied on high-resolution range scanners to construct a model; non-lambertian and dynamic effects are difficult to capture using this framework. With some manual intervention, 3D models can be learned directly from monocular video [15, 30]; an automatic method for computing a 3D morphable model from video was shown in [5]. These methods all used textured polygonal mesh models for representing and rendering shape.

Multi-view 2D [9] and textured polygonal 3D [4, 15, 30] appearance models cannot model objets with complex surface reflectance and geometry. Image-based models have become popular in computer graphics recently and can capture these phenomenon; with an image-based model, 3D object appearance is captured in a set of sampled views or ray bundles. Light field [25] and lumigraph [19] rendering techniques create new images by resampling the set of stored rays that represent an object. Most recently the unstructured lumigraph [6] was proposed, and generalized the light field/lumigraph representation to handle arbitrary camera placement and geometric proxies.

Recently, Gross et. al. [20] have proposed *eigen light fields*, a PCA-based appearance model built using light fields. They extend the approach of Turk and Pentland [36] to light fields and define a robust pose-invariant face recognition algorithm using the resulting model. A method to morph two light fields was presented in [42]; this algorithm extended the classic Beier and Neely algorithm to work directly on the sampled light field representation and to account for self-occlusion across views. Features were manually defined, and only a morph between two light fields was shown in their work.

In this thesis we develop the concept of a *light field appearance manifold*, in which 3 or more light fields are "vectorized" (in the sense of [3]) and placed in correspondence. We construct a light field deformable model of facial appearance from real images, and show how that model can be automatically matched to single static intensity images with non-lambertian effects (e.g. glasses). Our model differs from the multi-view appearance model of [9] in that we build a 4D representation of appearance with light fields. With our method, model coefficients between views are explicitly linked and we do not model any pose variation within the shape model at a single view. We are therefore able to model self-occlusion and complex lighting effects better than a multi-view AAM. We support this claim with experiments in Chapter 7.

1.2 Outline

We first introduce the concepts of image warping and morphing in Chapter 2. In Chapter 3 we present a brief overview of deformable models. In specific, we discuss Multidimensional Morphable Models and Active Appearance Models. Light field rendering is discussed in Chapter 4 and light field morphing in Chapter 5. The main contributions of this thesis are detailed in Chapter 6, where we discuss light field appearance manifolds. Experiments and results are presented in Chapter 7. Finally, in Chapter 8 we provide concluding remarks and discuss possible avenues of future work.

Chapter 2

Image Warping and Morphing

Object metamorphosis is an important topic of computer graphics [17]. It concentrates on how objects may evolve over time or smoothly change into one another. Metamorphosis is a naturally occurring phenomenon and therefore its study is pertinent to both graphics and vision research. As described by Wolberg [40], object metamorphosis between two or more objects is comprised of three steps: (1) feature specification, (2) geometric alignment, and (3) color blending. Feature specification is used to establish correspondence between the objects involved in the morph. The resulting deformation field is used to geometrically align each object by means of a warping operation. Color blending is then performed. This step usually consists of a linear interpolation between the colors of the geometrically aligned objects.

There are many morphing techniques defined for 2D images [17]. All of these methods follow the same general image interpolation and blending paradigm outlined above. This methodology is detailed in Sections 2.1, 2.2 and important issues associated with image warping are discussed. Each image morphing algorithm is differentiated by how it defines a deformation field and blends textures. Beier and Neely were one of the first to describe such an algorithm [2]. They present a simple, intuitive method for defining an image deformation field via user defined shape features. The study of their algorithm serves as a good introduction to image warping and morphing. We discuss this algorithm in Section 2.3. The deformable models of [8] make use of a piecewise image warping algorithm that establishes image correspondence via

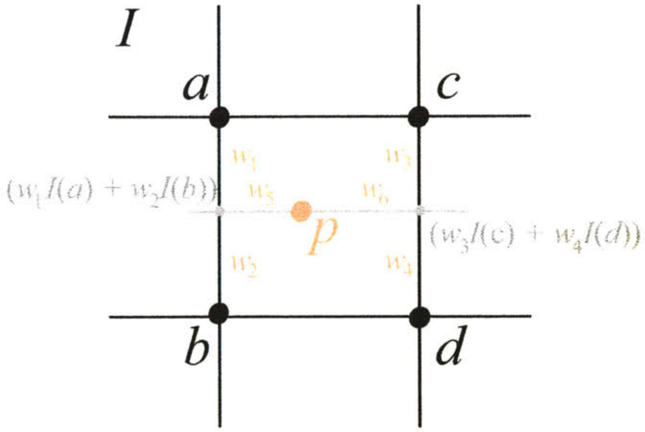


Figure 2-1: Bilinear interpolation. The texture at point p is interpolated from the texture of its four neighbors a, b, c, d.

the use of a triangular mesh. We conclude with the description of this algorithm in Section 2.4.

2.1 Image Warping

Let $d(x, y)$ be a 2D deformation of the image I such that the pixel $(x, y) \in I$ is at location $d(x, y)$ in the warped image. Namely,

$$I_w(d(x, y)) = I, \tag{2.1}$$

where I_w is the warped image formed by deforming the image I along the deformation field $d(x, y)$. Let $d_r(x, y)$ be the 2D deformation field found by flipping $d(x, y)$. Using $d_r(x, y)$ we can re-write Equation (2.1) as

$$I_w = I(d_r(x, y)) \tag{2.2}$$

Relationship (2.2) may be alternatively written as

$$I(d_r(x, y)) = I \circ d_r(x, y). \tag{2.3}$$

Equation (2.3) is a convenient notation that we will use to denote the warping operation. If $d(x, y) \in \mathbb{Z}^2$ then the computation of I_w is implemented via a straightforward

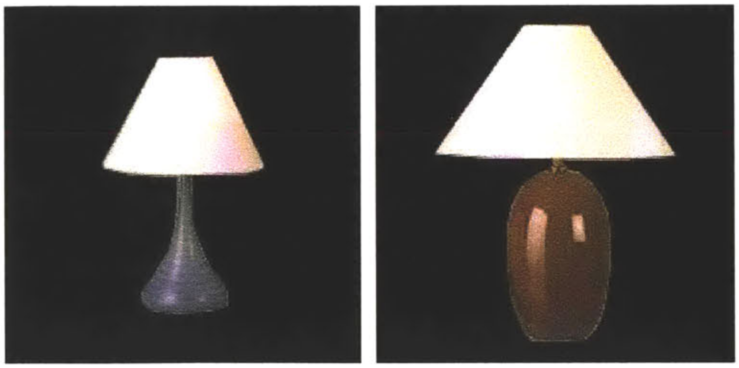


Figure 2-2: 180 × 180 color lamp images.

pixel copy operation. In general $d(x,y) \in \mathbb{R}^2$ and the texture values of the input image must be interpolated. In the first part of this section we discuss two popular image interpolation methods. Given an interpolation technique, the warped image is computed using one of two complementary algorithms. *Reverse warping* finds for each pixel in the warped image the corresponding pixel in the input image. *Forward warping* does exactly the opposite. We conclude this section with a discussion comparing these two algorithms.

Thus far we have discussed image warping in the context of a pre-defined deformation field $d(x,y)$. The definition of $d(x,y)$ is at the center of all image morphing algorithms. Methods for computing $d(x,y)$ are discussed in Sections 2.3, 2.4.

2.1.1 Image Interpolation

Nearest Neighbor

Nearest neighbor interpolation is the simplest of interpolation algorithms. It interpolates the image by rounding $d(x,y)$ to the nearest integer. Namely, with nearest neighbor interpolation I_w is computed as

$$I_w = I \circ r(d(x,y)) \tag{2.4}$$

where $r : \mathbb{R}^2 \to \mathbb{Z}^2$ is a function that rounds each entry of $d(x,y)$ to the nearest integer. Nearest neighbor is computationally efficient, however, may result in image aliasing effects around highly textured portions of the input image (e.g. near a sharp intensity

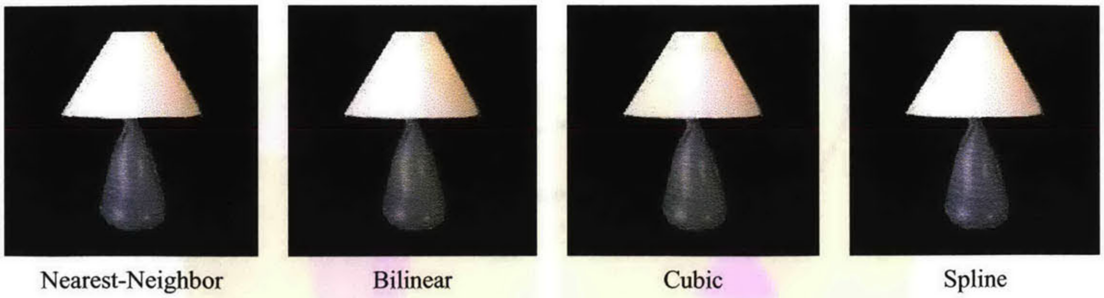


Figure 2-3: First lamp of Figure 2-2 warped halfway towards the second lamp, obtained using nearest neighbor, bilinear, cubic and spline interpolation respectively. Nearest-neighbor and bilinear interpolation obtain comparable results to higher order interpolation methods with much less computation.

gradient). Nonetheless, the simplicity of nearest neighbor has made it quite popular. Another popular image interpolation technique is bilinear interpolation. This method is slightly more complicated, however, generally leads to a large improvement in the synthesized texture of the warped image. We discuss this method next.

Bilinear Interpolation

Bilinear interpolation is summarized by Figure 2-1. In the figure, the texture at point p is computed by considering the texture of its four neighbors:

$$I(p) = w_5(w_1 I(a) + w_2 I(b)) + w_6(w_3 I(c) + w_4 I(d)) \tag{2.5}$$

where

$$\begin{aligned} w_1 &= (p_y - a_y)/(b_y - a_y) \\ w_2 &= (b_y - p_y)/(b_y - a_y) \\ w_3 &= (p_y - c_y)/(d_y - c_y) \\ w_4 &= (d_y - p_y)/(d_y - c_y) \\ w_5 &= (p_x - a_x)/(c_x - a_x) \\ w_6 &= (c_x - p_x)/(c_x - a_x) \end{aligned}$$

Equation (2.5) computes $I(p)$ by first performing a vertical interpolation on the four neighbors of p followed by a horizontal interpolation. Bilinear interpolation achieves better performance than nearest neighbor by smoothing over the region of interest, thus avoiding aliasing effects. For greater amounts of smoothing one may use higher

Interpolation Technique	**Execution Time** (ms)
Nearest Neighbor	171
Bilinear	250
Cubic	591
Spline	2,864

Table 2.1: Interpolation execution times.

order interpolation techniques (e.g. cubic interpolation) or spline based methods. These techniques tend to be more complicated and less computationally efficient and in many cases they do not greatly improve performance. We compare some of these algorithms along with nearest neighbor and bilinear interpolation in the following subsection.

Example

Consider the 180 × 180 color lamp images of Figure 2-2. Figure 2-3 displays the first lamp warped halfway toward the second lamp, obtained using nearest-neighbor, bilinear, cubic and spline interpolation respectively. Each of these images were obtained using MATLAB's **interp2** routine. There is a noticeable difference between nearest-neighbor and bilinear interpolation (e.g. the specular highlights of the lamp appear more smooth and photo-realistic using bilinear interpolation). Notice, however, there is less of a difference between bilinear, cubic and spline interpolation. Of these interpolation methods spline interpolation is the most expensive.

Table 2.1 gives the running time of each interpolation techniques executed on the above lamp image. Bilinear interpolation is the optimal choice for this example since it gives similar results to the higher order interpolation methods with much greater efficiency. In general, the choice of interpolation method is application dependent, however, as demonstrated by this example nearest-neighbor and bilinear interpolation can give comparable results with much less computation.

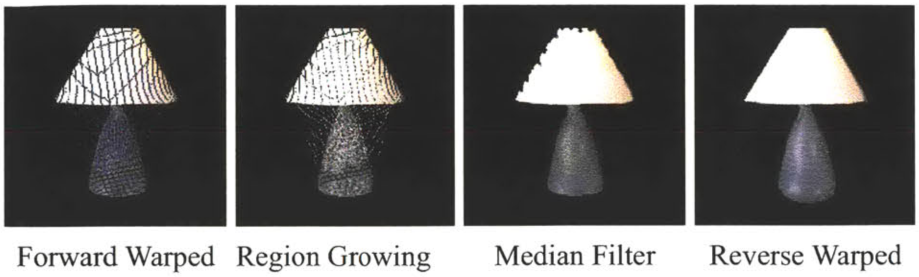


Figure 2-4: Forward vs. reverse warping: The holes in the forward warped image result from the possibly many-to-one mappings in $d(x, y)$. The holes can be removed by growing regions and then using a median filter, however, at the cost of blurring the image. Reverse warping gives the best results.

2.1.2 Reverse vs. Foward Warping

Forward and reverse warping are outlined by Algorithm 1. Forward warping traverses the input image to compute I_w whereas reverse warping traverses the warped image. Note, that unlike reverse warping, forward warping does not guarantee that every pixel in I_w is assigned a texture value. This results in black patches or *holes* in the warped image. Consider the lamps of Figure 2-2. Correspondence was established manually and a displacement field computed via the piecewise image warping algorithm of Section 2.4. Figure 2-4 displays the first lamp warped half-way toward the other lamp both using forward and reverse warping. Note the holes in the forward warped image. These holes are usually not more than a few pixels wide and thus can be removed by growing regions and then applying a median filter as was done in the figure. The median filter successfully removes undesirable holes but it also blurs the image.

The artifacts associated with forward warping may be avoided using reverse warping. Note reverse warping utilizes the deformation field, $d_r(x, y)$, given by Equation (2.2), that specifies for each pixel of the warped image the corresponding pixel in the input image. Comparing Equations (2.1) and (2.3) one finds that $d_r(x, y)$ has the opposite function of $d(x, y)$ and can therefore be computed by effectively flipping $d(x, y)$. Since $d(x, y)$ does not specify a correspondence for each pixel in I_w, some values of $d_r(x, y)$ would not be specified with this technique and these values would

Algorithm 1 Forward vs. Reverse Warping

```
$I$ is the input image and $I_w$ is the warped output image.

*Forward warping* :
**for** all $(x, y) \in I$ **do**
  $I_w(d(x, y)) = I(x, y)$
**end for**

*Reverse warping* :
**for** all $(x, y) \in I_w$ **do**
  $I_w(x, y) = I(d_r(x, y))$
**end for**
```

need to be approximated by interpolating the values of their neighbors. Alternatively, as will be seen from Sections 2.3 and 2.4 $d'(x, y)$ may be more accurately computed directly from the input images in analogous fashion to $d(x, y)$.

Unlike forward warping, reverse warping is able to compute the warped image without the introduction of holes or blurring the image. It also is simpler as it does not warrant the use of region growing and a median filter. For these reasons, reverse warping is preferred over the use of forward warping whenever reverse warping is applicable. Both reverse and forward warping are used to define the deformable models of this thesis.

2.2 Image Morphing

Consider images I_1 and I_2 of two different objects that we wish to morphologically combine. Note simply blending the texture of each image fades one image into the other and does not produce a smooth transition between each object (see Figure 2-5). A morph between these images is computed via a convex combination of both their texture and shape. Each object is first aligned to an intermediate geometry using image warping and the morphed image is formed by blending the textures of the warped images. More formally, the morphed image, I_m, is computed as

$$I_m = (1 - \alpha)I_w^1 + \alpha I_w^2 \qquad (2.6)$$

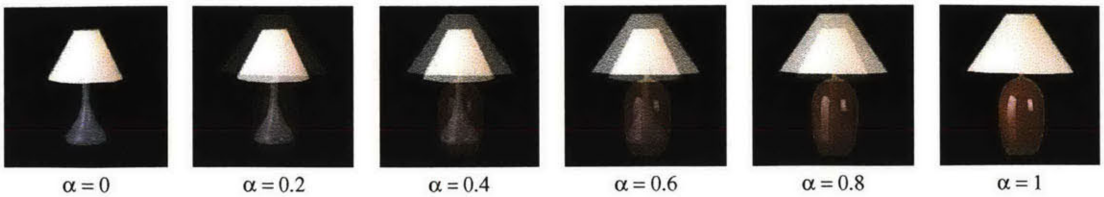


Figure 2-5: Interpolating image textures results in a ghosting or fading effect and does not produce a convincing metamorphosis.

where α is a scalar taking values $0 \leq \alpha \leq 1$ and I_w^1 and I_w^2 are images of the geometrically aligned objects given by

$$\begin{aligned} I_w^1 &= I_1 \circ \alpha d_{21}(x, y) \\ I_w^2 &= I_2 \circ (1 - \alpha) d_{12}(x, y) \end{aligned} \tag{2.7}$$

In Equation (2.7) $d_{ij}(x, y)$ is the deformation field from image I_i to image I_j. To understand Equation (2.6) consider I_m for different values of α. At $\alpha = 0$, $I_w^1 = I_1$, the geometry of I_w^2 is equal to that of the object in image I_1 and $I_m = I_1$. Similarly, at $\alpha = 1$, $I_m = I_2$. For $\alpha = 0.5$ both I_1 and I_2 are deformed half way toward one another and blended equally. The resulting image is sometimes referred to as the *average object* image.

Consider the lamp images of Section 2.1.2. A metamorphosis from the first to the second lamp is displayed in Figure 2-6, where morphed images for different values of α are displayed. Note how the average lamp shares characteristics from both lamp images. The morphed lamp images of Figure 2-6 look more or less like a particular lamp depending on their α values. As will be seen in the following chapters, deformable models generalize Equation (2.6) to morph between images of multiple objects. By combining the appearance of a few example objects they are able to represent a wide range of object appearance.

Thus far we have discussed the general concepts of image warping and morphing but have yet to present a formal image morphing algorithm that supplies a method for computing $d(x, y)$. In the following section we describe the Beier and Neely feature-based image morphing algorithm. We then discuss piecewise image warping in Section 2.4.

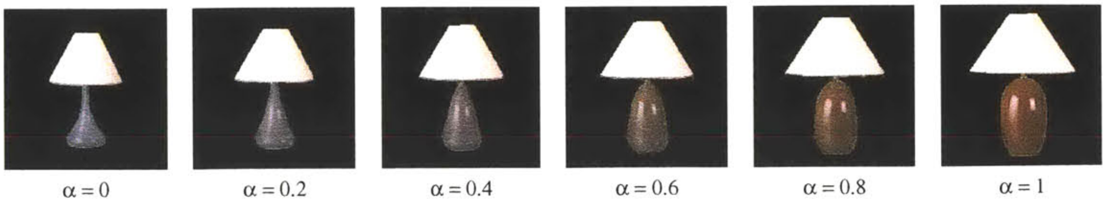


Figure 2-6: Image metamorphosis. A smooth transition between the lamp images of Section 2.1.2 is computed by linearly interpolated both shape and texture. Images from steps along this transition, computed at increasing values of α are displayed.

2.3 Beier and Neely

The Beier and Neely feature-based image morphing algorithm [2] was the first image morphing algorithm presented in the literature. Ever since its introduction image morphing has grown to become a popular field of study in computer graphics and a useful tool in both the vision and graphics communities. More sophisticated, principled image morphing techniques have followed the Beier and Neely algorithm [17]. Nonetheless, due to its simplicity it serves as a good introduction to image morphing. A more principled image morphing algorithm is discussed in the next section.

In the image morphing algorithm of Beier and Neely image correspondence is established using directed line segments, specified in each image. Consider the single line segment feature between two images, illustrated in Figure 2-7. In this figure the line segment $P'Q'$ is in the source image, and PQ is in the geometrically aligned or warped image. The color at each pixel location in the warped image is attained by projecting each 2D pixel location onto the line PQ. The resulting scaled distance u along the line and perpendicular distance v (in pixels) from the line is used to compute the corresponding pixel location in the source image. The color at a point X in the warped image is therefore computed as the color at location X' in the source image, where

$$\begin{aligned} u &= \frac{(X-P)\cdot(Q-P)}{\|Q-P\|^2} \\ v &= \frac{(X-P)\cdot Perpendicular(Q-P)}{\|Q-P\|} \\ X' &= P' + u(Q'-P') + \frac{v\ Perpendicular(Q'-P')}{\|Q'-P'\|} \end{aligned} \tag{2.8}$$

When more than one feature line is specified the point corresponding to X in the source image is computed as a weighted sum between the points X_i' defined by each

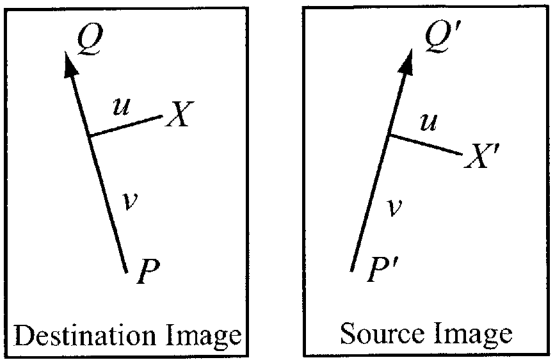


Figure 2-7: A pair of feature line segments in the source and warped (destination) images. [2]

line segment. Each X_i' is weighted using the weight,

$$w_i = \left(\frac{l_i^p}{a + d_i}\right)^b, \tag{2.9}$$

where,

$$\begin{aligned} l_i &= \|Q_i - P_i\| \\ d_i &= \begin{cases} v_i, & 0 \leq u_i \leq 1 \\ \|X - P_i\|, & u_i < 0 \\ \|X - Q_i\|, & u_i > 1 \end{cases} \end{aligned}$$

The constants a, p, and b in (2.9) describe the relative importance of line length and distance, and weighting respectively. Moreover, if b is zero each line is weighted equally independent of length and distance. As discussed in [2], typical values for these constants are,

$$\begin{gathered} a > 0 \\ 0.5 \leq b \leq 2 \\ 0 \leq p \leq 1 \end{gathered} \tag{2.10}$$

The resulting Beier and Neely field warping function $f(X)$, outlined in [17], is defined as

$$f(X) = X + \frac{\sum_i w_i(X_i' - X)}{\sum_i w_i}. \tag{2.11}$$

Comparing Equation (2.11) to Equation (2.3), one finds that $f(X)$ defines the deformation field $d_r(x, y)$.

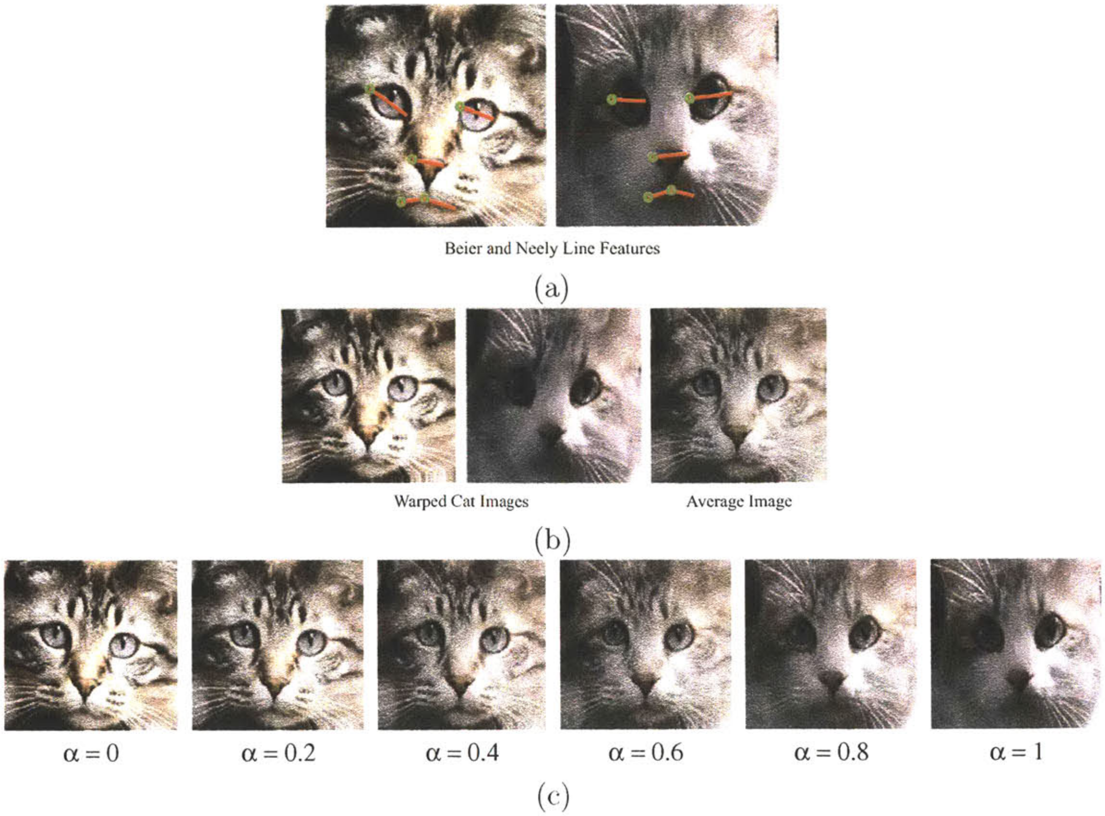


Figure 2-8: Beier and Neely image morphing: (a) manually specified feature lines used to place each object into correspondence, (b) warped source images and average image, (c) metamorphosis between each cat face. Parameter values of $a = 1.0$, $b = 2.0$, $p = 0.1$ were used.

Metamorphosis between two images is defined using Beier and Neely by first specifying a set of directed feature lines between each image. To align each object to the geometry of the morphed image, the vertices of these feature lines are linearly interpolated:

$$v_i = (1 - \alpha)v_i^0 + \alpha v_i^1. \tag{2.12}$$

In the above equation v_i are the line feature vertices of the morphed image, v_i^0 and v_i^1 are the feature vertices specified in each source image and as in (2.6) α is a scalar taking values $0 \leq \alpha \leq 1$. With the features v_i, v_i^0, and v_i^1 each source image is geometrically aligned to the geometry of the morphed image using the field warping function (2.11). Color blending is then applied to the aligned images using Equation (2.6).

A morph between two cat faces, performed using Beier and Neely, is illustrated by Figure 2-8. In the figure, (a) displays the feature lines specified for each cat face, (b) displays the average cat face ($\alpha = 0.5$) and the warped source images used to compute it, and (c) shows the morphed images for various values of α. In Figure 2-8 values of $a = 1$, $b = 0.5$, and $p = 0.5$ were used. These values worked well for this example, however, may vary depending on the situation. Note, whether the Beier and Neely algorithm performs well depends on the values of these parameters and on the choice of appropriate directed line features. We present a more automated, principled image morphing algorithm next.

2.4 Piecewise Image Warping

In the Beier and Neely morphing algorithm $d(x, y)$ was computed via the use of a sparse set of directed line features (see Equation (2.11)). For a point in the morphed image, its corresponding point in each source image was computed by weighting the corresponding point found using each line feature. Although the Beier and Neely algorithm does a good job of producing convincing object metamorphosis (see Figure 2-8) its performance is highly dependent on the right choice of line features and values for a, b, and p. Observing Equation (2.11), one finds that with Beier and Neely $d(x, y) \sim f(x, y)$, where $f(x, y)$ is the field warping function. Since the introduction of the Beier and Neely algorithm, many more principled image morphing techniques have been formulated [17]. These methods try to achieve better approximations to $d(x, y)$ that require less manual intervention.

In this section we introduce the piecewise image warping algorithm. With this algorithm, a few exterior sample points of $d(x, y)$ are manually specified and its inner values are interpolated using a piecewise triangular mesh defined over the convex hull of the sample points. This image warping algorithm is commonly used to construct deformable models and we therefore conclude this chapter with a discussion of this algorithm.

Consider the lamp images of Figure 2-2. To compute a metamorphosis between

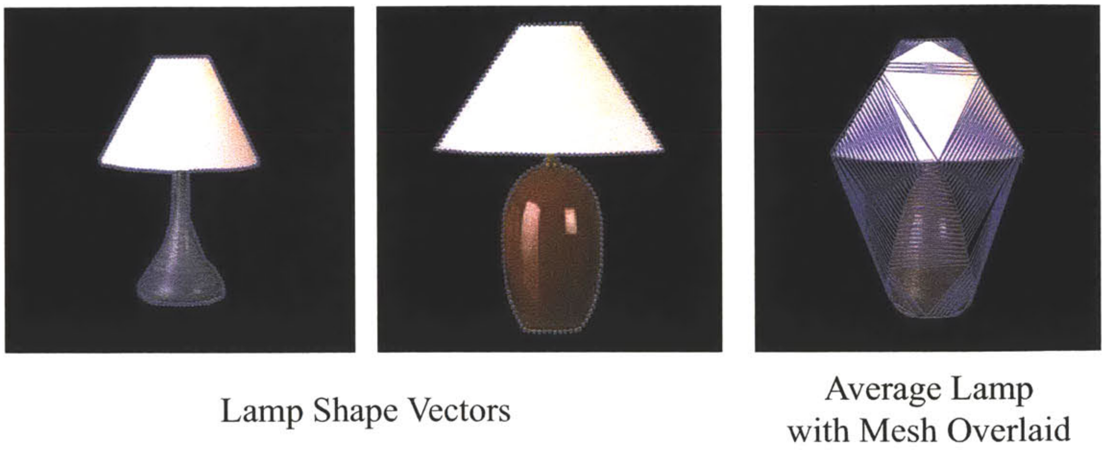


Figure 2-9: Piecewise image warping: (left) 2D feature points that place each lamp into correspondence, (right) average image with mesh overlaid.

these two cat images using peicewise image warping, object feature point correspondences are specified for each lamp image as is done in Figure 2-9. Let v_i^0 and v_i^1 be the feature point correspondences in each lamp source image respectively. Given a value of α, the feature points of the morphed image are computed using Equation (2.12). To compute the deformation field $d_r(x, y)$ a triangular mesh is computed over the convex hull of the feature points of the morphed image using *Delauney triangulation* []. Assuming $\alpha = 0.5$, applying this method gives the triangular mesh displayed in Figure 2-9. This mesh is directly applied to the feature points of each source image, resulting in corresponding triangles between each source image and the morphed image.

Let X be a point in the morphed image located inside triangle $T(A, B, C)$ with vertices A, B, and C. Let $T'(A', B', C')$ be the corresponding triangle in one of the source images and X' the corresponding point inside T'. We can express X as a linear combination of the vertices of T [10],

$$\begin{aligned} X &= A + \beta(B - A) + \gamma(C - A) \\ X &= \alpha A + \beta B + \gamma C \end{aligned} \tag{2.13}$$

where $\alpha = 1 - (\beta + \gamma)$ such that $\alpha + \beta + \gamma = 1$. To find X' we apply the weights α, β, γ, found using Equation (2.13) to the vertices of T',

$$X' = \alpha A' + \beta B' + \gamma C' \tag{2.14}$$

Algorithm 2 Piecewise Image Warping

```
Let $I$ and $I_w$ be the input and warped images respectively, and $M$ be the triangular
mesh defined over the convex hull of the feature points in $I_w$.

for all $X \in I$ do
  for all $T \in M$ do
    Compute $\alpha$, $\beta$, and $\gamma$ using Equation (2.13).
    if $\alpha \geq 0$, $\beta, \gamma \leq 1$ then
      Compute $X'$ using Equation (2.14).
      $I_w(X) = Interpolate(I, X')$.
    end if
  end for
end for
```

where

$$\begin{aligned} \begin{pmatrix} \beta \\ \gamma \end{pmatrix} &= \begin{pmatrix} (B-A) & (C-A) \end{pmatrix}^{-1} (X-A), \\ \alpha &= 1-(\beta+\gamma) \end{aligned}$$

Equations (2.13) and (2.14) collectively define the deformation field $d_r(x, y)$. Each source image is warped to the geometry of the morphed image by considering each point X inside the triangular mesh M defined over the feature points v_i, and using Equation (2.14) to find the corresponding points in each of the source images. This algorithm is presented as Algorithm 2. Note, a point X is inside a triangle T if $\alpha \geq 0$ and $\beta, \gamma \leq 1$. Algorithm 2 has a running time of $O(nk)$, where n is the number of image pixels and k is the number of triangles. Better performance can be achieved by considering the points of each triangle instead of looping over the entire image. Algorithm 2 presents the simpler version of the algorithm for clarity. The image warp can be implemented in real-time by using the mipmap capabilities of OpenGL, the texture of each triangle computed using texture mapping in hardware.

Figure 2-6 displays a metamorphosis between the lamp images of Figure 2-2 using the piecewise image warping algorithm described above. As illustrated by the figure, a smooth metamorphosis is generate via the specification of only a few point correspondences. We will use this algorithm to build the deformable models of the next and following chapters.

Chapter 3

Statistical Shape and Texture Appearance Models

An object's appearance is governed by many physical factors, e.g. illumination, pose, surface properties, and geometry (shape). The construction of a model that parameterizes all such properties of an object's appearance is clearly a challenging task. Many appearance models present in contemporary vision literature try to learn these properties from examples [36, 29, 28, 24, 8]. Eigenfaces [36] is probably the simplest of all these approaches. With this algorithm, many images of an object class are collected, possibly under different imaging conditions, and a linear generative model is constructed using Principle Component Analysis (PCA) [22] (see Figure 3-1). This simple, yet effective representation has proven to be useful for many recognition tasks. As appearance becomes more varied, however, this model requires a lot of examples to be able to faithfully represent the object's appearance. A natural progression from this approach is to try and separate some of the different components that govern object appearance to reduce model complexity.

Shape and texture appearance models achieve a more compact, efficient representation of object appearance by independently modelling object geometry and surface properties/illumination. With these approaches each prototype image is vectorized into shape and texture vectors and linear generative models are constructed independently over each vector space. Shape vectors define the geometry of each prototype

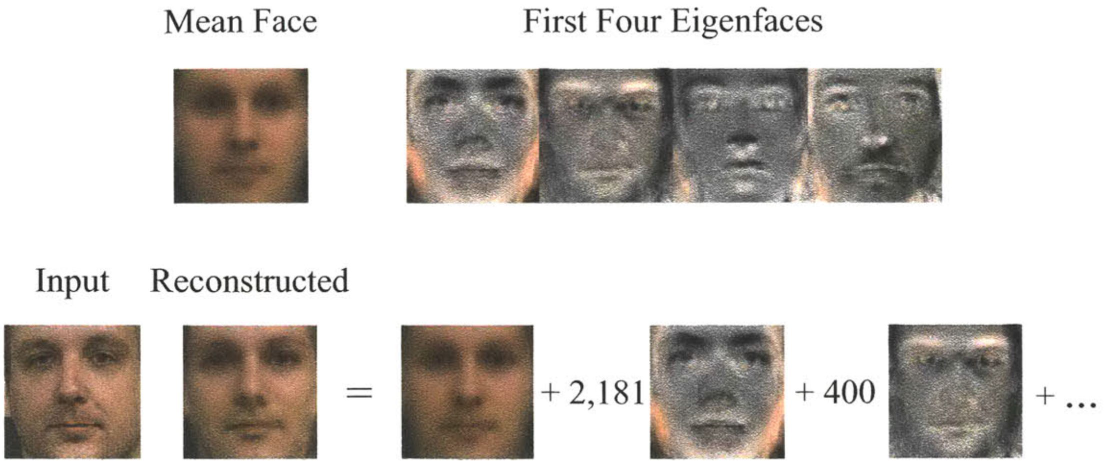
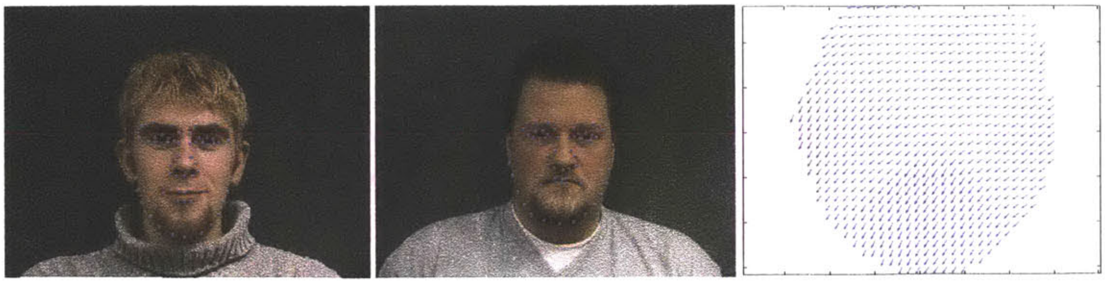


Figure 3-1: Eigenfaces: PCA [22] is applied to the centered face images taken from the IMM face database. The mean face and first four principle axes (eigen faces) are displayed. A subject taken outside of the database is reconstructed using this model. The reconstructed image exhibits error because of shape misalignment between the prototype objects. Shape and texture appearance models separately model object geometry and reflectance.

image and are commonly represented by either point features or optical flow vectors. Texture vectors are "shape free" image representations and are computed by warping each prototype image to a reference model shape. Thus, image intensity is considered under a common coordinate frame independent of object geometry (see Figure 3-3). An image is matched to such a model by optimizing over the shape and texture parameters of the model, minimizing the mean squared error between the model and input images. This is a non-linear optimization problem that has been approached in various ways in the computer vision literature. The methods presented in this chapter have their own algorithms for minimizing this objective function as will be discussed shortly.

The Multidimensional Morphable Model (MMM) [24] and Active Appearance Model (AAM) [8] are two of the most well known shape and texture appearance models in the literature. Both of these approaches build linear generative models of shape and texture to represent object appearance as described above. They differ in how they represent object geometry and match the model. Figure 3-2 displays two face images, each marked with point landmarks that outline the contours of each



First Subject

Second Subject

Computed Flow Field

Figure 3-2: Shape feature points for two subjects, along with computed flow field.

face. Also displayed in this figure is the resulting flow field between each image computed using the piecewise image warping algorithm of Section 2.4. The point shape features and flow vectors are equivalent representations of object geometry from the perspective of shape and texture appearance models: they both are able to define a deformation from each prototype image to a reference model shape. This is the component necessary to vectorize each prototype image into a shape and texture vector.

AAMs define shape as the point landmarks whereas MMMs define shape as the deformation field between each image and the reference image. Each representation has its advantages and disadvantages - we will mention some of them here but will not suggest which one is a better representation as this is beyond the scope of this thesis. An advantage of using flow shape vectors, is that MMMs are able to automatically acquire shape using optical flow techniques [26]. The disadvantage is that flow fields tend to be noisy and often ambiguous in regions that are less textured. The landmark points of AAMs are usually manually specified, however, they have the advantage that the user is able to locally define which correspondences bear more importance and the computed deformation fields are usually more structured and less noisy than optical flow.

The MMM and AAM are also distinguished by the optimization methods used to fit the model. MMMs use stochastic gradient decent to fit an image to the model. Stochastic gradient descent is a principled matching algorithm that is able to optimally fit the model, however, it is computationally expensive and slow. Alternatively,

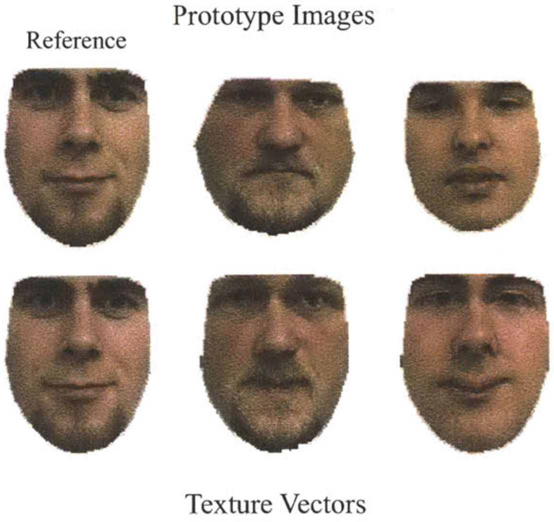
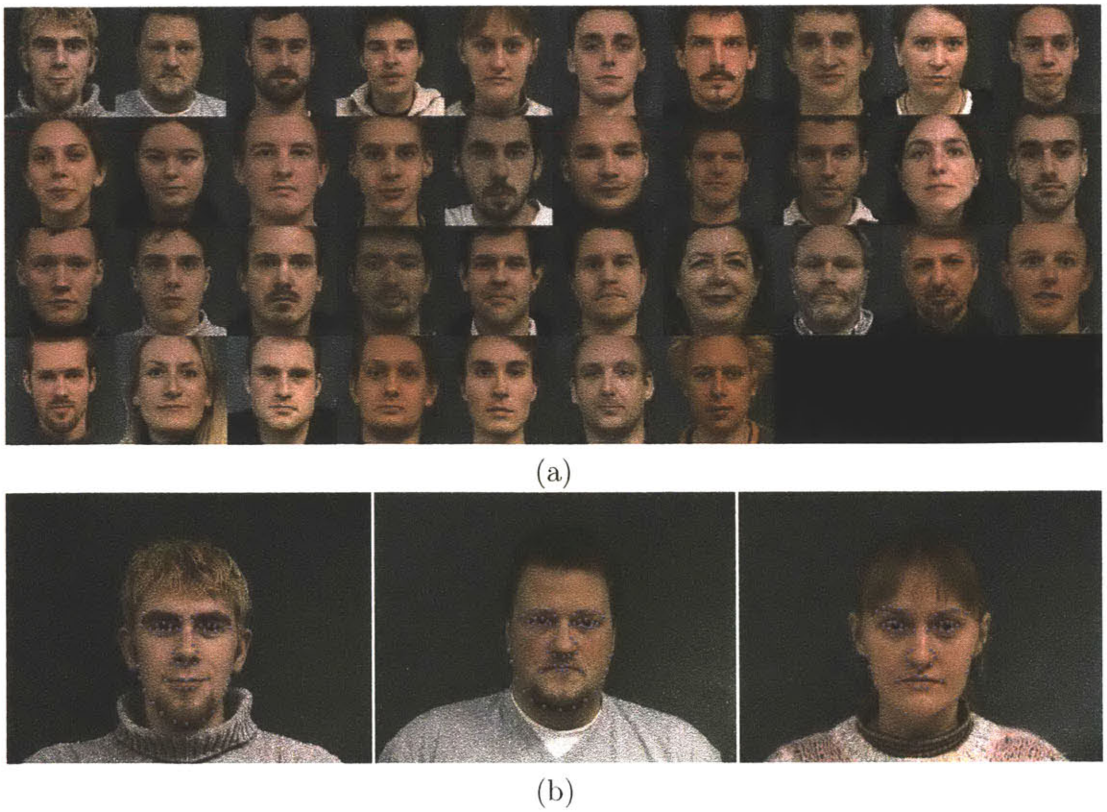


Figure 3-3: Image texture: each prototype is warped to the geometry of a model reference. This results in a "shape free" representation of each prototype image, the intensities of each object considered under a common coordinate frame.

AAMs attempt to efficiently match the model by learning a linear relationship of how the objective function changes with respect to the parameters of the model. This is done by perturbing the model in known directions and recording how the model objective changes. In order to achieve efficiency AAMs assume that the learned relationship is constant and independent of the value of the model parameters. This assumption is not true in general [27] and may lead to non-optimal fits. Nevertheless, this algorithm gives good results in practice and is much faster than the stochastic gradient decent algorithm present in MMMs.

In this chapter we formally present the MMM and AAM algorithms and provide examples. In Section 3.1 we will introduce notation that we will use to describe both algorithms under a unifying context. The MMM algorithm is detailed in Section 3.2 and the AAM algorithm in Section 3.3. We show results for both algorithms using the IMM face database [34], a collection of 37 face, 320×240, annotated color images (see Figure 3-4). In Chapter 6 we extend both of these approaches to 4D and show how objects with complex surface reflectance and geometry can be easily modelled across pose with these methods using light fields.


(a)

(b)

Figure 3-4: IMM face database [34]: (a) 37 320 × 240 color face images, (b) example shape annotations.

3.1 Shape and Texture Appearance Models

Let I_i, $i = 1, .., N$ be a set of N prototype images of an object class. We construct a shape and texture appearance model from these images by separating each image into shape and texture vectors and then modelling the variation of the prototype images in each of these vector spaces using PCA. For each image we define a shape vector $\mathbf{x}_i$ that defines the geometry of each prototype and places each prototype into correspondence with a reference object having shape $\mathbf{x}_{ref}$. We compute the texture of each example $\mathbf{g}_i$ by warping each image to have the reference shape:

$$\mathbf{g}_i = I_i \circ W(\mathbf{x}_i, \mathbf{x}_{ref}), \quad i = 1, ..., N \tag{3.1}$$

where $W(\mathbf{x}_1, \mathbf{x}_2)$ is a function that given shapes $\mathbf{x}_1, \mathbf{x}_2$ returns the deformation field $d_{21}(x, y)$ that specifies for each pixel in image I_2 (having shape $\mathbf{x}_2$) the corresponding pixel in image I_1.

Given $\mathbf{x}_i$, $\mathbf{g}_i$ for each prototype we model the variation in each of these vector spaces using PCA:

$$\begin{aligned} \mathbf{x} &= \bar{\mathbf{x}} + \mathbf{P}_s\mathbf{b}_s \\ \mathbf{g} &= \bar{\mathbf{g}} + \mathbf{P}_g\mathbf{b}_g \end{aligned}, \tag{3.2}$$

where $\mathbf{P}_s$, $\mathbf{P}_g$ are matrices containing as their columns the $d \leq N$ eigenvectors of the shape and texture covariance matrices and represent the principle axis of variation away from the respective means, $\mathbf{x}$ and $\mathbf{g}$ are shape and texture vectors of an input image and $\mathbf{b}_s$ and $\mathbf{b}_g$ are the shape and texture parameters of the model.

Note the principle axes do not necessarily have a meaningful interpretation other than that they are the axes directed along the highest variation in the data set. For example, one might expect that if the data contained many smiling versus neutral faces that one of the first principle axes may capture this variation. This may or may not be the case, however, with this approach there is no guarantee of the axes bearing such an interpretation. Similar methods have learned a mapping between the PCA basis and class specific labels to enforce such a meaningful parameter space [4]. Nevertheless, the goal of these models is to faithfully represent the appearance manifold in each of the shape and texture vector spaces, which is accomplished using PCA. Given an input image these models may be used to extract the object's shape and texture by matching the image to a point on this bi-linear appearance manifold. In turn this information may be used as low-level input to a high-level vision task (e.g. recognition).

To make the model more flexible, we also parameterize Equation (3.2) to handle arbitrary affine transformation and global illumination,

$$\begin{aligned} \mathbf{x} &= S_t(\bar{\mathbf{x}} + \mathbf{P}_s\mathbf{b}_s) \\ \mathbf{g} &= T_u(\bar{\mathbf{g}} + \mathbf{P}_g\mathbf{b}_g) \end{aligned}, \tag{3.3}$$

where S_t is a function that applies a rigid body transformation to the model shape according to a pose parameter vector $\mathbf{t}$ and T_u is a function which scales and shifts the model texture to an arbitrary contrast and brightness using an illumination parameter vector $\mathbf{u}$.

Using Equation (3.3) the bi-linear shape and texture appearance manifold is defined as,

$$I_m(\mathbf{b}_s, \mathbf{b}_g, \mathbf{t}, \mathbf{u}) = t(\mathbf{b}_g, \mathbf{u}) \circ W(\mathbf{x}_{ref}, \mathbf{x}(\mathbf{b}_s, \mathbf{t})). \tag{3.4}$$

As seen above, a point $\mathbf{p} = (\mathbf{b}_s^T, \mathbf{b}_g^T, \mathbf{t}^T, \mathbf{u}^T)^T$ on this bi-linear appearance manifold maps to a model image I_m of an object contained in the object class. Using Equation (3.4) the model can represent a wide range of object appearance by interpolating the shape and texture of the prototype images. A novel input image, I_s, is matched to this manifold by minimizing the mean squared error between the model and input images,

$$E(I_s, \mathbf{b}_s, \mathbf{b}_g, \mathbf{t}, \mathbf{u}) = \|I_s - I_m(\mathbf{b}_s, \mathbf{b}_g, \mathbf{t}, \mathbf{u})\|^2. \tag{3.5}$$

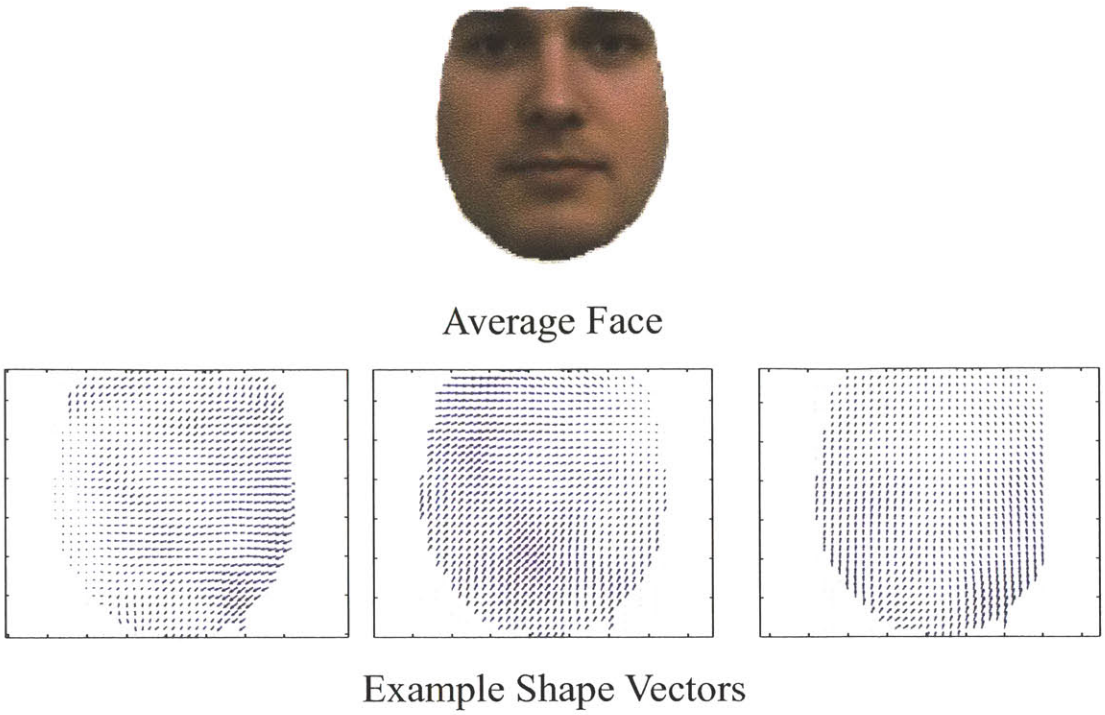
It is often more convenient to work in the coordinate frame of the reference object, so we re-write Equation (3.5) as

$$E(I_s, \mathbf{b}_s, \mathbf{b}_g, \mathbf{t}, \mathbf{u}) = \|I_s \circ W(\mathbf{x}(\mathbf{b}_s, \mathbf{t}), \mathbf{x}_{ref}) - \mathbf{g}(\mathbf{b}_g, \mathbf{u})\|^2. \tag{3.6}$$

Equation (3.6) defines a non-linear objective function that is difficult to optimize in general. In the following sections we will discuss techniques for optimizing this objective and provide example matches to deformable models of the human face. In each section we will be using the face prototype images displayed in Figure 3-4 to construct the model.

3.2 Multidimensional Morphable Models

Multidimensional Morphable Models [24] define shape vectors using optical flow. Although the choice of a reference object is arbitrary, they compute the *average object* from the prototype images and use this image for the reference. In some sense the average object is the optimal choice, being that it is equidistant from all prototype images in shape and texture. As discussed in Section 2.2 the average object is computed as the object whose blending parameters α are of equal weight (i.e. $\alpha = 1/N$ for multiple objects). Since MMMs use optical flow based shape vectors the shape of the average object cannot be directly computed, since the average object is not



Average Face

Example Shape Vectors

Figure 3-5: Average face computed using MMM (top). Example model shape vectors (bottom).

known a priori with optical flow. Instead they define an iterative algorithm, in which an object is chosen as the reference, a set of shape and texture vectors are defined, and then the average shape and texture is computed thus defining a new reference object. This algorithm is then iterated until convergence. For convenience, we formally present this algorithm as Algorithm 3. The average face along with some example flow fields obtained using the prototype faces of Figure 3-4 is displayed in Figure 3-5.

Algorithm 3 provides for each image a shape and texture vector $\mathbf{x}_i$, $\mathbf{g}_i$, $i = 1, ..., N$. For a MMM by definition we have,

$$\mathbf{g}_i = I_i \circ \mathbf{x}_i. \tag{3.7}$$

Comparing Equation (3.7) to Equation (3.1) one finds that for MMMs we have

$$W(\mathbf{x}_i, \mathbf{x}_{ref}) = \mathbf{x}_i. \tag{3.8}$$

Using Equation (3.7) we can express the shape and texture appearance manifold of

Algorithm 3 Compute Average Image [24]

```
Let $I_1, ..., I_n$ be a set of prototype images.
Select an arbitrary image $I_i$ as the reference image $I_{ref}$
repeat
  for all $I_i$ do
    Compute correspondence fields $\mathbf{x}_i$ between $I_{ref}$ and $I_i$ using optical flow.
    Backwards warp $I_i$ onto $I_{ref}$ using $\mathbf{x}_i$.
  end for
  Compute the average over all $\mathbf{x}_i$ and $\mathbf{g}_i$.
  Forward warp $\mathbf{g}_{average}$ using $\mathbf{x}_{average}$ to create $I_{average}$.
  Convergence test: is $I_{average} - I_{ref} < limit$ ?
  Copy $I_{average}$ to $I_{ref}$
until convergence
```

Equation (3.4) in terms of an MMM as follows

$$I_m(\mathbf{b}_s, \mathbf{b}_g, \mathbf{t}, \mathbf{u}) = t(\mathbf{b}_g, \mathbf{u}) \circ inv(\mathbf{x}(\mathbf{b}_s, \mathbf{t})), \tag{3.9}$$

where $inv(\mathbf{x})$ is a function that flips the deformation field $\mathbf{x}$ to point in the opposite direction. Since the mapping may not be one-to-one some of the values of $inv(\mathbf{x})$ may need to be interpolated from neighboring values of the inverted deformation field.

Alternatively, once the reference image I_{ref} of Algorithm 3 is computed we can compute optical flow between each prototype image and I_{ref} to result in shape vectors $\mathbf{x}_i^r = inv(\mathbf{x}_i)$. This method would avoid the need to interpolate the inverted flow fields, however, the resulting PCA space computed over these vectors would be inefficient, since each $\mathbf{x}_i^r$ is in a different coordinate frame: $\mathbf{x}_i$ specifies for each point in the reference image the corresponding point in I_i. The flow vectors of each $\mathbf{x}_i$ are therefore aligned to the geometry of the reference image. In contrast, each $\mathbf{x}_i^r$ is aligned to the geometry of the different prototypes. Although, it is possible to form a PCA space over $\mathbf{x}_i^r$, their misalignment would lead to in-accuracy and inflation of the computed PCA space which is undesirable. Instead we choose to maintain the PCA space over the original $\mathbf{x}_i$ and forward warp the model texture vector to form the model image:

$$I_m(\mathbf{b}_s, \mathbf{b}_g, \mathbf{t}, \mathbf{u}) = t(\mathbf{b}_g, \mathbf{u}) \circ_f \mathbf{x}(\mathbf{b}_s, \mathbf{t}). \tag{3.10}$$

where the subscript f is used to denote the forward warping operation. The use of

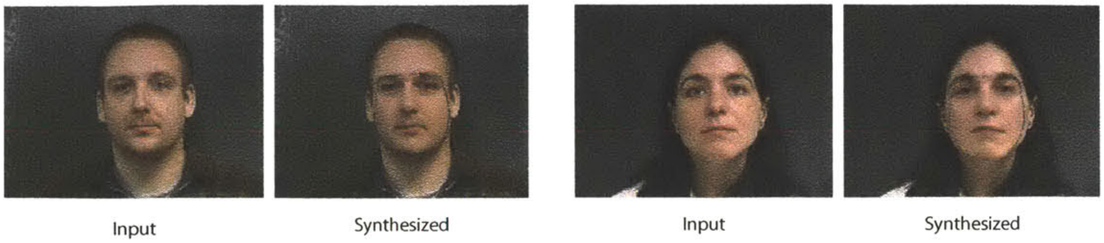


Figure 3-6: Multidimensional morphable model (MMM) built from 35 faces of the IMM face database, optimized over two novel input images. The MMM is able to faithfully model the appearance of novel subjects.

forward warping will slightly blur the synthesized model image, however, we believe that this is a fair tradeoff for efficiency in the model.

To match a novel input image to a MMM we adopt the optical flow based matching algorithm introduced by Beymer et. al [3]. This algorithm is sub-optimal to the stochastic gradient descent algorithm of [24], however, it is efficient, as optical flow is relatively cheap to compute, and performs fairly well. With this method we directly minimize the objective function (3.6) by computing optical flow between the input image I_s and I_{ref} to yield shape $\mathbf{x}_s$ and texture $\mathbf{g}_s$ given by

$$\mathbf{g}_s = \mathbf{I}_s \circ \mathbf{x}_s. \tag{3.11}$$

We can then match I_s onto the appearance manifold by solving the following linear relationships using linear-least squares

$$\begin{array}{c} \mathbf{x}_s = \bar{\mathbf{x}} + \mathbf{P}_s \mathbf{b}_s \\ \mathbf{g}_s = \bar{\mathbf{g}} + \mathbf{P}_g \mathbf{b}_g \end{array}. \tag{3.12}$$

Solving Equation (3.12) for $\mathbf{b}_s$ and $\mathbf{b}_g$ gives

$$\begin{array}{c} \mathbf{b}_s = \mathbf{P}_s^+ (\mathbf{x}_s - \bar{\mathbf{x}}) \\ \mathbf{b}_g = \mathbf{P}_g^+ (\mathbf{g}_s - \bar{\mathbf{g}}) \end{array}. \tag{3.13}$$

Note in the above solution we do not solve directly for the parameters $\mathbf{t}, \mathbf{u}$ since this linear-least squares solution automatically accounts for such variation: the optical flow field $\mathbf{x}_s$ takes into account any affine transform and any global lighting variation is solved for directly and is represented in $\mathbf{b}_g$. In contrast to the stochastic gradient descent solution of [24] the above method assumes that any differences in affine

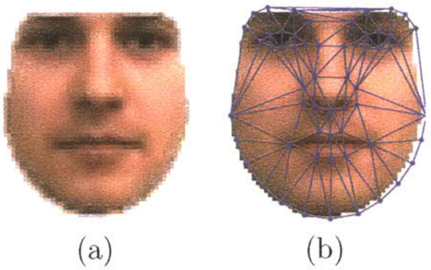


Figure 3-7: Average face computed using AAM displayed (a) by itself and (b) with the reference shape and mesh overlaid.

alignment or global lighting are relatively small, namely they are those captured by the PCA model.

Figure 3-6 illustrates matches to images taken out of the database displayed in Figure 3-4 using the above matching algorithm. For each input image, the location of the face is manually specified. The online optical flow computation is sensitive to scene clutter; as such a segmentation mask for each input is also provided. As illustrated by the figure, the MMM is able to faithfully model the appearance of novel subjects, although there is some jitter in the matches as a result of noise in the optical flow fields. In the next section we will discuss the AAM; we will show how a shape and texture model can be generated with manually specified point features. We then show how to optimize Equation (3.6) using a direct search algorithm, in which a constant relationship between the change in this error objective and the model parameters is learned and used to match the model in real-time.

3.3 Active Appearance Models

An AAM [8] defines shape as a set of 2D point features that are manually specified along the contours of an object. For each prototype a shape vector, $\mathbf{x}'_i$, is defined by placing the n, x, y coordinates of each point into a vector,

$$\mathbf{x}'_i = (x_1, x_2, ..., x_n, y_1, y_2, ..., y_n)^T. \tag{3.14}$$

Figure 3-4(b) displays example shape feature point vectors for the prototype faces of the IMM face database. A plot of all shape vectors super-imposed onto one another is displayed in Figure 3-8. As seen from the figure there is a large variation in the shape vectors across example faces. It is important to note, however, that the shape variation displayed in this figure is mainly due to global in-plane pose change as oppose to the non-rigid shape changes that we are interested in modelling (e.g. the differently shaped mouths, noses, and heads). We therefore wish to normalize each shape such that they are all defined under the same global pose or coordinate frame. To do this we employ Procrustes Analysis [18]. This algorithm is described as Algorithm 4. The normalized shapes found using Procustes Analysis are displayed in Figure 3-8. As illustrated by the figure the aligned shapes are mostly in the same global orientation and scale and centered with respect to one another. The variation in the aligned shapes is mainly due to the interesting non-rigid shape changes. Note this shape normalization step is not necessary, however, allows us to build a more efficient, compact shape model.

From the aligned shapes we compute the reference shape, $\mathbf{x}_{ref}$ as follows,

$$\mathbf{x}_{ref} = \mathbf{M}_{\alpha}\bar{\mathbf{x}}_i, \tag{3.15}$$

where $\mathbf{x}_i$ is used to denote normalized shapes and $\mathbf{M}_{\alpha}$ is an affine transform matrix that scales and shifts $\bar{\mathbf{x}}_i$ into image coordinates.

Given shapes $\mathbf{x}'_i$, $i = 1, ..., N$ and reference shape $\mathbf{x}_{ref}$ we compute the texture, $\mathbf{g}'_i$ of each prototype using the piecewise image warping algorithm of Section 2.4,

$$\mathbf{g}'_i = I_i \circ W(\mathbf{x}'_i, \mathbf{x}_{ref}), \quad i = 1, ..., N \ , \tag{3.16}$$

where $W(\mathbf{x}'_i, \mathbf{x}_{ref})$ is defined using Equations (2.13) and (2.14). Figure 3-7 displays the average face computed using the prototypes of Figure 3-4. In this figure $\mathbf{x}_{ref}$ is displayed superimposed on the average face along with the triangular mesh used to perform the piecewise warping. This mesh was computed using Deluanay triangulation. Similar to shape we wish to normalize the texture of each prototype to be under the same global illumination, thus allowing a more efficient representation of texture variation. To do this we employ the texture normalization algorithm described as Algorithm 5. This algorithm normalizes each texture vector such that their mean vector has zero mean and unit norm.

Algorithm 4 Procrustes Analysis [10]

```
Let $\mathbf{x}_i = (\mathbf{x}_i^x, \mathbf{x}_i^y)$, $i = 1, ..., n$ be a set of 2D input shapes.

Center each shape to the origin $(0, 0)$.
Pick a reference shape: $\mathbf{x}_{ref} = \mathbf{x}_j$, $j \in \{1, ..., n\}$.
repeat
  for $i = 1, ..., n$ do
    Align shape to reference: $\mathbf{x}_i = align(\mathbf{x}_i, \mathbf{x}_{ref})$.
  end for
  Compute average over all $\mathbf{x}_i$ to give $\mathbf{x}_{average}$.
  Align average to reference: $\mathbf{x}_{average} = align(\mathbf{x}_{average}, \mathbf{x}_{ref})$.
  Normalize average shape: $\mathbf{x}_{average} = \mathbf{x}_{average}/||\mathbf{x}_{average}||$.
  Set $\mathbf{x}_{prevref} = \mathbf{x}_{ref}$.
  Set $\mathbf{x}_{ref} = \mathbf{x}_{average}$.
until $||\mathbf{x}_{average} - \mathbf{x}_{prevref}|| < threshold$
for $i = 1, ..., n$ do
  Project $\mathbf{x}_i$ onto the tangent space of the reference shape: $\mathbf{x}_i = \mathbf{x}_i/(\mathbf{x}_i \cdot \mathbf{x}_{ref})$.
end for

function $\mathbf{x}_a = align(\mathbf{x}_1, \mathbf{x}_2)$
Compute $a = (\mathbf{x}_1 \cdot \mathbf{x}_2)/||\mathbf{x}_1||^2$.
Compute $S_x = ||\mathbf{x}_1^x \cdot \mathbf{x}_2^y||^2$.
Compute $S_y = ||\mathbf{x}_1^y \cdot \mathbf{x}_2^x||^2$.
Compute $b = (S_x - S_y)/||\mathbf{x}_1||^2$.
Define $R = \begin{pmatrix} a & -b \\ b & a \end{pmatrix}$.
Align shape: $\mathbf{x}_a = R\mathbf{x}_2$.
```

With the normalized shape and texture vectors $\mathbf{x}_i$, $\mathbf{g}_i$, $i = 1, ..., N$, the model is defined using Equation (3.2). As there may exist a correlation between texture and shape, active appearance models define a more compact model of appearance by applying PCA on the concatenated shape and texture parameter vectors of each prototype image:

$$\mathbf{b} = \begin{pmatrix} \mathbf{W}_s \mathbf{b}_s \\ \mathbf{b}_g \end{pmatrix} = \mathbf{P}_c \mathbf{c} = \begin{pmatrix} \mathbf{P}_{cs} \\ \mathbf{P}_{cg} \end{pmatrix} \mathbf{c}, \tag{3.17}$$

where $\mathbf{W}_s$ is a matrix that comensurates the variation in shape and texture when performing the combined texture-shape PCA. In our experiments we use $\mathbf{W}_s = r\mathbf{I}$ where $r = \sqrt{\sigma_s^2/\sigma_g^2}$ and σ_s^2 and σ_g^2 represent the total variance of the normalized shape and texture vectors.

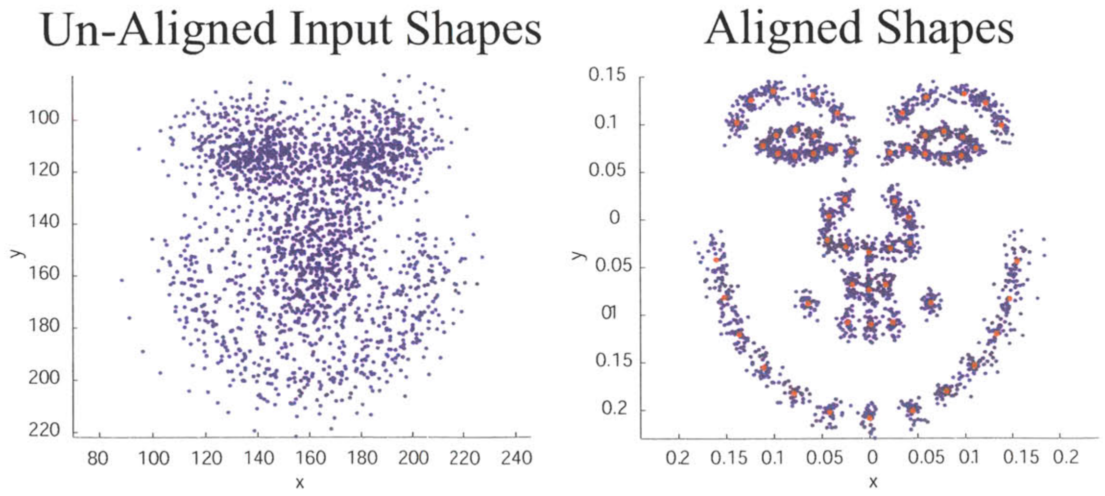


Figure 3-8: Procrustes analysis aligns the input shapes into a common coordinate frame such that each shape is centered and under the same orientation and scale. The computed reference shape is displayed in red, overlaid a top of the aligned shapes. [34]

This results in a combined texture-shape PCA space,

$$\begin{aligned} \mathbf{x} &= \bar{\mathbf{x}} + \mathbf{Q}_s\mathbf{c} \\ \mathbf{g} &= \bar{\mathbf{g}} + \mathbf{Q}_g\mathbf{c} \end{aligned}, \tag{3.18}$$

where,

$$\begin{aligned} \mathbf{Q}_s &= \mathbf{P}_s\mathbf{W}_s^{-1}\mathbf{P}_{cs} \\ \mathbf{Q}_g &= \mathbf{P}_g\mathbf{P}_{cg} \end{aligned}. \tag{3.19}$$

As in Equation (3.3), Equation (3.18) is parameterized to handle arbitrary affine transformation and global illumination,

$$\begin{aligned} \mathbf{x} &= S_t(\bar{\mathbf{x}} + \mathbf{Q}_s\mathbf{c}) \\ \mathbf{g} &= T_u(\bar{\mathbf{g}} + \mathbf{Q}_g\mathbf{c}) \end{aligned}. \tag{3.20}$$

In this equation, $\mathbf{t}$ and $\mathbf{u}$ for an AAM are defined as,

$$\begin{aligned} \mathbf{t} &= \begin{pmatrix} s\ cos(\theta) - 1 \\ s\ sin(\theta) \\ t_x \\ t_y \end{pmatrix}, \\ \mathbf{u} &= \begin{pmatrix} u_0 - 1 \\ u_1 \end{pmatrix} \end{aligned} \tag{3.21}$$

Algorithm 5 Texture Normalization [10]

```
Let $\mathbf{g}_i$, $i = 1, ..., n$ be a set of input textures.

Pick a reference texture: $\mathbf{g}_{ref} = \mathbf{g}_j$, $j \in \{1, ..., n\}$.
repeat
  Normalize the reference texture to have zero mean and unit variance.
  for $i = 1, ..., n$ do
    Compute $\alpha = \mathbf{g}_i \cdot \mathbf{g}_{ref}$.
    Compute $\beta = \bar{\mathbf{g}}_i$.
    Normalize texture: $\mathbf{g}_i = (\mathbf{g}_i - \beta)/\alpha$.
  end for
  Compute average over all normalized textures to form $\mathbf{g}_{average}$.
  Set $\mathbf{g}_{prevref} = \mathbf{g}_{ref}$.
  Set $\mathbf{g}_{ref} = \mathbf{g}_{average}$.
until $||\mathbf{g}_{prevref} - \mathbf{g}_{ref}|| < threshold$
```

where s is a scalar that scales the shape vector, θ is an angle that defines the in-plane orientation of the shape and t_x, t_y define a horizontal and vertical shift of the shape respectively. Each of these parameters align the model shape to the global pose of the input image. In the above equation u_0 and u_1 scale and shift the texture values to match the global illumination of the input. Note that both $\mathbf{t}$ and $\mathbf{u}$ exhibit a -1 in their first coordinate. This is done so that the identity transform of each parameter vector is given by $\mathbf{p}, \mathbf{u} = 0$.

The shape and texture appearance manifold of an AAM is given by Equation (3.4) parameterized over the combined texture-shape PCA space,

$$I_m(\mathbf{c}, \mathbf{t}, \mathbf{u}) = \mathbf{g}(\mathbf{c}, \mathbf{u}) \circ W(\mathbf{x}_{ref}, \mathbf{x}(\mathbf{c}, \mathbf{t})). \tag{3.22}$$

To match a novel input image I_s to this manifold AAMs employ a direct search algorithm we describe next.

Let I_s be the input image with hypothesized shape and texture given by $\mathbf{x}(\mathbf{c}, \mathbf{t})$ and $\mathbf{g}(\mathbf{c}, \mathbf{u})$, where the parameters $\mathbf{c}, \mathbf{t}, \mathbf{u}$ can be initialized to any point in the convex hull of the examples. Note, here $\mathbf{t}$ is chosen such that the model template defined by $\mathbf{x}(\mathbf{c}, \mathbf{t})$ and $\mathbf{g}(\mathbf{c}, \mathbf{u})$ lies mostly over the input object in the image I_s. In the case of faces, this initial alignment can be provided by a face detector for example. The error between the current model fit and the input image is given by an altered version of

Variables	Perturbations
x, y	±5% and ±10% of the height and width of the reference shape
θ	±5, ±15 degrees
scale	±5%, ±15%
c_{1-k}	±0.25, ±0.5 standard deviations

Table 3.1: Perturbation scheme used to compute AAM Jacobian. [34]

Equation (3.6), re-defined to function over the combined texture-shape PCA space:

$$E(\mathbf{p}) = \|I_s \circ W(\mathbf{x}(\mathbf{c}, \mathbf{t}), \mathbf{x}_{ref}) - \mathbf{g}(\mathbf{c}, \mathbf{u})\|^2, \tag{3.23}$$

where $\mathbf{p} = (\mathbf{c}^T, \mathbf{t}^T, \mathbf{u}^T)^T$ is a point on the texture-shape appearance manifold. We can re-express Equation (3.23) as follows,

$$E(\mathbf{p}) = \mathbf{r}(\mathbf{p}) \cdot \mathbf{r}(\mathbf{p}) \tag{3.24}$$

where the residual vector $\mathbf{r}(\mathbf{p})$ is given by

$$\mathbf{r}(\mathbf{p}) = I_s \circ W(\mathbf{x}(\mathbf{c}, \mathbf{t}), \mathbf{x}_{ref}) - \mathbf{g}(\mathbf{c}, \mathbf{u}).$$

We wish to find a $\delta\mathbf{p}$ that minimizes the objective of Equation (3.24). We do so by taking a first order Taylor expansion of the residual function $\mathbf{r}(\mathbf{p} + \delta\mathbf{p})$ [10]:

$$\mathbf{r}(\mathbf{p} + \delta\mathbf{p}) = \mathbf{r}(\mathbf{p}) + \frac{d\mathbf{r}(\mathbf{p})}{d\mathbf{p}}\delta\mathbf{p} \tag{3.25}$$

Setting the above expression equal 0 and solving for $\delta\mathbf{p}$ one finds,

$$\delta\mathbf{p} = -\mathbf{R}\mathbf{r}(\mathbf{p}) \tag{3.26}$$

where

$$\mathbf{R} = \left(\frac{d\mathbf{r}(\mathbf{p})}{d\mathbf{p}}\right)^+$$

To implement the direct search we learn the Jacobian matrix $\mathbf{J} = \frac{d\mathbf{r}(\mathbf{p})}{d\mathbf{p}}$ by starting from each prototype image, perturbing the model in known directions and then averaging over the Jacobian matrices found for each prototype. One may think of the direct search as a pattern matching algorithm. Each column of $\mathbf{R} = \frac{d\mathbf{p}}{d\mathbf{r}(\mathbf{p})}$ is a difference image that records how a given parameter of $\mathbf{p}$ changes with respect to

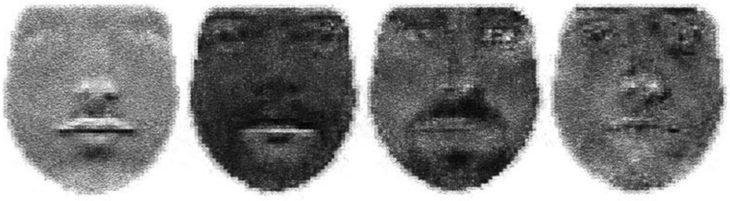


Figure 3-9: First four columns of AAM Jacobian.

$\mathbf{r}(\mathbf{p})$. The dot product of Equation (3.26) compares $\mathbf{r}(\mathbf{p})$ to the columns of $\mathbf{R}$ and the parameters of similar columns in $\mathbf{R}$ are given more emphasis in $\delta\mathbf{p}$. The Jacobian matrix for the faces of Figure 3-4 was computed using the perturbation scheme outlined by Table 3.1. The first few columns of this matrix are displayed in Figure 3-9. As illustrated by the Figure, the columns of the Jacobian matrix are difference images that record how the residual function changes with respect to the model parameters. Note a key assumption of the above algorithm is that $\mathbf{J}$ is constant and independent of the values of $\mathbf{p}$. This assumption is false in general [27] and may lead to non-optimal matches, however, as will be seen shortly this algorithm is efficient and gives good results in practice.

The complete direct search algorithm is presented as Algorithm 6. This algorithm utilizes relationship (3.26) to perform a gradient decent to minimize the objective (3.23). It can be implemented in real-time, using graphics hardware to realize the piecewise image warps [34]. Note Algorithm 6 does not solve for global lighting $\mathbf{u}$ and instead normalizes the model and image texture vectors to have unit variance and zero mean. After the model is fit, we compare the converged model texture with the image texture and approximate $\mathbf{u}$ as the transform which scales and shifts the model texture to have the same maximum and minimum values as the input texture. Alternatively, linear-least squares can be used to solve for $\mathbf{u}$, however, we found the above method to work best in our experiments.

We implemented Algorithm 6 in MATLAB and example matches for novel images outside of the model, built using the face database of Figure 3-4 are displayed in Figure 3-10. In the figure, intermediate iterations are displayed. In each match, the search was initialized from the average face. As illustrated by the figure, the model is

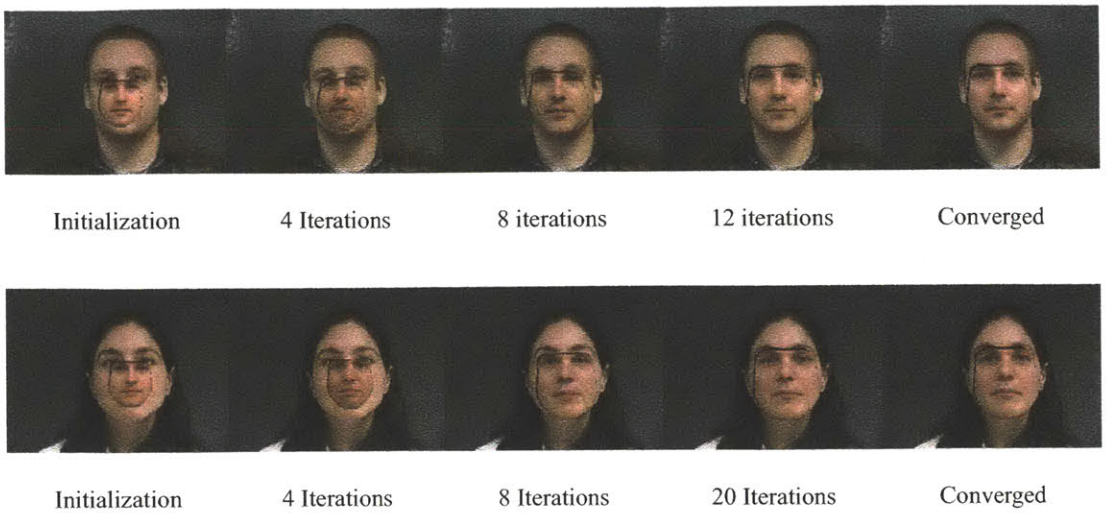


Figure 3-10: Active appearance model (AAM) built from 35 faces of the IMM face database, optimized over two novel face images. Intermediate iterations are displayed for each optimization. The AAM is able to fit the image from rough starting points and converges to a good fit in a few iterations.

able to fit the image from rough starting points and converges to a good fit in a few iterations. In our un-optimized code, convergence is usually declared in under one second.

In both the example fits of this and the previous section the model was built using frontal faces. Extending these models to handle 3D pose variation in 2D is difficult, since out-of-plane pose change in 2-dimensions results in non-linear differences in appearance that is poorly modelled using PCA. Deformable models in 4 dimensions are able to easily model objects with complex geometry and surface reflectance across 3D pose. In order to understand these models, we need to first introduce the concept of image based rendering, in specific light field rendering. We discuss light fields in the next chapter. In following chapters we show how one can construct a deformable model over light fields of objects and demonstrate the advantages of such models over existing 2D and 3D shape and texture appearance models.

Algorithm 6 Direct Search [10]

```
Let $I_s$ be the input image we wish to match.

Set $\mathbf{p} = \mathbf{p}_0$
Evaluate $\delta\mathbf{g} = Residual(I_s, \mathbf{p})$
**repeat**
  Compute error $E_0 = |\delta\mathbf{g}|^2$
  Evaluate $\delta\mathbf{p} = -\mathbf{R}\delta\mathbf{g}$
  Update parameters, $\mathbf{p}_1 = \mathbf{p} + \delta\mathbf{p}$
  Evaluate $\delta\mathbf{g} = Residual(I_s, \mathbf{p}_1)$
  Compute error at new $\mathbf{p}$ value: $E = |\delta\mathbf{g}|^2$
  **if** $|E - E_0| \geq 0$ **then**
    Set $i = 0$, $k = 1.5$
    **while** $|E - E_0| \geq 0$, $i \leq n$ **do**
      Set $\mathbf{p}_1 = \mathbf{p} + k\delta\mathbf{p}$
      Set $i = i + 1$
      Evaluate $\delta\mathbf{g} = Residual(I_s, \mathbf{p}_1)$
      Compute error $E = |\delta\mathbf{g}|^2$
      **if** $k > 1$ **then**
        Set $k = 0.5$
      **else**
        Set $k = k/2$
      **end if**
    **end while**
  **end if**
  **if** $|E - E_0| < 0$ **then**
    Set $\mathbf{p} = \mathbf{p}_1$
  **end if**
**until** $|E - E_0| \geq 0$

**function** $\delta\mathbf{g} = Residual(I_s, \mathbf{p})$
$\mathbf{x}_s = S_t(\bar{\mathbf{x}} + \mathbf{P}_s\mathbf{b}_s)$
$\mathbf{g}_s = Whiten(I_s \circ W(\mathbf{x}_s, \mathbf{x}_{ref}))$
$\mathbf{g}_m = Whiten(\bar{\mathbf{g}} + \mathbf{P}_g\mathbf{b}_g)$
$\delta\mathbf{g} = \mathbf{g}_m - \mathbf{g}_s$

**function** $\mathbf{g}_w = Whiten(\mathbf{g})$
$\mathbf{g}_w = \mathbf{g} - mean(\mathbf{g})$
$\mathbf{g}_w = \mathbf{g}_w / var(\mathbf{g}_w)$
```

Chapter 4

Image-Based Rendering: The Light Field

A large part of computer graphics research focuses on the real-time synthesis of photo-realistic images. Scientists in computer graphics study the physics of nature and the image formation process to design models that can recreate the images we see everyday. Naturally this is an extremely challenging task and has been an on-going topic of study for many years. Data-driven approaches in computer graphics have recently become popular, where images of a scene are collected and used to synthesize the scene from novel vantage points. Methods in this sub-topic of computer graphics and vision, known as *image-based rendering* (IBR), synthesize photo-realistic images of a scene in real-time without the use of complex physical models. Instead, image-based rendering algorithms use clever image sampling algorithms and data structures to perform real-time scene manipulation and re-rendering.

Many image-based rendering algorithms are based off a concept called the *plenoptic function*, introduced by Adelson and Bergen [1]. The plenoptic function models the complete flow of light in space and is parameterized by viewing location, direction, wavelength, and time. We discuss this function in more detail in the following section. If you consider the viewing sphere modelled by this function (see Figure 4-2(b)), and carve out samples on the surface of this sphere, each sample forms an image of the scene. Image-based rendering algorithms implement the plenoptic function by interpolating its samples to synthesize the scene radiance from any viewing direction and location.

Each image-based rendering algorithm is differentiated by the data structures and information used to render the scene. At one end of the spectrum of IBR techniques are the *visual hull* [39] and *view-dependent texture mapping* [12] algorithms. These algorithms use few images in arbitrary positions, with knowledge of scene geometry to render novel views of the scene. At the other extreme of the spectrum is *light field rendering*. This algorithm relies on highly structured and redundant imagery to render the scene without the use of any scene geometry. Other IBR algorithms lie somewhere in-between these two extremes in the amount of imagery versus detailed geometry they use to render the scene. Recently, the *unstructured lumigraph* algorithm [6] was proposed that generalizes many existing IBR methods. This algorithm is used to render the light field deformable model of Chapter 7 and is detailed in Section 4.3.

Although light fields are data intensive they exhibit many attractive properties in the context of deformable models. As discussed above, many of the IBR algorithms render the scene with few images but with detailed geometry of the scene. In practice, the acquisition of detailed, accurate depth proxies is often a difficult and tedious task. Also, because these methods rely on fewer images and scene geometry they cannot as easily model complex lighting, structures and surfaces as the light field rendering algorithm, a purely image-based approach. Provided a set of densely sampled images of a set of objects, one is able to construct a deformable model that easily models complex objects across many different poses. We discuss this algorithm in Chapter 6 and support this claim with experiments in Chapter 7, where we construct a deformable model of the human head using light fields.

We begin this chapter with a brief discussion of the plenoptic function in Section 4.1. We then introduce light fields and discuss light field rending in Section 4.2. Finally, the unstructured lumigraph is presented in Section 4.3.

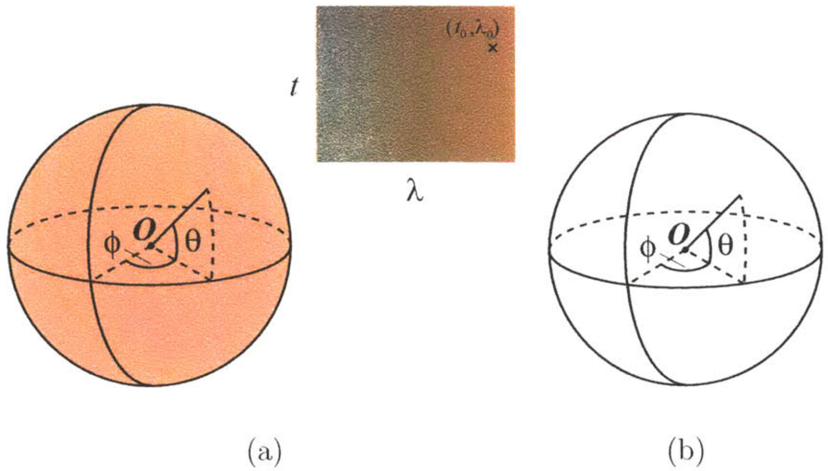


Figure 4-1: Different parameterizations of the plenoptic function: (a) 7D and (b) 5D. [1]

4.1 The Plenoptic Function

The plenoptic function $p(x, y, z, \theta, \phi, \lambda, t)$, introduced by Adelson and Bergen [1], models the complete flow of light in space. It is parameterized by viewing location (x, y, z) and direction (θ, ϕ) along with wavelength λ and time t (see Figure 4-1). A more common parametrization of the plenoptic function is in 5D, where time is held constant and light is assumed to be monochromatic. This treatment of the plenoptic function is used in the development of light fields discussed below.

The 5D plenoptic function $p(x, y, z, \theta, \phi)$ is displayed in Figure 4-2. As illustrated by the figure, the plenoptic function models the world by an infinite set of viewing spheres centered about the different locations in space. Consider a single viewing sphere centered at location $O = (x_0, y_0, z_0)$. Next consider tracing out a rectangular area on the surface of the sphere by considering $p()$ for different values of (θ, ϕ) (see Figure 4-2). It is clear that this structure forms an image with optical center O and field of view $\theta_0 \leq \theta \leq \theta_1$ and $\phi_0 \leq \phi \leq \phi_1$. Images are used to form samples of the plenoptic function. These samples are then interpolated to synthesize views of the scene from different viewing locations and directions. The interpolation method and data structures used to represent the samples of the plenoptic function define the many different image-based rendering algorithms.

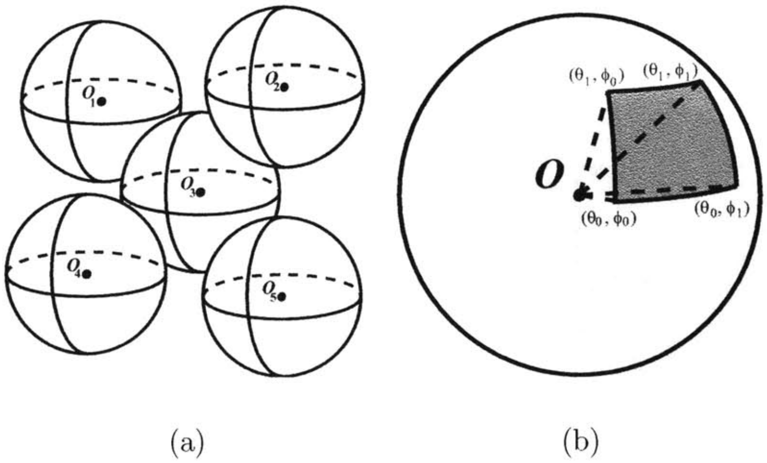


Figure 4-2: The 5D plenoptic function. The plenoptic function models the world by an infinite set of viewing spheres centered about different locations in space (a). Considering a single viewing sphere(b). We can treat the different regions of this sphere as images with optical center O. We synthesize the plenoptic function for different values along this sphere by interpolating these image samples. [17]

Light fields [25, 19] implement a 4D parametrization of the plenoptic function, in which light is assumed independent of viewing location along the light ray. They realize the plenoptic function by treating its samples as sets of ray bundles indexed by pairs of viewing planes and render novel views in real-time by interpolating the sample rays that intersect these planes. A distinguishing characteristic of light fields is that they synthesize novel views of a scene without the use of any scene geometry. A purely image-based approach, light fields can also easily model complex lighting, surfaces, and fine structure. We discuss this algorithm next and demonstrate with examples.

4.2 Light Field Rendering

The *light field* or *lumigraph* is an image based rendering algorithm simultaneously introduced by Levoy and Hanrahan [25] and Gortler et. al [19]. Light fields define a 4D parametrization of the plenoptic function. Consider sampling the plenoptic function from a set of viewpoints looking at a particular scene. Furthermore, assume that these viewpoints lie on some surface M and the objects of the scene are encompassed

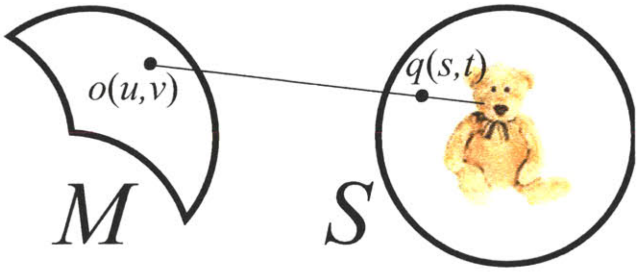


Figure 4-3: 4D parameterization of plenoptic function $L(u, v, s, t)$. [17]

by a spherical surface S. A ray of light leaving the scene intersects the surface S at point $q(s,t)$ and arrives at the surface M at viewpoint $o(u,v)$. The resulting scenario is illustrated by Figure 4-3. Note that any light leaving the scene must intersect surface S. Also, each light ray may only intersect S at a single point. Thus all light rays leaving the scene and arriving at viewpoints M are completely characterized by their unique points of intersection with the surfaces S and M.

This 4D parametrization of the plenoptic function, $L(u, v, s, t)$ is known as a light field [25] or lumigraph [19] and provides a complete characterization of the flow of light emanating from the convex hull of a scene. The light field is realized by constraining S to a planar approximation of the sphere, and M to a plane or set of planes surrounding the scene. This leads to the description of a light field as a set of *light slabs* [25], each light slab consisting of corresponding planar surfaces on S and M.

A light slab is depicted in Figure 4-4(a). In [19], Gortler et. al. describe how to construct a light slab by taking various views of a scene. The light rays from these views that intersect the light slab are kept, and the uv- and st-planes are then discretized via a binning process. By extending a light ray from each discrete location on the uv-plane to that of the st-plane, one finds that the light slab may be described as consisting of a camera and focal plane, the uv-plane consisting of a set of camera centers that each share a common st focal plane. Conversely, using this description the light slab may be thought of as providing the scene radiance for each location on the st-plane as seen by each camera on the uv-plane. This formulation of a light slab, illustrated in Figure 4-4(b), was exploited in [25], in which Levoy and Hanrahan composed light fields by either capturing or synthesizing views of the scene at discrete

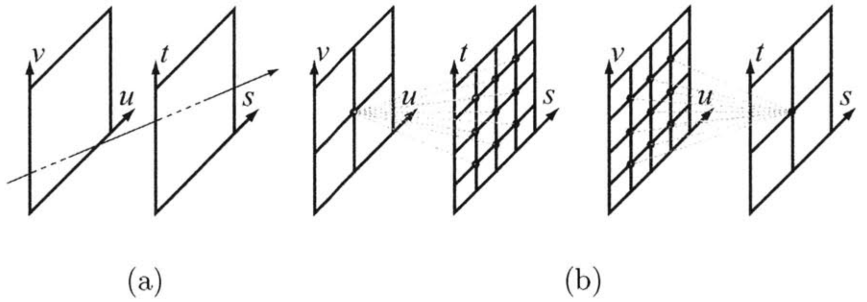


Figure 4-4: (a) Light slab. (b) uv-camera plane and st-focal plane. [25]

points on the uv-plane, each viewpoint sharing a common focal plane st obtained via a skewed perspective projection [25].

A novel view of the scene is computed using light field rendering by taking a slice from one or more of its light slabs as depicted in Figure 4-5. More formally, the color value at each pixel is computed by passing a ray through the center of the virtual camera and the pixel of interest, and forming its intersection with each of the planes uv and st of a light slab. The color of the resulting ray is obtained by interpolating the colors of the rays passing through the discrete values of uv and st near the intersection points. A more efficient method for performing this computation may be obtained via texture mapping [25]. Note by definition light slabs are unique and thus a light ray can only intersect a single light slab. If a light ray does not intersect any light slab, then the color value for that pixel is left blank.

As other image-based rendering algorithms, light field rendering proves advantageous in situations where the scene contains complex structures and/or lighting conditions that are hard to model using traditional 3D rendering techniques. It is different from other image-based rending methods, however, in that it does not require any stereo information [25]. This also comes at a cost, in that light fields usually require many source views to perform well, and if they are to be acquired in real time, using the techniques outlined in [19] and [25] requires the cameras to be constrained to lie on a plane, or set of planes.

A more generalized framework for performing light field rendering was recently outlined by Buehler et. al. [6], known as the *unstructured lumigraph*. The proposed

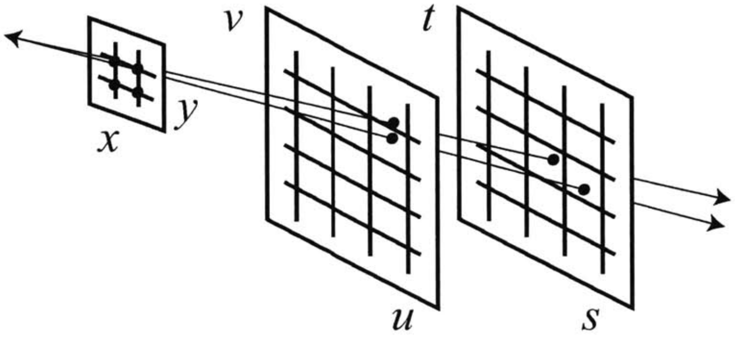


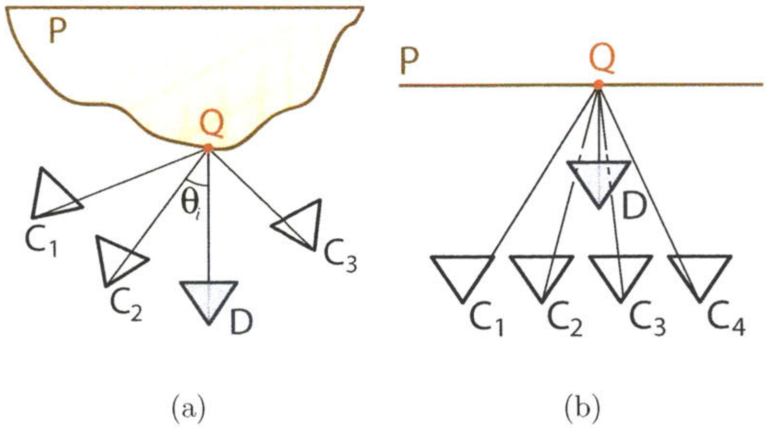
Figure 4-5: Light field rendering. [25]

image-based rendering algorithm functions as view dependent texture mapping in one extreme and as light field rendering in the other. Using a rough geometric proxy of the scene and a set of cameras viewing it from arbitrary locations and orientations, the unstructured lumigraph algorithm computes a virtual view by computing for each of its rays the closest set of camera rays that intersect the same point on the geometric proxy. The color at each pixel of the virtual view is then computed as the weighted sum of the k closest camera rays. This algorithm is discussed in greater detail in the following section.

4.3 The Unstructured Lumigraph

The unstructured lumigraph, introduced by Buelher et. al. [6], generalizes many current image-based rendering algorithms. This algorithm functions on a set of cameras viewing the scene and a geometric proxy that approximates the scene geometry (see Figure 4-6). In one extreme this algorithm is provided a few images of the scene with a detailed geometric proxy and functions as a view dependent texture mapping algorithm (Figure 4-6(a)). In the other extreme the geometric proxy is a plane and many views of the scene are provided situated to all lie in a plane parallel to the geometric proxy (Figure 4-6(b)). In this configuration the algorithm is a light field renderer.

Let C_i, $i = 1, ..., N$ be a set of cameras viewing the scene and P the geometric



(a) (b)

Figure 4-6: The unstructured lumigraph: (a) virtual view D is synthesized using a geometric proxy P and sample views C_i, (b) unstructured lumigraph configured for light field rendering. [6]

proxy (e.g. a triangular mesh of the objects of the scene). This construction is illustrated by Figure 4-6. Consider a virtual view D of the scene. To synthesize the texture of ray, r, contained in D we interpolate the texture of the nearby cameras C_i. This is done by intersecting the ray with the proxy P and then projecting the intersection point Q onto each sample camera, giving texture values,

$$t_i = I_i(M_i Q), \quad i = 1, ..., N, \tag{4.1}$$

where M_i is the projection matrix of each camera C_i and I_i are the sample images of the scene. In Equation (4.1) the texture values t_i are obtained by interpolating the images I_i of each camera at the projected points.

The unstructured lumigraph algorithm synthesizes the texture of a virtual ray by weighting the texture values of the k closest cameras, where proximity is defined by the angle between the ray r and the rays r_i of each sample camera (Figure 4-6(a)). To compute $I_D(r)$ the texture of the k closest cameras are interpolated using weights

$$w_i = \frac{\tilde{w_i}}{\sum_{i=1}^{k} \tilde{w_i}} \tag{4.2}$$

where

$$\tilde{w_i} = \begin{cases} 1 - (\theta_i/\theta_t), & \text{if } r \text{ is inside the field-of-view} \\ 0, & \text{otherwise} \end{cases},$$

Algorithm 7 Unstructured Lumigraph Rendering [6]

```
Let $I_v$ be the view we wish to render, with projection matrix $\mathbf{M}_v$, rotation matrix $\mathbf{R}_v$, translation $\mathbf{t}_v$, and intrinsic matrix $\mathbf{K}_v$. Also, let $N$ be the number of source views, each view with project matrix $\mathbf{M}_i$, focal point $\mathbf{O}_i$ and sample image $I_i$; $k$ the number of source views used to render $I_v$ and $P$ the scene geometric proxy.

Compute $\mathbf{O}_v = -\mathbf{R}_v^{-1}\mathbf{t}_v$.
for all $(x, y) \in I_v$ do
  Project $\mathbf{p} = (x, y, 1)^T$ onto the focal plane $z = z_0$:
  Compute $\mathbf{Q} = BackProject(\mathbf{p}, \mathbf{M}_v, P)$.
  Compute $\mathbf{V} = \mathbf{O}_v - \mathbf{Q}$.
  for $i = 1, ..., M$ do
    Compute $\mathbf{P} = \mathbf{O}_i - \mathbf{Q}$
    Compute $\theta_i = arccos(\mathbf{V} \cdot \mathbf{P})$.
    Compute $\mathbf{q} = \mathbf{M}_i\mathbf{Q}$.
    if $q$ is inside the field of view of view $i$ then
      Compute texture value $t_i = I_i(\mathbf{q})$.
    else
      $t_i = NULL$
    end if
    Sort $\theta_i$ and $t_i$ such that $\theta_i$ is in increasing order and remove $NULL$ texture values.
    Compute threshold angle: $\theta_t = \max_i \theta_i, \; i = 1, ..., k + 1$.
    for $i = 1, ..., k$ do
      $w_i = \theta_i / \theta_t$
    end for
  end for
  Compute total weight $w_t = \sum_i w_i$
  Compute texture value in novel view: $I_v(x, y) = \frac{\sum_i w_i t_i}{w_t}$.
end for
```

and

$$\theta_t = \max_i \theta_i, \quad i = 1, ..., k + 1 \; .$$

In Equation (4.2) the weights w_i are set to zero if the projection of Q onto camera C_i is outside of the camera's field of view (i.e. the coordinates of the projected point fall outside of the image corresponding to that camera). The angle θ_t is a threshold angle that is typically defined as the largest angle of the $k + 1$ closest cameras.

The complete unstructured lumigraph algorithm, that renders the texture of a novel view D from a set of sample views C_i and geometric proxy P is presented as

Algorithm 7. Note the weight computation at each pixel of D searches through all N cameras and then computes the threshold angle by iterating through the $k+1$ closest cameras taking $O(N+k)$ operations, making the total running time of this algorithm $O(M(N+k))$, where M is the number of pixels in the virtual view D. Buelher et. al [6] achieve real-time rendering by computing the weight values at only a few points in the virtual image D and then triangulating all other values.

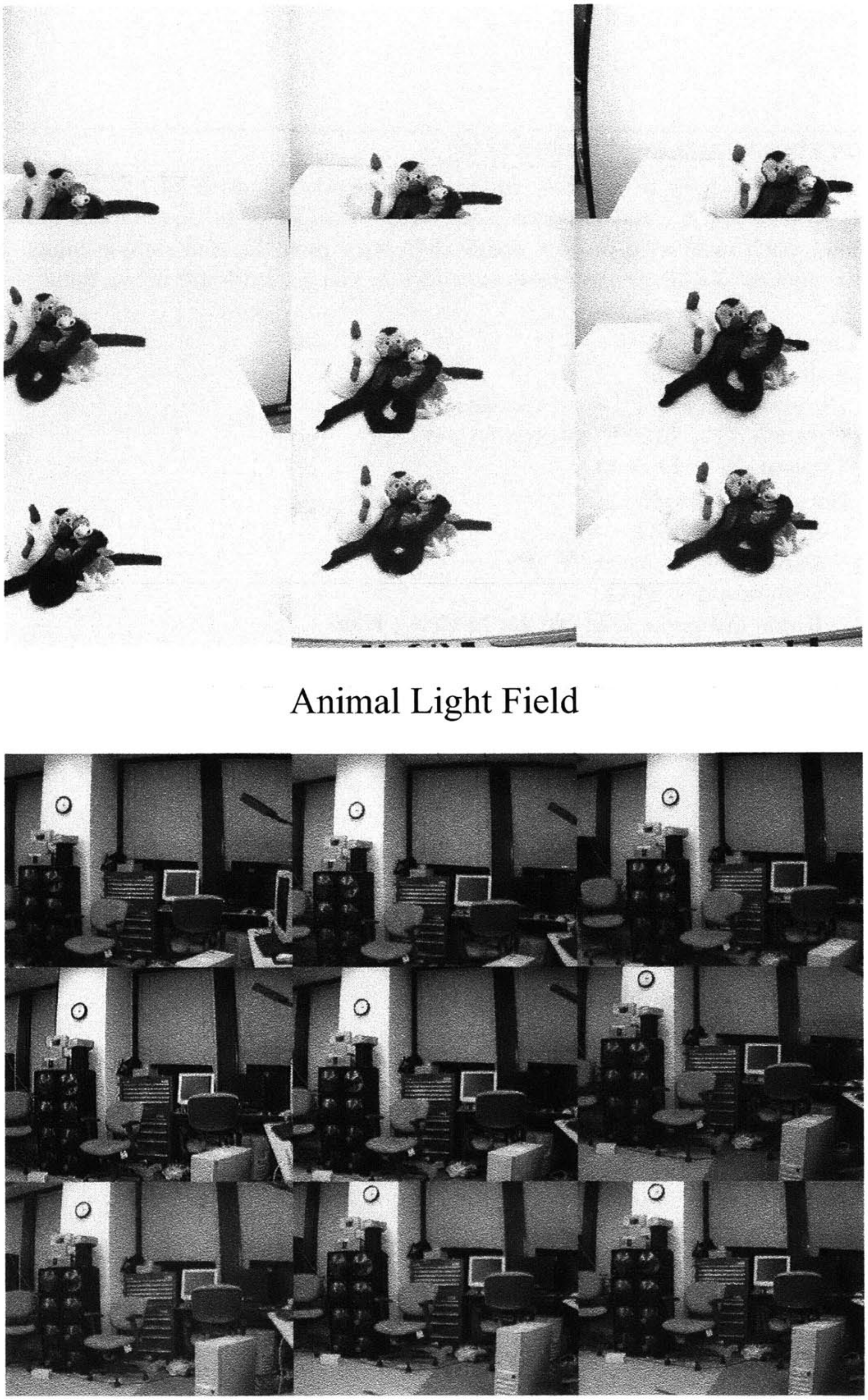
We implemented a version of Algorithm 7 in MATLAB that performs light field rendering. In our implementation P is a plane specified by a depth value z_0. This algorithm is presented as Algorithm 8. Select views of two example light fields are displayed in Figure 4-7 one of an office scene, the other of a group of stuffed animals, captured using a 8×8 camera array [41] (Figure 4-3). Each of these scenes exhibit complex structures (e.g. the fur of the stuffed animals), surfaces and lighting (e.g. the illumination of the water containers). These scenes are rendered from novel views in Figure 4-8 and with different values of z. Note, in our implementation z governs which objects are in focus. The value of this parameter depends on where the objects lie in the scene. In Figure 4-8, z was set using trial and error.

As illustrated by Figure 4-8 light fields are able to represent objects with complex geometry and surface properties across varying 3D pose. In the next chapter we discuss the light field morphing algorithm of Zhang et. al [42]. We extend this algorithm to combine multiple light fields to form a light field deformable in Chapter 6.

Algorithm 8 Light Field Rendering [6]

Let I_v be the view we wish to render, with projection matrix $\mathbf{M}_v$, rotation matrix $\mathbf{R}_v$, translation $\mathbf{t}_v$, and intrinsic matrix $\mathbf{K}_v$. Also, let N be the number of source views, each view with project matrix $\mathbf{M}_i$, focal point $\mathbf{O}_i$ and sample image I_i; k the number of source views used to render I_v and z_0 the depth of the focal plane.

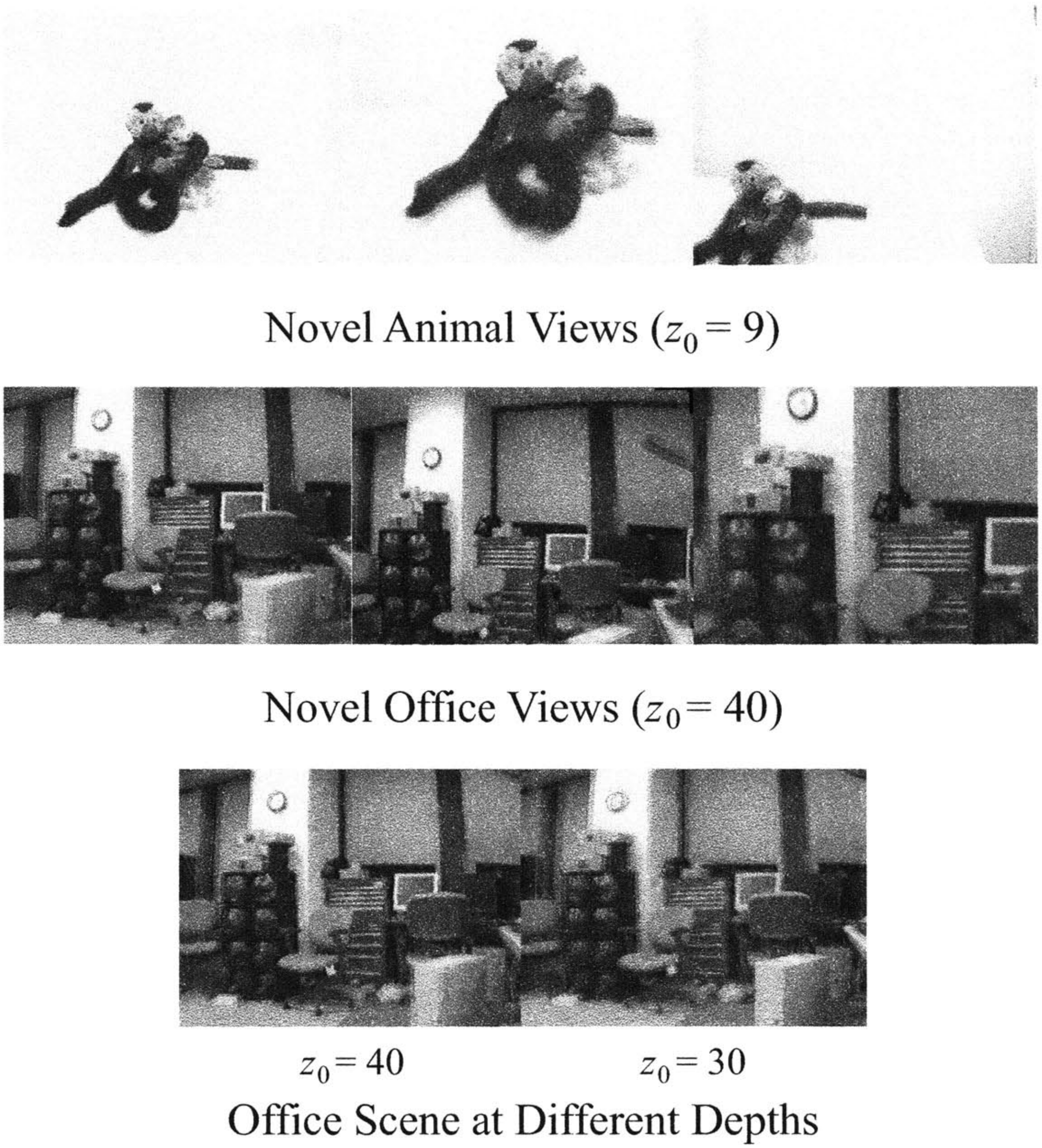
```
Compute $\mathbf{O}_v = -\mathbf{R}_v^{-1}\mathbf{t}_v$.
for all $(x, y) \in I_v$ do
  Project $\mathbf{p} = (x, y, 1)^T$ onto the focal plane $z = z_0$:
  Compute $\mathbf{Q} = BackProject(\mathbf{p}, \mathbf{M}_v, \mathbf{O}_v, z_0)$.
  Compute $\mathbf{V} = \mathbf{O}_v - \mathbf{Q}$.
  for $i = 1, ..., M$ do
    Compute $\mathbf{P} = \mathbf{O}_i - \mathbf{Q}$
    Compute $\theta_i = arccos(\mathbf{V} \cdot \mathbf{P})$.
    Compute $\mathbf{q} = \mathbf{M}_i\mathbf{Q}$.
    if $q$ is inside the field of view of view $i$ then
      Compute texture value $t_i = I_i(\mathbf{q})$.
    else
      $t_i = NULL$
    end if
    Sort $\theta_i$ and $t_i$ such that $\theta_i$ is in increasing order and remove $NULL$ texture values.
    Compute threshold angle: $\theta_t = \max_i \theta_i,\ i = 1, ..., k + 1$.
    for $i = 1, ..., k$ do
      $w_i = \theta_i/\theta_t$
    end for
  end for
  Compute total weight $w_t = \sum_i w_i$
  Compute texture value in novel view: $I_v(x, y) = \frac{\sum_i w_i t_i}{w_t}$.
end for

function $\mathbf{Q} = BackProject(\mathbf{p}, \mathbf{M}_v, \mathbf{O}_v, z_0)$
$\mathbf{V} = \mathbf{M}_v^{-1}\mathbf{p}$.
$\mathbf{P} = \mathbf{V} - \mathbf{O}_v$.
$t = \frac{z_0 - \mathbf{O}_v(3)}{\mathbf{P}(3)}$.
$\mathbf{Q} = \mathbf{P}t + \mathbf{O}_v$.
```



Animal Light Field

Office Light Field

Figure 4-7: Select views of animal and office light fields.



Novel Animal Views ($z_0 = 9$)

Novel Office Views ($z_0 = 40$)

$z_0 = 40$ $z_0 = 30$

Office Scene at Different Depths

Figure 4-8: Novel views of the animal and office light fields (top). Light fields are able to represent objects with complex geometry and surface properties across varying 3D pose. Unstructured lumigraph rendering for different values of z_0 (bottom). Moving the focal plane closer brings the computer into focus whereas moving it farther focuses the clock.

Chapter 5

Light Field Morphing

Object metamorphosis was discussed in the context of images in Chapter 2. In this Chapter we extend these concepts to light fields of objects: we discuss the light field morphing algorithm introduced by Zhang et. al. [42] and show how this algorithm can be extended to function on light fields of multiple objects.

Compelling object metamorphosis can be synthesized using images of objects, as illustrated in Chapter 4. It is difficult, however, to apply these methods to objects that exhibit complex geometry or 3D pose variation. Consider the example objects of Figures 5-1 and 5-2. In Figure 5-1(a) two images of an object under different pose is exhibited. The goal is to synthesize the object under novel poses by interpolating each source image using image morphing. The half-view obtained by morphing each source image with equal weight is displayed in Figure 5-1(b). Note the black regions apparent in the morphed image.

The task of generating novel views of an image using image morphing, known as *view morphing*, was first discussed by Seitz and Dyer [33]. In their paper, they describe the black patches in Figure 5-1(b) as *holes* in the image caused when a part of the object is visible in the morphed image but not in one or both of the source images. A similar phenomena, that they refer to as *folds* in the image, occurs when a part of the object disappears in the morphed image but is present in one or both of the source images. Holes and folds in the morphed image are both caused by *visibility change*: when parts of the object appear or disappear as a result of object

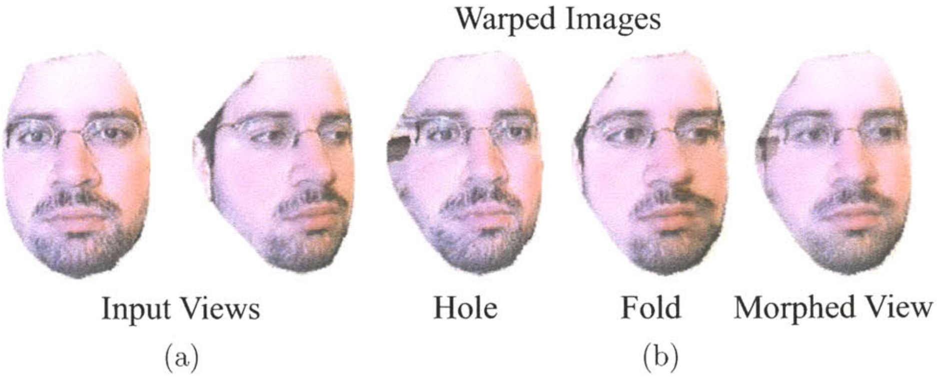


Figure 5-1: Holes and folds: (a) input views, (b) intermediate morphed view. The warped and morphed images are displayed. Note the ear of the side view. The ear folds in the warped side pose and results in a hole in the warped front pose. The hole is prominent in the morphed view.

pose change.

Visibility change may also occur as a result of object shape change. Figure 5-2 shows two images of a cow with different articulation taken from [42]: its legs are under a different configuration in each source image. Note performing metamorphosis between these two images, the cows geometry or shape changes in such a way that parts of its legs appear and disappear as we transform its articulation from the first to the second image. The middle image of Figure 5-2 illustrates such a scenario, where the part of the cow's leg that appears in the morphed image but not in the first source image is highlighted in green. This would result in a hole in the morphed image.

The holes and folds caused by visibility change can be avoided using 3D morphing algorithms [4], however, these methods incorporate a 3D mesh making it difficult to handle objects with complex geometry and surface properties (e.g. a furry animal). Performing object metamorphosis between objects with complex geometry, surface properties, and varying pose is possible in 4D using light fields, that model an object's appearance without the use of scene geometry. A light field morph is computed by forming *ray correspondence* between the morphed light field and each source light field. Holes and folds caused by visibility change are filled using the redundant imagery of the source light fields using a process called *visibility processing*.

In [42], Zhang et. al. present a feature-based light field morphing algorithm that

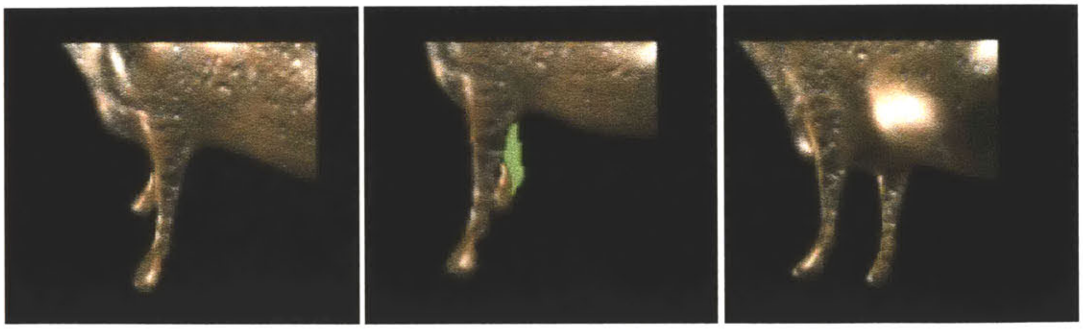


Figure 5-2: Hole caused by object shape change: the cow's legs are under a different configuration in each source view. The potential hole is highlighted in green; this part of the cow's leg appears in the morphed image but not in the first source image. Figure taken from [42].

performs visibility processing to fill arbitrarily large holes present in a morphed view of the light field due to visibility change. This algorithm is detailed in Section 5.1. This algorithm is easily extendible to function on light fields of multiple objects; the extended algorithm is detailed in Section 5.2. When the objects viewed by each source light field are aligned with respect to one another, visibility processing is no longer warranted. The resulting morphing algorithm is also discussed. An interactive light field warping and morphing system was implemented using C++. This system is outlined in Section 5.3 and light field morphing examples obtained using this system are provided in Section 5.4.

5.1 Light Field Morphing Algorithm

Given two light fields L_0 and L_1, we wish to compute a morphed light field L_α, where $0 \leq \alpha \leq 1$, that smoothly transforms L_0 into L_1, each L_α representing a plausible object O_α that preserves the essential features of O_0 and O_1.

Zhang et. al. address the problem of light field morphing by breaking it down into two sub-problems: feature specification and visibility processing. The primary focus of the paper is to deal with arbitrarily large holes generated from visibility change, using the redundant information of the light field. Namely, a part of the object or scene not visible in one view of the source light field, may be visible in another view.

Through feature specification, the user is able to define a rough polygonal model of the object being viewed. In turn, this model is used to partition the light field into *ray bundles*, each ray bundle consisting of a set of rays that intersect a polygon of the model. When performing the morph, visibility processing is applied to fill the holes of L_α using the defined ray bundles.

In their paper, Zhang et. al. outline three basic feature types used to perform feature-based light field morphing. The most important of these is the feature polygon, used for visibility processing. The feature polygon is a 3D planar polygon that approximates a surface patch of a 3D object and by definition contains no self-occlusion. It consists of $\{E^1, ..., E^{n+k}\}$ *control primitives*, each control primitive being a 3D feature line segment, n of which are the edges of the polygon and k supplementary 3D line segments used for additional control within the polygon are also defined. By definition, each view of the light field is calibrated and thus the endpoints of each feature line are obtained by applying stereo. *Background edges* are 2D line segments used to control parts of the object where visibility change does not occur. They are also useful for regions of the object that are hard to approximate using feature polygons, such as the object's silhouette. Background edges are defined in a few key frames and then linearly interpolated into other views.

Once specified, the feature polygons are used to define a *global visibility map*. The global visibility map of a light field L having $\{P_1, ..., P_m\}$ feature polygons is a mapping $V : L \rightarrow N$ from the ray space L to the set of integers N such that,

$$V(u,v,s,t) = \begin{cases} i, & \text{if the ray } L(u,v,s,t) \text{ belongs to } P_i \\ -1, & \text{otherwise} \end{cases} . \tag{5.1}$$

The global visibility map specifies the views of the light field for which a polygon P_i is visible. It also partitions the light field into ray bundles, $R(P_i)$. Namely, using the global visibility map, each ray of the light field is associated a label i corresponding to a feature polygon P_i. The rays not associated to a feature polygon are referred to as *background rays*, which are controlled using background edges. Note that rays may only be associated to a single polygon, since z-buffering is performed in the computation of the visibility map, which means that rays are not associated to occluded feature polygons.

The feature elements and global visibility map are used to geometrically align each light field L_0 and L_1, prior to color blending, to result in L_α, such that L_α does not contain any holes. As outlined in [42], a light field L_0 is warped to L'_0 via the following steps: (a) calculate the feature polygons and background edges of L'_0, (b) build the global visibility map of L'_0, (c) compute the ray bundles of the warped light field L'_0, and (d) treat background rays.

In the first step, the feature polygons and background edges of L'_0 are obtained by linearly interpolating the endpoints of each 2D and 3D line segment v_i^0 and v_i^1 defined for L_0 and L_1 respectively, using (2.12). The global visibility map (5.1) is then computed using the interpolated features. The ray bundles associated with each feature polygon are then warped view-by-view using a technique defined as ray space warping in [42].

Given the ray $L'_0(u, v, s, t)$ in the warped light field, ray space warping defines the corresponding set of rays $\{L_0(u', v', s', t')\}$ in the source light field. These rays may be used to define the color of the warped light field ray. More formally, ray space warping is defined as follows:

Let L be a light field containing m feature polygons. Consider an n-sided feature polygon P_i taken from this set, having feature lines $\{E^1, ..., E^{n+k}\}$ in the source light field, and $\{E'^1, ..., E'^{n+k}\}$ in the warped light field, computed using (2.12). For each ray in the ray bundle $R(P'_i)$, the color of that ray is found as,

$$L'(u, v, s, t) = L(u', v', s', t'), \tag{5.2}$$

where (u', v') are free variables in the uv-plane,

$$(s', t')^T = f\left(s, t, E^1_{(u',v')}, ..., E^{n+k}_{(u',v')}, E'^1_{(u,v)}, ..., E'^{n+k}_{(u,v)}\right), \tag{5.3}$$

$f()$ is the Beier and Neely field warping function (2.11), and $E^j_{(u',v')}$ is the projection of feature line E^j onto view (u', v').

Thus, using (5.2) ray space warping defines a set of possible rays $\{L(u', v', s', t')\}$ from which the color of $L'(u, v, s, t)$ may be assigned. In the case where (u', v') equals

(u, v), ray space warping is equivalent to 2D image warping in each view of the light field.

Ray space warping is applied to each ray of every ray bundle view-by-view. Consider the view (u, v), and the ray bundle associated with polygon P'_i in the warped light field L'_0. The color of each ray $L'_0(u, v, s, t)$ contained by ray bundle $R(P'_i)$ is found by first computing (s', t') using (5.3), with (u', v') equal (u, v). The global visibility map of L_0 is then checked to see whether P_i is visible at ray $L_0(u, v, s', t')$. If so, the color of ray $L'_0(u, v, s, t)$ in the warped light field is assigned to that of $L_0(u, v, s', t')$ in the source light field. If not, it assigned to the ray $L_0(u', v', s', t')$ taken from the set of rays $\{L_0(u', v', s', t')\}$ found using ray space warping (5.2), such that the ray $L_0(u', v', s', t')$ is the "closest ray" to $L_0(u, v, s', t')$ where P_i is visible. The "closest ray" is defined as the (u', v') that minimizes the distance $||(u', v') - (u, v)||$. Note that this ray is guaranteed to exist since the feature polygon was originally specified by the user as visible in a given view of the source light field L_0.

In the last step of light field warping the background rays of L'_0 are treated with 2D image warping using the projected feature lines and interpolated background edges in each view. To complete the morph, L_α is computed by linearly interpolated the colors of the geometrically aligned light fields L'_0 and L'_1 view-by-view using (2.6).

5.1.1 Warping Aligned Objects

When the objects imaged by each source light field are aligned to the same 3D pose and articulation, visibility change does not occur. Consider the dinosaur images of Figure 5-3. Clearly the dinosaurs displayed in the figure have a different geometry and texture. They both have the same pose and articulation, however, and morphing between them will not cause any visibility change. For aligned objects light field morphing reduces to 2D image morphing in each view of the source light fields. More formally, in (5.2) (u', v') may always be assigned to (u, v) and ray space warping is equivalent to 2D image warping. Note, this simplified algorithm, although restrictive, decreases computational cost and is still applicable to the construction of deformable models where input objects can be aligned a priori. This increases the efficiency of

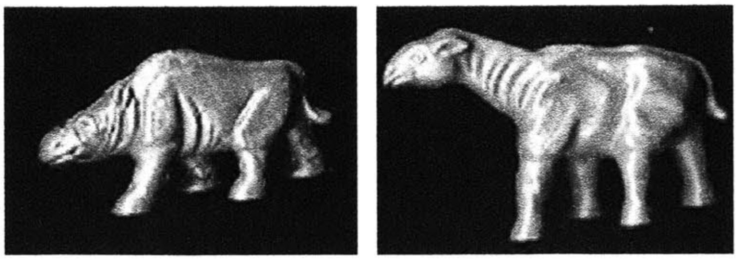


Figure 5-3: Aligned dinosaur images: each dinosaur is under the same pose and articulation. Morphing between them will not generate any visibility change.

the deformable model and is what we employ in the construction of the human head model of Chapter 7.

5.2 Metamorphosis Between Multiple Objects

The light field morphing algorithm of Section 5.1 is easily extendible to function on light fields of multiple objects. For multiple objects, the geometry of the morphed light field is computed by linearly interpolating the 3D point features of each source light field and its texture is computed by blending the texture of the warped light fields. Note, that the ray-space warping algorithm of [42] remains un-altered for multiple objects: the geometry and texture of the morphed object in the case of multiple objects is defined by more than two light fields. The warping algorithm used to align each source light field to the target geometry, however, is the same as that described by Zhang et. al. in [42].

Let L_i, $i = 1, ..., N$ be a set of object light fields, each with 3D feature points v_j^i. The geometry of the morphed light field is defined by,

$$v'_j = \sum_i w_i v_j^i, \tag{5.4}$$

where $\sum_j w_j = 1$, and v_j^i are the vertices of the j^{th} 3D feature line defined with respect to the i^{th} light field. Each source light field, L_i, is aligned to the geometry of the morphed light field using the ray-space warping algorithm of Section 5.1:

$$L'_i = L_i \circ f(v_j^i, v'_j), \tag{5.5}$$

where $f(v_j^i, v_j')$ is the ray-space warping function (5.3). Finally, the morphed light field is computed by blending the texture of the warped light fields L_i' using the interpolation weights w_i:

$$L_m = \sum_i w_i L_i'. \tag{5.6}$$

Relationship (5.4) defines a weighted warp in which the objects of each light field are geometrically aligned to favor the characteristics of the objects having larger weights. The weights are normalized to have sum equal to one such that E_i' is a valid interpolation between the corresponding set of feature lines E_j^i. An alternative interpretation of (5.4) is that the light fields L_j form a basis for the space of light fields that image a particular object class. The projection of a light field from this space onto the basis set L_j is found as the weights w_j. The object viewed in the projected light field may be approximated via the morphing operation defined by the interpolated features (5.4). In Chapter 6 we utilize light field morphing between multiple objects to define a light field deformable model that models the appearance of an object class, exhibiting complex geometry and surface properties, across multiple poses.

5.3 System Overview

A light field morphing system was developed in C++ using QT [35] and MKL [21]. The system was designed to interactively view and morph light fields. The user is able to read a light field that is stored as an array of images. The calibration parameters of each view of the light field is also read from a file. The system is illustrated in Figure 5-4. The system implements the light field morphing algorithm on aligned objects discussed in Section 5.1.1. To compute the morphed light field, the user specifies a set of 3D line features that are projected into each view of the source light fields and Beier and Neely image morphing is applied.

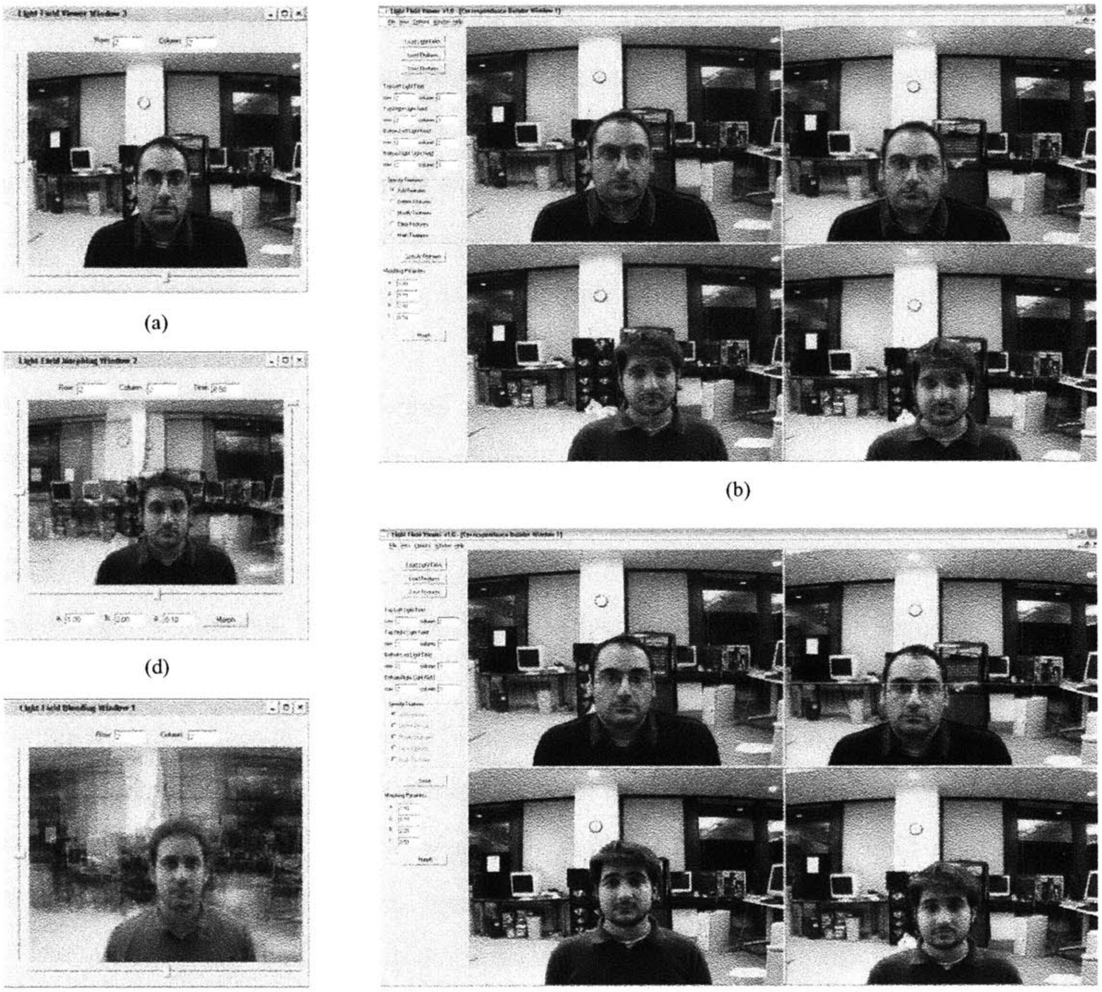
A user may view a light field using a viewer window, displayed in Figure 5-4(a). Using this widget the user is able to traverse the various views of the light field. To morph a set of light fields the user begins by specifying features in a correspondence builder window, shown in Figure 5-4(b), which allows the user to specify a set of 3D feature lines between two light fields. This widget consists of four sub-windows, each top and bottom pair used to display different views of the same light field. The user proceeds to specify features by first clicking the endpoints of the 3D feature line in one view of the light field. The epipolar line for each endpoint is displayed sequentially in the neighboring view to guide the user in specifying the corresponding endpoints as illustrated in Figure 5-4(c). Once completed, stereo is performed to compute the 3D endpoints of the directed feature line, which is then projected into each view for display by the user. The user then repeats this procedure for the corresponding 3D feature line in the other light field. The ability to mark features as not visible in a particular view is also provided.

After feature specification, the user is able morph the set of light fields in a morphing window, Figure 5-4(d). From this window the user loads a light field pair and a set of features saved from the correspondence builder, specifies the morphing parameters, and performs the light field morph. The interpolated and original 3D features are also interactively displayed atop of the morphed and source light fields respectively if so desired by the user.

The last component of the light field morphing system is a blending window, Figure 5-4(e), used to morph between a set of light fields. To do so, the user loads a set of n light fields and n sets of features, one for each light field, and specifies a set of weights as in (5.4). The morph is then performed and displayed to the user. As in the morphing window, the interpolated features may be overlaid if so desired.

5.4 Examples

An array [41], displayed in Figure 5-5, was used to capture a set of aligned face light fields. This was done by placing each subject in front of the light field at fixed location and position. Although the process was not exact, for example there are differences in scaling and translation, and slight differences in rotation between each subject, the main concern is that each subject is aligned such that similar parts of their face are


Light Field Viewer Window 3
Row
Column
Light Field Morphing Window 2
Light Field Blending Window 1
Morph

(a)

(d)

(e)

(b)

(c)

Figure 5-4: Light field morphing system: (a) light field viewer window, (b) correspondence builder window, (c) feature specification guided using epipolar line (in yellow), (d) morphing window with interpolated features overlaid, and (e) blending window with interpolated features overlaid.

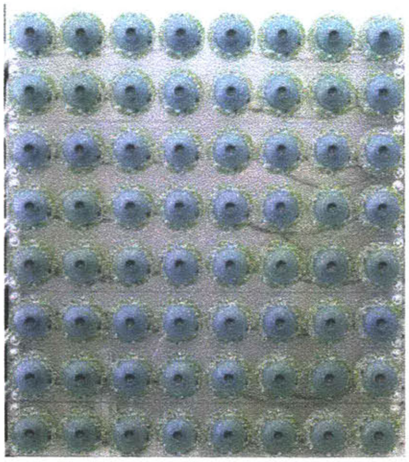
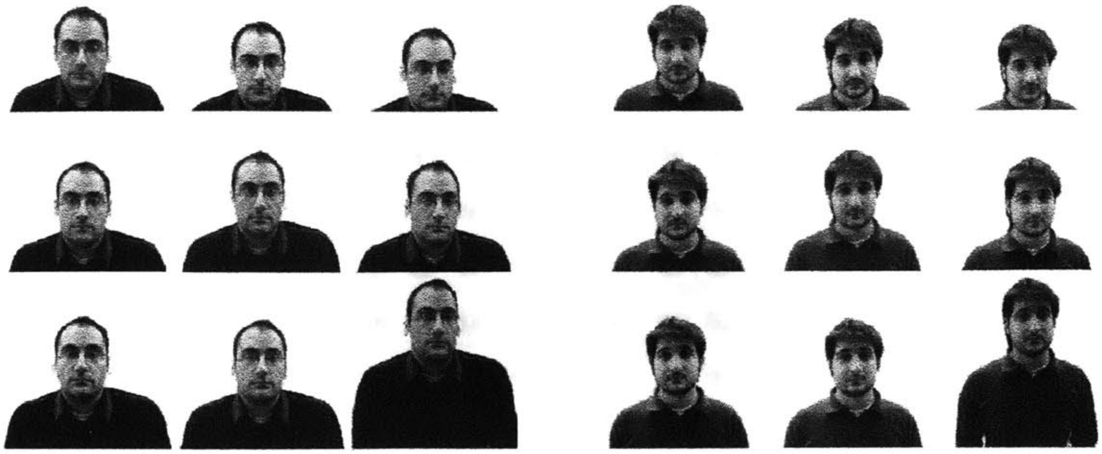


Figure 5-5: Light field camera array [41].

visible in corresponding views, i.e. there is no visibility change between them. The differences in scaling, rotation and translation are accounted for by the geometric alignment step of the morphing operation.

Select views taken from 8 × 8 light fields of two subjects are displayed in Figure 5-6. The morphed light field computed using $a = 1.0$, $b = 2.0$, $p = 0.1$, and $t = 0.5$ is displayed for these views in Figure 5-7. The interpolated features overlaid onto the morphed light field are also displayed in the figure. Although various values of a, b, and p are applicable, these values gave the most aesthetically pleasing results. The features for each light field were specified using the center views of the 3×3 light field displayed in the figure. Unfortunately, due to camera calibration error the features do not project well into the outer views of the 3×3 light field, thus the faces are incorrectly aligned resulting in a poor morph about these views. Nonetheless, focusing on the inner views of Figure 5-7, where correct camera calibration is provided, one finds that the morph performed quite well.

The morphed light field gives the effect of having different views of the same person, this person preserving the important characteristics of each subject. More specifically, the 3D line features defined a consistent deformation field across views resulting in a convincing 3D morph. A morph across time between the two subjects of Figure 5-6 is displayed in Figure 5-8 using a 2×2 light field of each subject. A smooth



First Light Field

Second Light Field

Figure 5-6: 3 × 3 light fields of two subjects.

transition between each person is defined that is uniform across different views of the light field. Independent morphs between the two subjects of Figure 5-6, and three other subjects displayed in Figure 5-9 are displayed in Figure 5-10. As seen from the figure, each morph conveys a realistic 3D morph.

A morph between multiple light fields is provided in Figure 5-11, where the average face is computed between subsets of the subjects from Figure 5-9. The average face between the first three subjects shares each of their characteristics, which are strikingly present in the morphed light field. As the number of subjects used to compute the average face is increased these characteristics become less dominant and the average face more neutral. One is also able to vary the weights to generate a person that has a stronger resemblance to the subject associated with the largest weight, as is done in Figure 5-12. In Chapter 6 we apply light field morphing between multiple objects to define a light field deformable model, that models the appearance of complex objects under varying 3D pose. We demonstrate such a model in Chapter 7, where we construct a 4D deformable model of the human head.

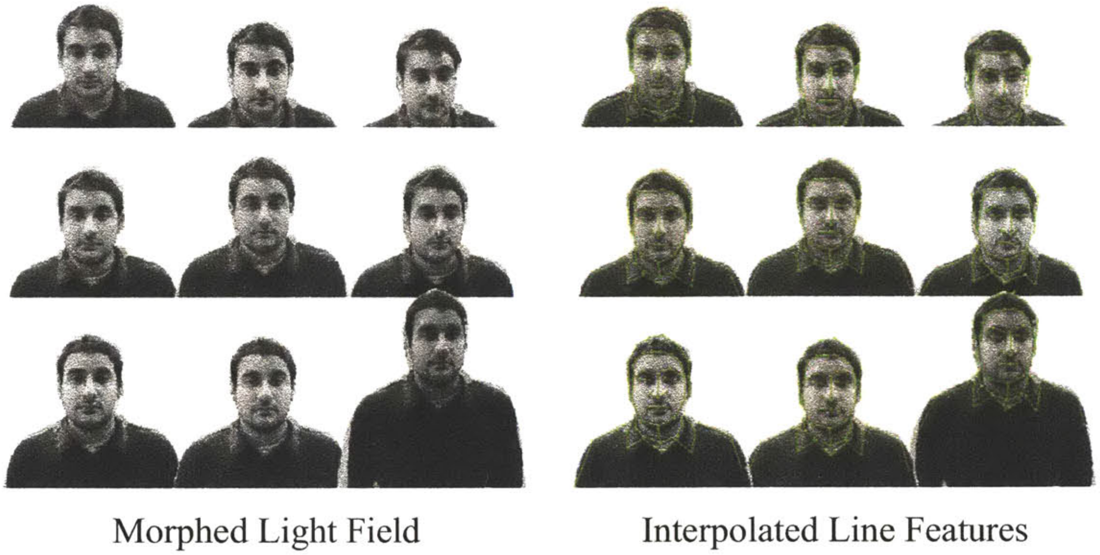
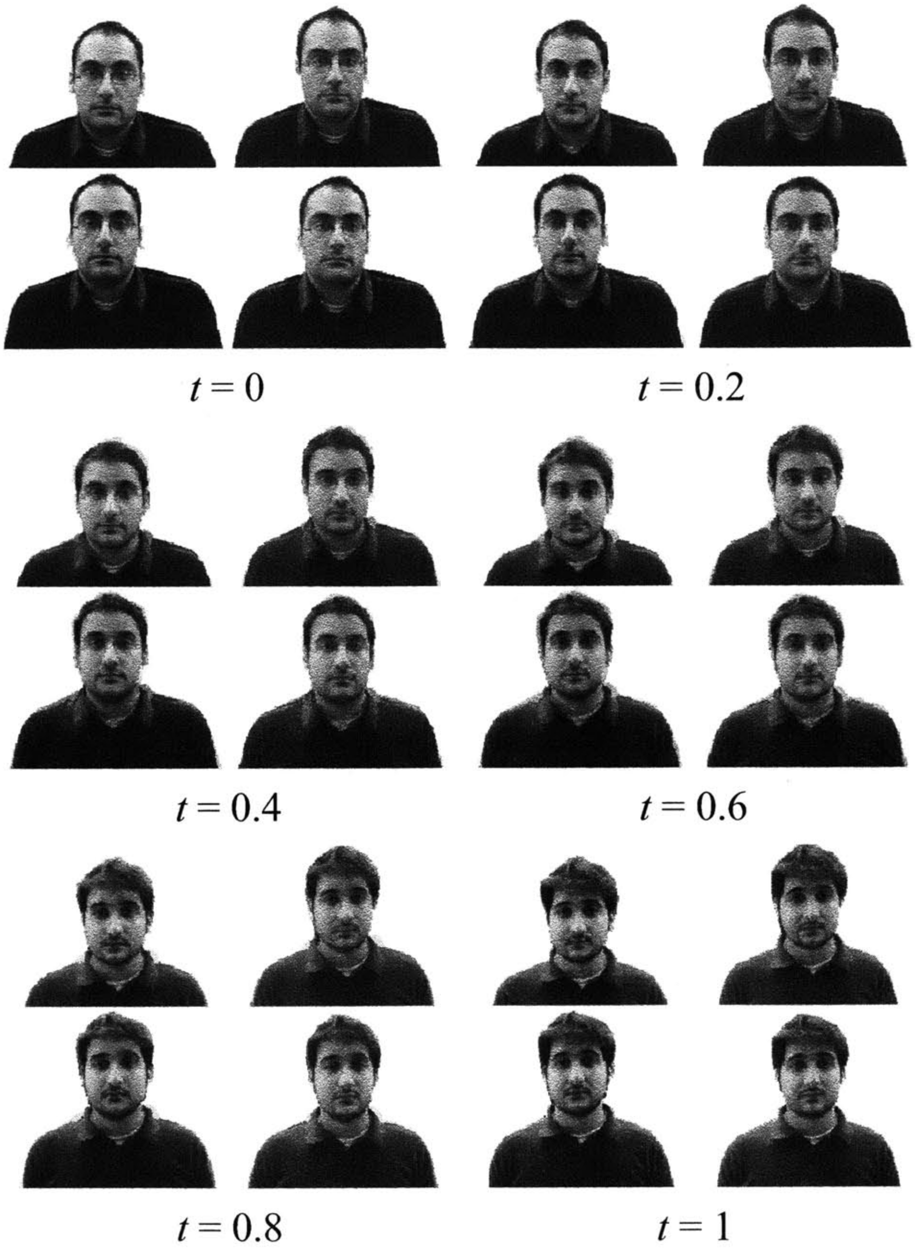


Figure 5-7: Morph between the two subjects of Figure 13 with and without interpolated feature lines overlaid. Parameters used are $a = 1.0$, $b = 2.0$, $p = 0.1$, and $t = 0.5$.



$t = 0$

$t = 0.2$

$t = 0.4$

$t = 0.6$

$t = 0.8$

$t = 1$

Figure 5-8: A morph between subjects 1 and 2 across time, with $a = 1.0$, $b = 2.0$, $p = 0.1$.

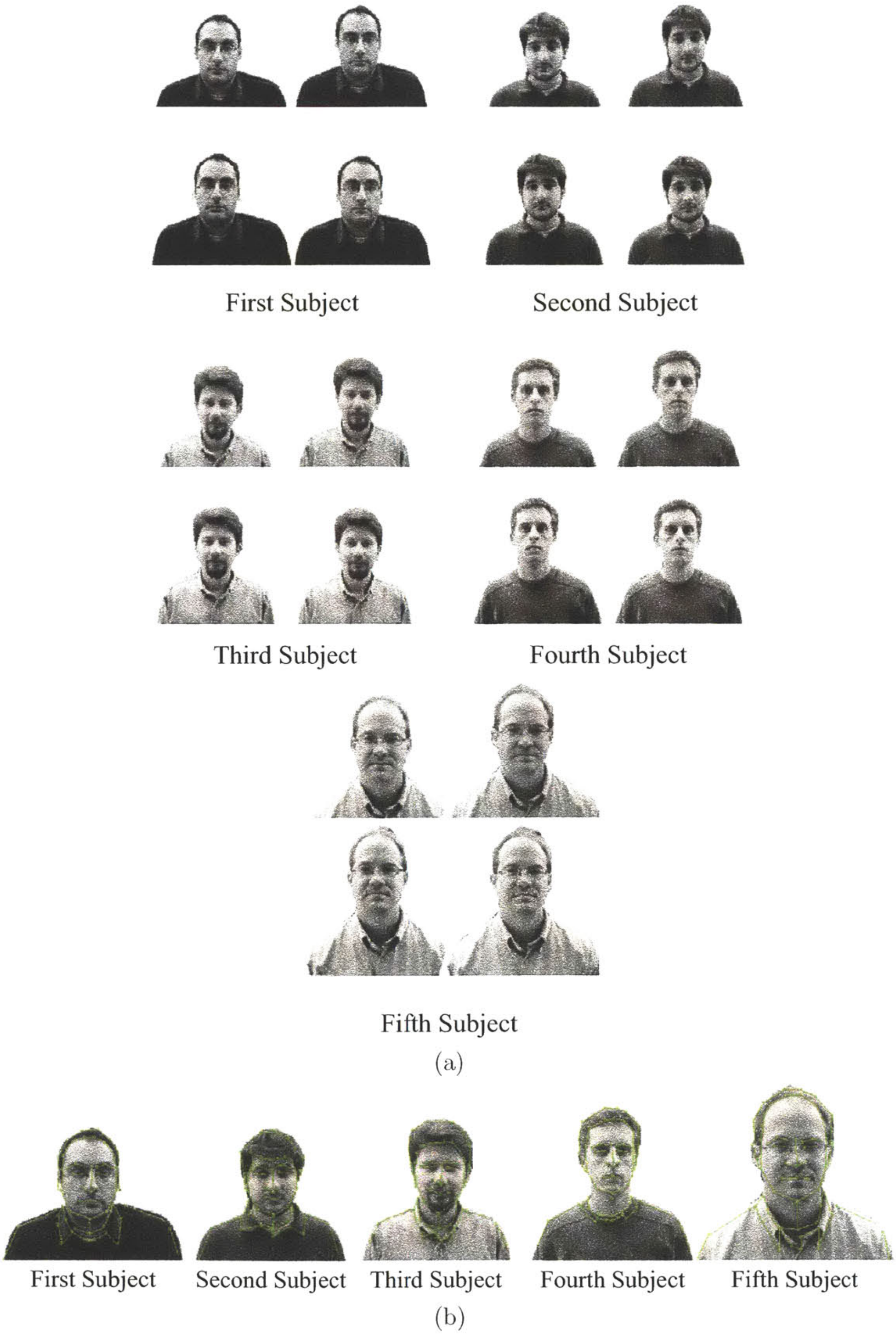
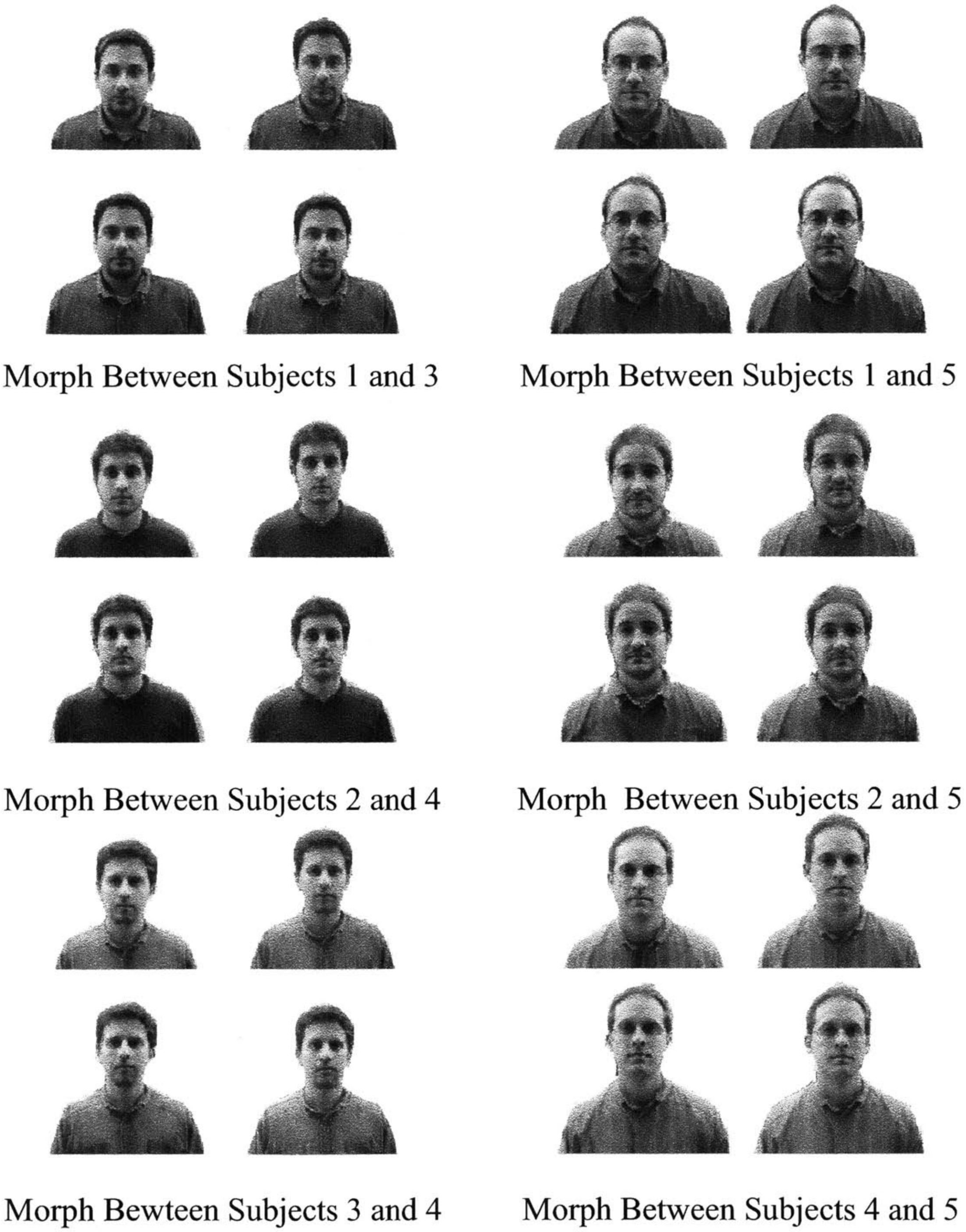


Figure 5-9: 2 x 2 light fields of five different subjects (a) and the projected feature lines seen from a single view (b).



Morph Between Subjects 1 and 3

Morph Between Subjects 1 and 5

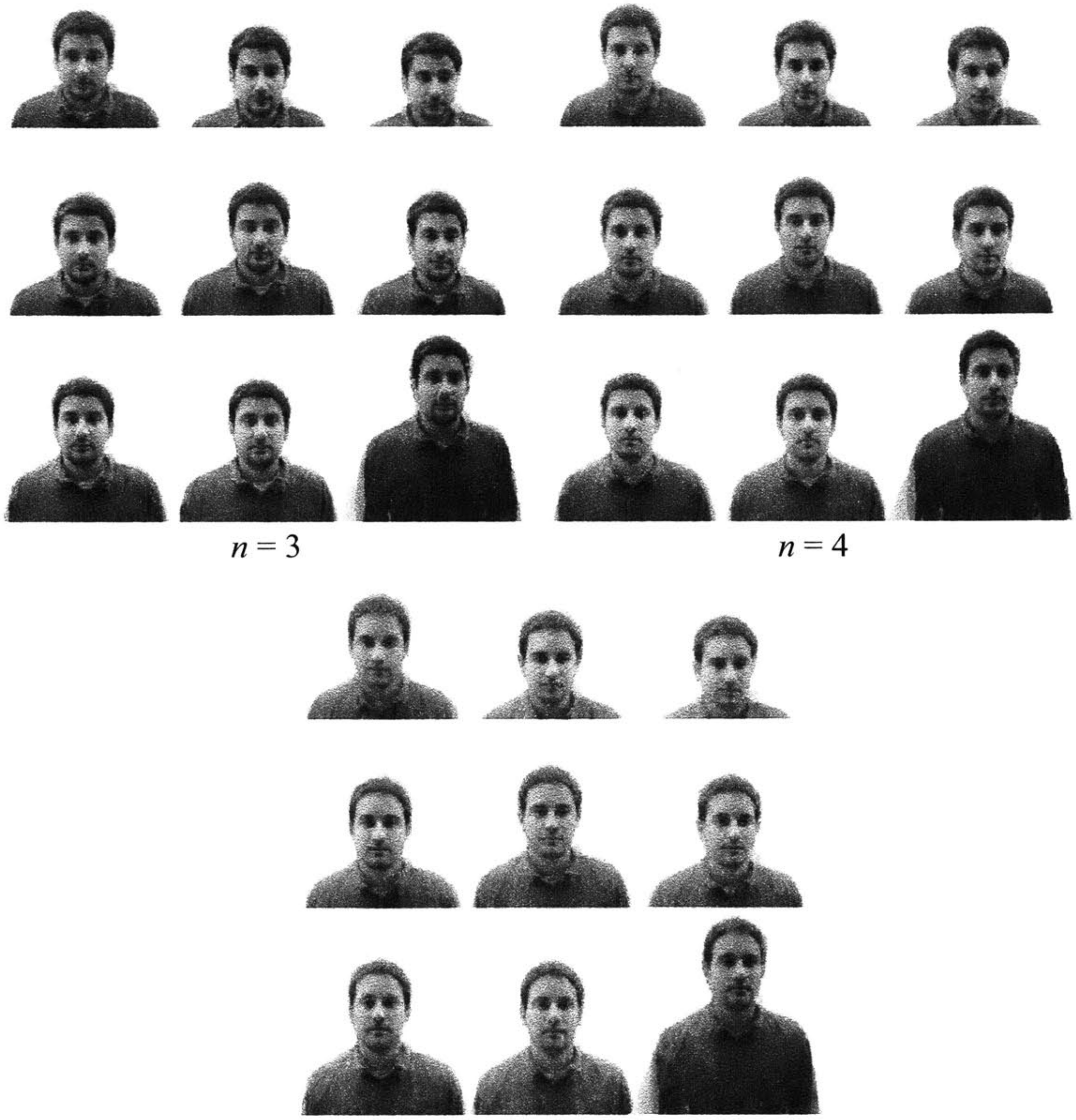
Morph Between Subjects 2 and 4

Morph Between Subjects 2 and 5

Morph Bewteen Subjects 3 and 4

Morph Between Subjects 4 and 5

Figure 5-10: Various light field morphs between the subjects of Figure 5-9, with $a = 1.0$, $b = 2.0$, $p = 0.1$.

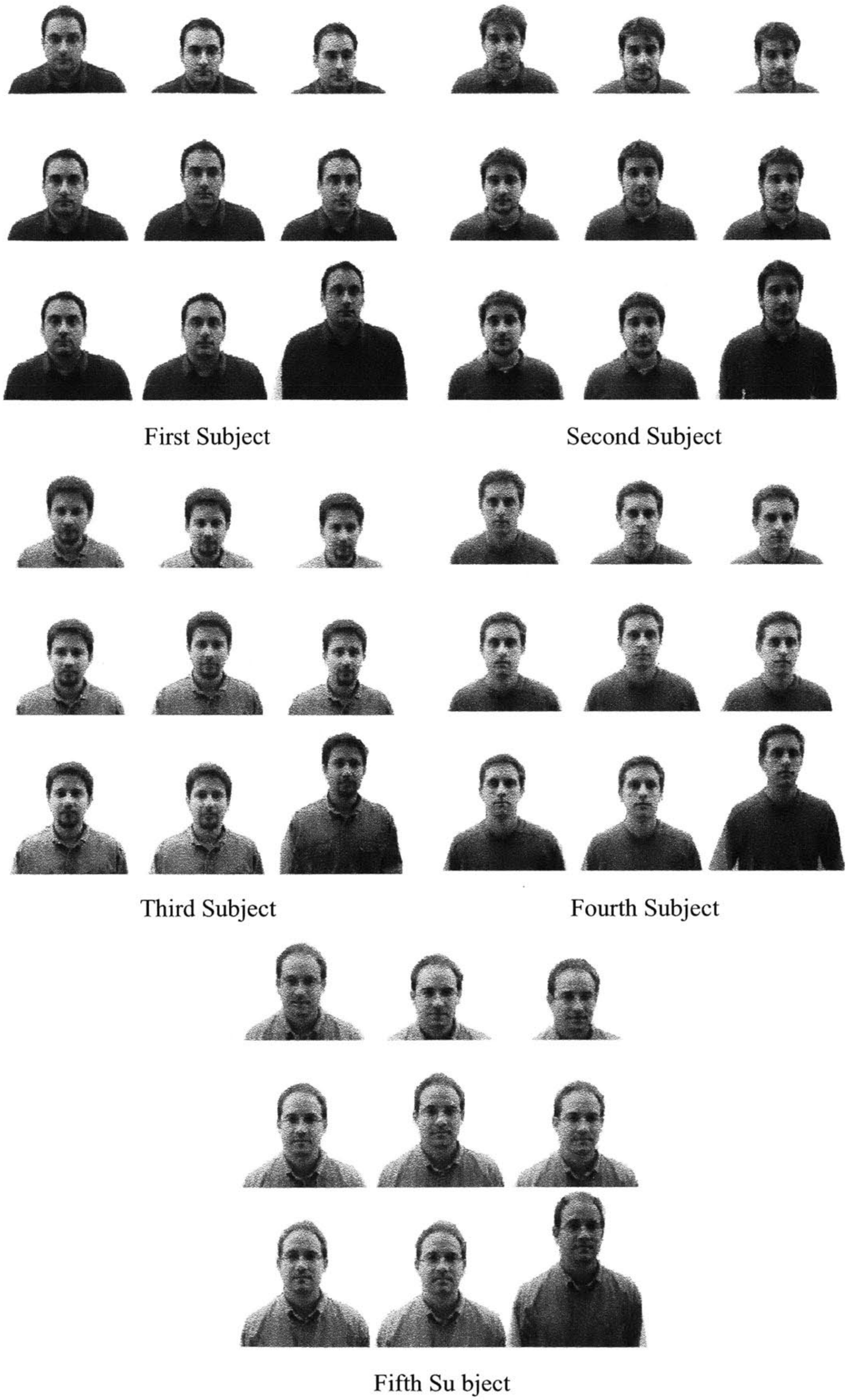


$n = 3$

$n = 4$

$n = 5$

Figure 5-11: Average face ($w_j = 1/n$) between subjects one through three ($n = 3$), subjects one through four ($n = 4$), and all five subjects ($n = 5$). Morphing parameters are $a = 1.0, b = 2.0, p = 0.1$. As the number of light fields used to compute the average face increase, the dominant features contributed by each subject fade and the average face becomes more neutral.



First Subject

Second Subject

Third Subject

Fourth Subject

Fifth Su bject

Figure 5-12: Weighted morph between all five subjects. In each morph, one subject is assigned a weight 0.6 and the others 0.1. In each experiment, the largest weight was assigned to the first subject, the second subject, the third subject, the fourth subject, and the fifth subject. The resulting morphs strongly resemble the subject assigned the largest weight. Morphing parameters are $a = 1.0$, $b = 2.0$, $p = 0.1$.

Chapter 6

4D Shape and Texture Appearance Manifolds

Two-dimensional deformable models represent the appearance of an object class using a bilinear appearance manifold defined over object shape and texture; each point on this manifold maps to an image belonging to the object class. Extending these models to capture 3D pose variation introduces non-linearities in the appearance manifold. As illustrated by Figure 1-2, pose variation is correlated with non-linear differences in appearance - different parts of the object are visible depending on its pose. As a result, the manifold is no longer convex and thus linear combinations of points on the manifold may lead to invalid images. Clearly the linear models of Chapter 3 cannot model full 3D pose variation.

Many extensions to 2D deformable models have been introduced that handle 3D pose variation. Romdhani et. al. [31] use kernel-PCA to model the non-linear shape manifold. With this method, a kernel is applied to the points of the manifold to project them into a high dimensional space where the manifold varies linearly. A generative model of object shape is computed by applying PCA on the high dimensional, linear manifold. Although effective, this method is complex to optimize.

An alternative approach is to model the manifold using a piecewise linear model, as was done by Taylor et. al. [9]. With their approach local-linear models of appearance are manually fit to the different portions of the manifold. A linear regression is then applied to link model parameters across views. When fitting this model, the input image is fit to each local-linear model and the best model fit is retained. The model parameters of the selected model can then be mapped to other local-linear models to synthesize the object under a different pose.

The piecewise linear model of [9] is able to faithfully represent object appearance across 3D pose in 2D. Since these models blend several poses at a given local linear model, however, they have difficulty representing objects that exhibit a non-Lambertian surface reflectance or have a high degree of self-occlusion between poses (e.g. the cow example of Chapter 5). Many local-linear models must be defined to be able to faithfully model the appearance of such object classes, rendering the piecewise linear model in-efficient.

In this chapter we discuss how to build an appearance model that represents object appearance in four-dimensions using light fields. With our model, each point on the bilinear shape and texture appearance manifold maps to a light field of an object (Figure 1-3). Pose is kept as an external parameter to the model and the resulting appearance manifold is well approximated using a linear model. Light fields are purely image-based and do not use any scene geometry to model the appearance of an object. Unlike the view-based 2D models of [9] and the 3D models of [4], our model easily represents object classes with complex surfaces and geometry.

In the following sections we define the concepts of shape and texture in the context of light fields and show how to build a generative model of appearance over these vector spaces. Light field shape can be defined using either 3D or 2D point features. Alternatively, light field shape can be defined using a 4D deformation field that places each ray of the light field in correspondence with the model reference light field, that is automatically computed using optical flow techniques [26]. The texture of each light field is computed by warping each prototype light field to the model reference shape. PCA is then applied to each vector space to build a generative model of appearance.

To match the model, we extend the direct search algorithm of [8] to function over the space of light fields. We also develop an algorithm analogous to Beymer et. al [3] when building the model using optical flow based shape features. Using either algorithm, we show how to match a light field or 2D image of an object to a point on the manifold. When matching to an image, we automatically estimate its pose by searching over the views of the model light field. In turn, the model fit can be used to synthesize the object under unseen views.

In Section 6.1 we provide formal definitions of light field shape and texture in the context of both geometric and optical flow based shape features. We then describe the light field appearance manifold in Section 6.2. We discuss model matching using optical flow in Section 6.3 and present a direct search matching algorithm in Section 6.4. Finally, in Section 6.5 we outline an automatic pose estimation algorithm, used by the matching algorithms of Section 6.3 and Section 6.4 to match the manifold to images having unknown pose.

6.1 Light Field Shape and Texture

In this section we provide a formal description of the shape and texture of a set of light field prototypes that define the appearance manifold of an object class. Let $L(u, v, s, t)$ be a light field consisting of a set of sample views of the scene, parameterized by view indices (u, v) and scene radiance indices (s, t), and let $L_1, ..., L_n$ be a set of prototype light fields with shape $X'_1, ..., X'_n$.

In general, for any image-based rendering technique, X'_i is a set of 3D feature points which outline the shape of the imaged object. With a light field, no 3D shape information is needed to render a novel view of the object. It is therefore sufficient to represent the shape of each light field as the set of 2D feature points, which are the projections of the 3D features into each view. More formally, we define the shape, X, of a light field L as

$$X = \{x_{(u,v)} \big| (u, v) \in L\} \tag{6.1}$$

where $x_{(u,v)}$ is the shape in a view (u, v) of L. If the camera array is strongly calibrated its sufficient to find correspondences in two views and re-project to the remaining views. With only weak calibration and the assumption of a densely sampled array, feature points may be specified in select views of the light field and tracked into all other views. Note the shape representation (6.1) assumes that objects are aligned to have the same approximate 3D pose as discussed in Chapter 5. Since the prototype objects can easily be aligned to the same pose during light field acquisition, this assumption is reasonable.

Once shape is defined for each prototype light field, to increase model efficiency Procrustes analysis [18] is performed to place the shape of each object into a common coordinate frame. Effectively, Procrustes analysis applies a rigid body transformation to the shape of each light field such that each object is aligned to the same exact 3D pose. The normalized shapes X_i are obtained by applying Algorithm 4 of Chapter 3 to the input shapes. Note this algorithm is unchanged for light field shape defined using Equation 6.1. From the set of normalized shapes X_i of each prototype, the reference shape X_{ref} is computed as

$$X_{ref} = \mathbf{M}_{\alpha}\bar{X} \tag{6.2}$$

where $\bar{X}$ is the mean aligned shape and $\mathbf{M}_{\alpha}$ is a matrix which scales and translates the mean shape such that it is expressed in pixel coordinates (i.e. with respect to the height and width of the discrete view of the light field). The matrix $\mathbf{M}_{\alpha}$ constrains the shape in each view of the reference light field to be within the height and width of the view.

As in [3], the texture of a prototype light field is its "shape free" equivalent. It is found by warping each light field to the reference shape X_{ref}. As will be shown in the next section, this allows for the definition of a texture vector space that is decoupled from shape variation. Specifically, the texture of a light field L is defined as

$$G'(u,v,s,t) = L(D(u,v,s,t)) = L \circ D(u,v,s,t) \tag{6.3}$$

where D is the mapping,

$$D : \mathcal{R}^4 \longrightarrow \mathcal{R}^4 \tag{6.4}$$

that specifies for each ray in L_{ref} a corresponding ray in the prototype light field L and is computed using the shape of L and X_{ref}. Note Equation (6.3) implements the light field warping operation discussed in Chapter 5. As in the 2D deformable models of Chapter 3, the texture of each prototype, G_i', is normalized to be under the same global illumination using Algorithm 5. This results in normalized light field texture vectors G_i.

6.1.1 Automatic Shape Acquisition: Optical Flow

Equation (6.3) suggests an alternative, equivalent definition of light field shape given by,

$$X_i = D_i(u, v, s, t), \tag{6.5}$$

where D_i is defined by the mapping (6.4) and specifies for each ray in the reference light field L_{ref} a corresponding ray in the prototype light field L.

Similar to [24], the shape defined by Equation (6.5) can be automatically acquired using optical flow. As with the shape point features of Equation (6.1), shape defined using optical flow also assumes that objects are aligned to have the same approximate 3D pose. This is a reasonable assumption since the prototype light fields can easily be aligned during the acquisition process.

The shape X_i of each prototype light field, defined using Equation (6.5), is computed by applying optical flow between the views of each prototype light field and that of the reference light field. As in the MMMs of Chapter 3 the reference object is chosen to be the average object, since by definition its difference in shape and texture is minimal between each of the light field prototypes and therefore it is the preferred reference light field. Using optical flow, the average light field is computed via the bootstrapping algorithm outlined in [37], presented as Algorithm 3 in Chapter 3. This algorithm placed in the context of light fields is presented below as Algorithm 9. For efficiency we applied the algorithm independently to each view of the prototype set.

Using definition (6.5), light field texture is computed as,

$$G_i(u, v, s, t) = L \circ X_i(u, v, s, t). \tag{6.6}$$

We will use the above definitions of light field shape and texture to define the light field appearance manifold of the following section.

Algorithm 9 Compute Average Light Field

```
Let $L_1, ..., L_n$ be a set of prototype light fields.
Select an arbitrary light field $L_i$ as the reference light field $L_{ref}$
repeat
  for all $L_i$ do
    Compute correspondence fields $X_i$ between $L_{ref}$ and $L_i$ using optical flow.
    Backwards warp each view of $L_i$ onto $L_{ref}$ using $X_i$.
  end for
  Compute the average over all $X_i$ and $G_i$.
  Forward warp each view of $T_{average}$ using $X_{average}$ to create $L_{average}$.
  Convergence test: is $L_{average} - L_{ref} < limit$ ?
  Copy $L_{average}$ to $L_{ref}$
until convergence
```

6.2 Light Field Appearance Manifolds

As illustrated in the previous section, once a reference is defined, each prototype light field may be described in terms of its shape and texture. The linear combination of texture and shape form an appearance manifold: given a set of light fields of the same object class, the linear combination of their texture warped by a linear combination of their shape describes a new object whose shape and texture are spanned by that of the prototype light fields. Compact and efficient linear models of shape and texture variation may be obtained using PCA, as shown in [8], [24]. Given the set of prototype light fields $L_1, ..., L_n$, each having shape X_i and texture G_i, PCA is applied independently to the shape and texture vectors to give

$$\begin{aligned} X &= \bar{X} + \mathbf{P}_s\mathbf{b}_s \\ G &= \bar{G} + \mathbf{P}_g\mathbf{b}_g \end{aligned} \tag{6.7}$$

Using Equation (6.7), the shape and texture of each model light field is described by its corresponding shape and texture parameters $\mathbf{b}_s$ and $\mathbf{b}_g$. As there may exist a correlation between texture and shape, a more compact model of shape and texture variation is obtained by performing PCA on the concatenated shape and texture parameter vectors of each prototype light field. This results in a combined texture-shape PCA space:

$$\begin{aligned} X &= \bar{X} + \mathbf{Q}_s\mathbf{c} \\ G &= \bar{G} + \mathbf{Q}_g\mathbf{c} \end{aligned} \tag{6.8}$$

where as in [8],

$$\begin{aligned} \mathbf{Q}_s &= \mathbf{P}_s\mathbf{W}_s^{-1}\mathbf{P}_{cs} \\ \mathbf{Q}_g &= \mathbf{P}_g\mathbf{P}_{cg} \end{aligned} \tag{6.9}$$

and $\mathbf{W}_s$ is a matrix which comensurates the variation in shape and texture when performing the combined texture-shape PCA. In our experiments we use $\mathbf{W}_s = r\mathbf{I}$ where $r = \sqrt{\sigma_s^2/\sigma_g^2}$. Here σ_s^2 and σ_g^2 represent the total variance of the normalized shape and texture vectors.

Equation (6.8) maps each model light field to a vector $\mathbf{c}$ in the combined texture-shape PCA space. To generalize the model to allow for arbitrary 3D pose and global illumination, Equation (6.8) may be re-defined as follows,

$$\begin{aligned} X_m &= S_t(\bar{X} + \mathbf{Q}_s\mathbf{c}) \\ G_m &= T_u(\bar{G} + \mathbf{Q}_g\mathbf{c}) \end{aligned} \tag{6.10}$$

where S_t is a function that applies a rigid body transformation to the model shape according to a pose parameter vector $\mathbf{t}$, T_u is a function which scales and shifts the model texture using an illumination parameter vector $\mathbf{u}$, and the parameter vectors $\mathbf{t}$ and $\mathbf{u}$ are as defined in Chapter 3. Note, the reference light field has parameters $\mathbf{c} = 0$, $\mathbf{t} = \alpha$ and $\mathbf{u} = 0$, where α is a pose vector that is equivalent to the matrix $\mathbf{M}_\alpha$ in Equation (6.2).

The light field appearance manifold is defined as,

$$L_{model} = G_m \circ D_m \tag{6.11}$$

where L_{model} is a model light field that maps to a point on the appearance manifold and D_m is a 4D deformation field which maps each ray in the model light field to a ray in the reference light field. Using feature-point based shape D_m is computed using the shape of the model light field, X_m and the reference light field, X_{ref}. When X_m is defined using optical flow based shape we set $D_m = X_m$ and we re-define Equation (6.11) as

$$L_{model} = G_m \circ_f X_m \tag{6.12}$$

where f denotes the forward warping operation.

The light field appearance manifold (6.11) is defined over the combined shape and texture PCA space of Equation (6.8). This results in a more compact representation of the manifold as any correlation between shape and texture is captured in the combined space. Note, however, that this manifold can also be defined over the independent shape and texture PCA spaces defined by Equation (6.7).

In the remaining sections we describe two model matching algorithms. We begin by describing an optical flow based model matching algorithm similar to Beymer et. al. [3] that optimizes the model over independent shape and texture spaces. We then present a direct search algorithm in Section 6.4 that optimizes the model over the combined shape-texture space and in Section 6.5 show how the light field appearance manifold can be automatically optimized over images with unknown pose using either matching algorithm.

6.3 Optical Flow Based Model Matching

In this section we present an optical flow based model matching algorithm that is similar to the algorithm of Beymer et. al [3]. With our algorithm, we match a light field or 2D image of an object by first computing the objects shape using optical flow and then match the model by solving the linear system (6.7) of the previous section. We present this algorithm in the context of matching a light field in Section 6.3.1 and then define it for matching an image in Section 6.3.2.

We found our flow-based matching algorithm to be robust and reasonably fast to demonstrate light-field manifold reconstruction. In the next section we present a direct search matching algorithm, similar to [8], that optimizes a model defined using feature-based shape vectors of Equation (6.1) and display results using both matching algorithms in Chapter 7.

6.3.1 Matching to a Light Field

Let X_i and G_i define the shape and the texture of a light field deformable model, specified by prototypes L_i, for $i = 1, ..., n$. We match a light field to a point on the manifold by minimizing the non-linear objective function:

$$E(\mathbf{b}_s, \mathbf{b}_g) = |L_{novel} - L_{model}|^2. \tag{6.13}$$

This objective function synthesizes the model light field, L_{model}, defined by parameters $\mathbf{b}_s$ and $\mathbf{b}_g$ using Equation 6.11 and then compares it to the input light field L_{novel}. We minimize the above error function by computing optical flow between each view of L_{novel} and L_{model} to give shape X_{novel}. We then match L_{novel} to a point on the bilinear shape and texture appearance manifold defined by Equation (6.12) by solving the linear system,

$$\begin{aligned} X_{novel} &= \bar{X} + \mathbf{P}_s\mathbf{b}_s \\ G_{novel} &= \bar{G} + \mathbf{P}_g\mathbf{b}_g \end{aligned} . \tag{6.14}$$

The above system is solved using linear least squares. We display example light field matches using the above algorithm in the next chapter.

6.3.2 Matching to an Image

A 2D image is matched to a point on the light field manifold by minimizing the non-linear objective function:

$$E(\mathbf{b}_s, \mathbf{b}_g, \epsilon) = |I_{novel} - F(L_{model}, \epsilon)|^2, \tag{6.15}$$

where L_{model} is as specified in Equation 6.11 and F is a function that renders pose ϵ of the model light field [25, 6].

The objective function in Equation 6.15 compares the novel 2D image to the corresponding view in L_{model} in a common coordinate frame. Given the weight vectors $\mathbf{b}_s$ and $\mathbf{b}_g$, a model light field is synthesized and the estimated pose ϵ is used to render the view corresponding to that of the novel 2D image.

We match a novel image to a point on the light field appearance manifold defined by Equation 6.12, by first estimating the object's pose using the algorithm outlined in Section 6.5 and then solving a bilinear system in shape and texture constructed using optical flow. More specifically, given a novel 2D image and an estimate of its pose, $\tilde{\epsilon}$, optical flow is computed between the novel image and the average light field rendered at pose $\tilde{\epsilon}$ resulting in x_{novel}, the shape of the novel image. Its texture g_{novel} is found by warping the novel image into the coordinate system of the reference via the deformation field defined by x_{novel}. Using the computed shape and texture we then find $\mathbf{b}_s$ and $\mathbf{b}_g$ by solving

$$\begin{array}{l} x_{novel} = F(\bar{X}, \tilde{\epsilon}) + F(\mathbf{P}_s, \tilde{\epsilon})\mathbf{b}_s \\ g_{novel} = F(\bar{G}, \tilde{\epsilon}) + F(\mathbf{P}_g, \tilde{\epsilon})\mathbf{b}_g \end{array}, \tag{6.16}$$

where F is the light field rendering function [25, 6] applied to the mean light field shape and texture vectors, and to the individual columns of $\mathbf{P}_g$ and $\mathbf{P}_s$ to render them at pose $\tilde{\epsilon}$. Similar to Equation (6.14), we solve the above system using linear least squares.

6.4 Model Matching via Direct Search

In this section, we show how to generalize the matching technique of [8] to light fields. We first illustrate how to match a light field and then discuss the more interesting task of fitting a model light field to a single 2D image.

6.4.1 Matching to a Light Field

A novel light field, L_s, is matched to a point $\tilde{\mathbf{c}}$ on the texture-shape appearance manifold by minimizing the following non-linear objective function:

$$E(\mathbf{p}) = |G_m - G_s|^2 \tag{6.17}$$

where $\mathbf{p}^T = (\mathbf{c}^T|\mathbf{t}^T|\mathbf{u}^T)$ are the parameters of the model, G_m is the model texture and G_s is the normalized texture of L_s assuming it has shape X_m. G_s is computed by warping L_s from X_m to the reference shape X_{ref}. The model shape and texture are computed at $\mathbf{p}$ using Equation (6.10).

The direct search gradient descent algorithm of [8] is easily extendible to a light field deformable model. In [8] a linear relationship for the change in image intensity with respect to the change in model parameters was derived via a first order Taylor expansion of the residual function $\mathbf{r}(\mathbf{p}) = G_m - G_s = \delta\mathbf{g}$. In particular, given a point $\mathbf{p}$ on the manifold, the parameter gradient that minimizes the objective function (6.17) was computed as, $\delta\mathbf{p} = -\mathbf{R}\delta\mathbf{g}$, where the matrix $\mathbf{R}$ is the pseudo-inverse of the Jacobian, $\mathbf{J} = \frac{\partial\mathbf{r}}{\partial\mathbf{p}}$, derived from the Taylor expansion of the residual function.

In a 2D deformable model the columns of the Jacobian are intensity gradient images which model how image intensity changes with respect to each model parameter. Analogously, the Jacobian of a light field deformable model represents the change in light field intensity with respect to the change in model parameters, each of its columns representing light field intensity gradients that describe the intensity change across all the views of a light field. Consequently, the algorithm for minimizing Equation (6.17) follows directly from [8]. As in a 2D AAM, the Jacobian is learned via numerical differentiation.

6.4.2 Matching to an Image

A more interesting extension of the AAM framework arises when performing direct search to match a light field deformable model to a single 2D image; with a light field the Jacobian matrix is rendered based on pose. A novel image I_s is matched to a point on the light field appearance manifold by minimizing the objective,

$$E(\mathbf{p}, \epsilon) = |F(G_m, \epsilon) - g_s|^2 \tag{6.18}$$

where ϵ is the camera pose of I_s, F is a function that renders the pose ϵ of the model texture [25, 6] and g_s is the texture of I_s assuming it has shape x_m. g_s is computed by warping I_s from x_m to the reference shape x_{ref}. Both 2D shapes are obtained by rendering X_m and X_{ref} into view ϵ using,

$$x = F_x(X, \epsilon) \tag{6.19}$$

where F_x is a variant of the light field rendering function F: it renders shape in view ϵ via a linear interpolation of the 2D shape features defined in each view of X.

Overall, the objective function in Equation (6.18) compares the novel 2D image to the corresponding view in L_{model}. Minimizing this objective function fits a model light field, L_{model}, that best approximates I in view ϵ. An efficient way to optimize Equation (6.18) is by defining a two step iteration process, in which the pose ϵ is optimized independently of the model parameters $\mathbf{p}$. We estimate ϵ using the pose estimation algorithm of Section 6.5. The pose parameter $\mathbf{t}$ can be used to further refine this pose estimate during matching.

Once ϵ is approximated, direct search may be employed to match I to a point on the texture-shape appearance manifold. As previously discussed, each column of the Jacobian, $\mathbf{J}$ of a light field deformable model is a light field intensity gradient. To approximate the intensity gradient in view ϵ of the target image I, light field rendering is applied to each column of $\mathbf{J}$. This yields a "rendered" Jacobian matrix, $\mathbf{J}_\epsilon$, specified as,

$$\mathbf{J}_\epsilon^i = F(\mathbf{J}^i, \epsilon), \quad i = 1, ..., m \tag{6.20}$$

where $\mathbf{J}^i$ represents column i of the matrix $\mathbf{J}$ and m is the number of columns in $\mathbf{J}$. Note similar to the model and image textures of Equation (6.17) the columns of $\mathbf{J}_\epsilon$ have shape x_{ref} defined above.

Using $\mathbf{J}_\epsilon$, we optimize Equation (6.18) using a modified version of the direct search algorithm of Cootes et. al. [8]. The modified algorithm is presented as Algorithm 10. Comparing this algorithm with Algorithm 6 of Chapter 3, an important difference is in the application of the pose parameter vector $\mathbf{t}$. Contrary to what is suggested by Equation (6.10), the global affine warp S_t is applied to the rendered model image and not to the model light field (step 3 of the *Residual* function in Algorithm 10). This is because rotating, scaling, and/or translating the images of L_{model} according to S_t may violate the light field construction when matching to an image. To see this, consider manipulating a single-slab light field. Applying S_t to this light field effectively rotates or displaces the focal plane (st-plane) of the light slab (note, scaling the images correlates to widening the gap between the camera and focal planes of the light slab). Clearly, moving the focal plane of the light field will alter where the scene rays will intersect it. If the imaged object is planar then the scene rays will follow

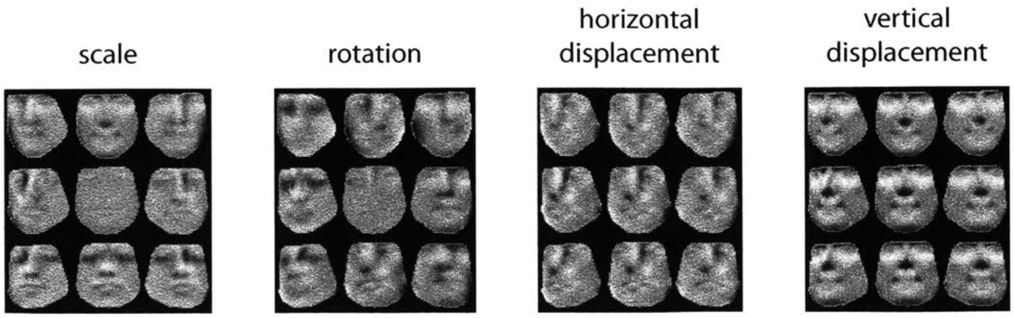


Figure 6-1: Select views of the four columns of the model Jacobian that correspond to scale, rotation, and horizontal and vertical translation of the light field focal plane. Such displacements, although correct for a light field input, are not appropriate when matching an image. Note the large displacements in the extreme views of the light field intensity gradients corresponding to scale and rotation.

the motion of the focal plane. For non-planar objects this is not necessarily the case, however.

By applying the affine warp on the rendered model image the model light field remains in the coordinate frame of the reference light field, while still affording the model affine flexibility in the coordinate frame of the input image. Another benefit of the above matching algorithm is that it avoids the need to optimize over z, the depth of the focal plane of the unstructured lumigraph, during matching. The scaling performed by S_t when applied to the model light field effectively changes this value and thus z would need to be optimized over as well when performing the match. By keeping the model light field in the coordinate frame of the reference light field, this need is eliminated and we let $z = z_0$, the depth of the average light field.

Note, when fitting the model to an object light field, we can safely apply S_t to the images of the model light field as is done in the optimization algorithm of Section 6.4.1. This is because the set of allowable affine transformations is constrained by the 3D pose of the input light field. Matching an image is more ambiguous, and can result in transformations S_t that when applied to the images of the light field violate its construction.

Algorithm 10 Direct Search Algorithm for Matching an Image

Let I_s be an input image with estimated pose ϵ, X_m, G_m the model shape and texture vectors of the light field appearance manifold, and $\mathbf{J}$ the model Jacobian.

```
Compute $\mathbf{J}_\epsilon$ using Equation (6.20).
Set $\mathbf{p} = \mathbf{p}_0$
Evaluate $\delta\mathbf{g} = Residual(I_s, \mathbf{p}, \epsilon)$
repeat
  Compute error $E_0 = |\delta\mathbf{g}|^2$
  Evaluate $\delta\mathbf{p} = -\mathbf{R}\delta\mathbf{g}$
  Update parameters, $\mathbf{p}_1 = \mathbf{p} + \delta\mathbf{p}$
  Evaluate $\delta\mathbf{g} = Residual(I_s, \mathbf{p}_1, \epsilon)$
  Compute error at new $\mathbf{p}$ value: $E = |\delta\mathbf{g}|^2$
  if $|E - E_0| \geq 0$ then
    Set $i = 0$, $k = 1.5$
    while $|E - E_0| \geq 0$, $i \leq n$ do
      Set $\mathbf{p}_1 = \mathbf{p} + k\delta\mathbf{p}$
      Set $i = i + 1$
      Evaluate $\delta\mathbf{g} = Residual(I_s, \mathbf{p}_1, \epsilon)$
      Compute error $E = |\delta\mathbf{g}|^2$
      if $k > 1$ then
        Set $k = 0.5$
      else
        Set $k = k/2$
      end if
    end while
  end if
  if $|E - E_0| < 0$ then
    Set $\mathbf{p} = \mathbf{p}_1$
  end if
until $|E - E_0| \geq 0$

function $\delta\mathbf{g} = Residual(I_s, \mathbf{p}, \epsilon)$
$x_{ref} = F_x(X_{ref}, \epsilon)$
$x_s = F_x(X_m(\mathbf{b}_s, t_0), \epsilon)$
$x_s = S_t(x_s)$
$\mathbf{g}_s = Whiten(I_s \circ W(x_s, x_{ref}))$
$\mathbf{g}_m = Whiten(F(G_m(\mathbf{b}_g), \epsilon))$
$\delta\mathbf{g} = \mathbf{g}_m - \mathbf{g}_s$

function $\mathbf{g}_w = Whiten(\mathbf{g})$
$\mathbf{g}_w = \mathbf{g} - mean(\mathbf{g})$
$\mathbf{g}_w = \mathbf{g}_w / var(\mathbf{g}_w)$
```

6.4.3 Jacobian Computation for Matching an Image

As discussed in the previous section, when matching to an image the pose parameter vector $\mathbf{t}$ affects the 2D pose of the rendered model image not the model light field. Naturally, this alters the Jacobian used to fit the model to images. Figure 6-1 displays the four columns of a Jacobian, computed for the face data of the next chapter, that correspond to the scaling, rotation, and horizontal and vertical translation parameters represented by $\mathbf{t}$. As discussed in Section 6.4 and displayed in the figure, each column of the Jacobian is a light field representing the change in scene intensity with respect to a given model parameter.

The pose columns of the Jacobian of Figure 6-1 are computed by globally applying S_t to the images of the model light field. Each column illustrates a global rotation, scaling or shifting about the object's center. Note the large displacements in the corner images of the columns corresponding to rotation and scaling. Although, this Jacobian is correct in the context of the light field matching algorithm of Section 6.4.1, it does not apply when fitting the model to images: the affine warping defined by the pose parameter $\mathbf{t}$ is applied locally, centered about the pose of the input image in Algorithm 10.

We desire the rendered Jacobian $\mathbf{J}_\epsilon$ to encode an affine warping local to the pose of the input image. To accomplish this we rotate, scale, and translate each view of the model light field independently in the computation of the model Jacobian. This results in the model Jacobian displayed in Figure 6-2, also computed using the face model data of Chapter 7. Note that the columns of this Jacobian display an affine warping local to each image of the model light field. Figure 6-3 displays the corresponding four columns of the Jacobian matrix $\mathbf{J}_\epsilon$, rendered at an arbitrary pose of the model light field. As desired, the difference images of the rendered Jacobian specify an affine transform that is centered about the input pose ϵ.

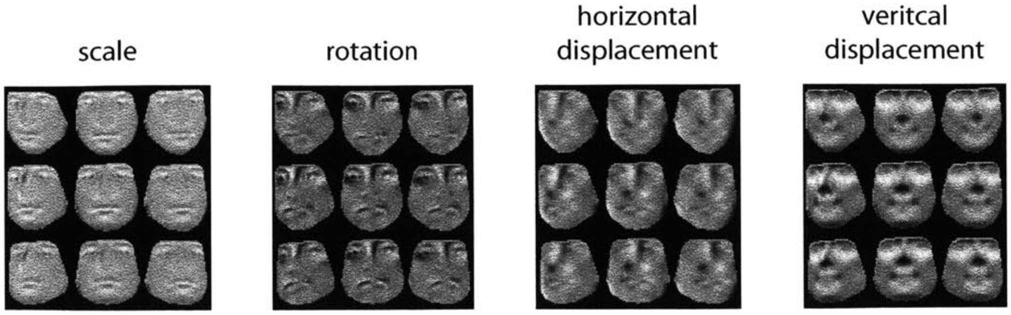


Figure 6-2: Model Jacobian used for optimizing the model over a 2D image. Select views of its four columns corresponding to scale, rotation, and horizontal and vertical translation are displayed.

6.5 Automatic Pose Estimation Algorithm

In this section we present an automatic pose estimation algorithm that estimates the pose of an input image by performing stochastic gradient decent over the views of the model light field. This algorithm is used by the optical flow based and feature-based matching algorithms presented above.

Our pose estimation algorithm is summarized as Algorithm 11. Provided an input image I_s, we obtain an initial estimate of the object's pose, ϵ_0, by performing cross-correlation between the image and each view of the average light field. We then match the image to this view and each of its eight-connected neighbors. We move to the neighbor with smallest fit error and iterate until the central view has the smallest fitting error of its neighbors. To avoid local minima we randomly perturb the fit upon convergence. Final convergence is declared when the algorithm converges to the same discrete pose twice.

Note, when matching to discrete views of the model light field, light field rendering is not required. Instead, the model shape and texture vectors, as well as the Jacobian of Section 6.4, are directly sampled at the current discrete pose. Since the pose estimation algorithm matches to potentially many views of the light field, this leads to a great increase in efficiency.

Once convergence is declared at a discrete pose of the model light field, we estimate the object's pose, ϵ, by fitting a quadratic to the fit error of the eight-connected

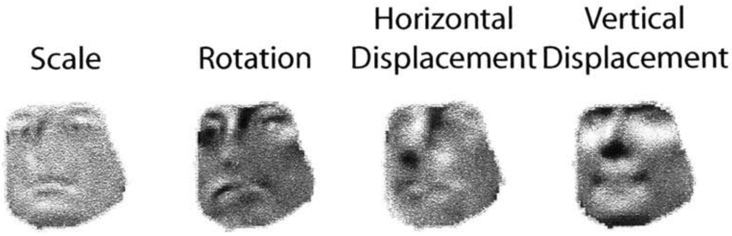


Figure 6-3: Model Jacobian rendered at an arbitrary pose. The rendered Jacobian portrays the desired affine transform centered about the rendered pose.

neighborhood centered about the computed discrete pose. The pose, ϵ, is set to the minimum of the fit quadratic. Note higher-order polynomials could have been used to interpolate the pose. We use a quadratic for simplicity and justify it using empirical evidence in the following chapter.

Algorithm 11 Pose Estimation Algorithm

```
$I_s$ is an input image for which we wish to estimate its pose. $C(\epsilon)$ is a lookup table that stores for each discrete pose the number of times the algorithm converged to that pose. $Fit()$ is a function that fits $I_s$ to the manifold $L_{model}$ at pose $\epsilon$ and returns the fitting error.

Initialize $C = 0$

*Compute initial pose estimate* $\epsilon_0$:
**for** all $\epsilon_i \in L_{ref}$ **do**
  Compute normalized cross-correlation $E(\epsilon_i) = N(L_{ref}(\epsilon_i), I_s)$
**end for**
Set $\epsilon_0 = \underset{\epsilon_i}{\text{argmin}}\ E(\epsilon_i)$

*Perform stochastic gradient descent starting from* $\epsilon_0$:
Set $\epsilon = \epsilon_0$
**repeat**
  **repeat**
    Fit model at pose $\epsilon_0 = \epsilon$: $E(\epsilon_0) = Fit(L_{model}, I_s, \epsilon)$
    **for** $i = 1, ..., 8$ **do**
      Set $\epsilon_i = \epsilon_0 + neighbor(i)$
      Fit model at neighbor $i$: $E(\epsilon_i) = Fit(L_{model}, I_s, \epsilon_i)$
    **end for**
    Set $\epsilon = \underset{\epsilon_i}{\text{argmin}}\ E(\epsilon_i)$
  **until** $\epsilon = \epsilon_0$
  Update lookup table: $C(\epsilon_0) = C(\epsilon_0) + 1$
  Randomly perturb pose from $\epsilon$ to give $\epsilon = \epsilon_p$.
**until** $C(\epsilon_0) = 2$

*Interpolate pose about* $\epsilon_0$:
Fit quadratic to error of 8-connected neighborhood: $Q = FitQuadratic(E)$
Compute final pose estimate $\epsilon = ComputeMin(Q)$
```

Chapter 7

Experiments and Results

In the previous chapter we presented a deformable model defined over a 4D appearance manifold using light fields. We placed this model in the context of both feature-point and optical flow based shape vectors and explained how to match such models to a light field or 2D image of an object with unknown pose. Unlike the 2D deformable models of Chapter 3, light field deformable models can easily handle object pose variation. In contrast to existing 3D and view-based approaches, they can also model objects exhibiting a complex surface reflectance and/or geometry.

In this chapter we support these claims with experiments. We begin by outlining our experimental setup in Section 7.1. In this section we discuss the capture apparatus we used to collect our data, along with the specifics of our data set and the parameters of our models. In Section 7.2 we compare our approach to the view-based AAM [9] and demonstrate how our model is able to capture the view-dependent texturing of a subject's glasses. We then demonstrate fitting both a feature-based and optical flow based head model to light fields and 2D images of subjects outside of the model database in Sections 7.3 and 7.4. In these sections we show how we can match our model to 2D images of an object with unknown pose and extract a full 4D representation of the object containing unseen views of the object. We also justify the use of a quadratic fit in our pose estimation algorithm using empirical evidence. Finally, in Section 7.5 we outline our implementation and provide timing results for our algorithms.

7.1 Experimental Setup

We built a light field deformable model of the human head by capturing light fields of 50 subjects using a real-time light field camera array [41]. We collected 48 views (6 x 8) of each individual and manually segmented the head from each light field. Our head database consists of 37 males and 13 females of various races (see Figure 7-1). Of these people, 7 are bearded and 17 are wearing glasses. To minimize the effects of color calibration error we built our models in grayscale. An example light field is displayed in Figure 7-2. The images in each view of the prototype light fields have resolution 320 x 240. Within each image, the head spans a region of approximately 80 x 120 pixels. The field of view captured by the camera array is approximately 25 degrees horizontally and 20 degrees vertically. To perform feature tracking, as described in Chapter 6, we used a multi-resolution Lukas-Kanade optical flow algorithm [26], with 4 pyramid levels and Laplacian smoothing [1]. We also use this algorithm to construct the shape vectors of the optical flow based model.

When matching our model to an image we assume that object location is approximately known. In the case of a head model, such information can be readily obtained from a face detector [23]. In our experiments we manually specify the object location in each input image. When fitting the model with optical flow we found the above flow algorithm to be sensitive to scene clutter. Thus in this situation we also provide an image mask that segments the region of interest. We discuss this assumption in the following chapter. Note, no such mask is provided when optimizing the feature-based model of Chapter 6 as the direct search algorithm does not share the same sensitivity to scene clutter.

To perform light field rendering we use the unstructured lumigraph algorithm described in [6]. As mentioned in Chapter 4 this algorithm has two parameters: k for the number of source views used to render a scene and z_0 the approximate depth of the focal plane of the light field. In our experiments we used a value of $k = 3$ when optimizing the feature based model and $k = 1$ for the optical flow based model. As

[1] We acknowledge Tony Ezzat for the Lukas-Kanade optical flow implementation.

Variables	Perturbations
x, y	±5% and ±10% of the height and width of the reference shape
θ	±2.5, ±5 degrees
scale	±2.5%, ±5%
c_{1-k}	±0.25, ±0.5 standard deviations

Table 7.1: Perturbation scheme used in both the view-based and light field AAMs.

discussed in Chapter 6 the model light fields are kept in the coordinate frame of the reference light field upon matching, thus we need only note the approximate depth of the reference light field to optimize the model. In our experiments we found values of $11 \leq z_0 \leq 12$ to work well for the approximate depth of the reference light field of both the optical flow and feature based models. Note that $k = 1$ views are used when rendering the model light field of the optical flow based model. This is because the reference light field of this model is slightly misaligned as a result of applying Algorithm 9 on each view separately, which we have done for efficiency. We have found using this value to work well in our experiments, however, this situation can be remedied by applying Algorithm 9 globally as suggested by the algorithm description.

For comparison, we built a view-based AAM using the views of the light field camera array [9]. In both the definition of the view-based and light field deformable models the parameter perturbations displayed in Table 7.1 were used to numerically compute the Jacobian matrix. To avoid over-fitting to noise, texture-shape PCA vectors having low variance were discarded from each model, the remaining PCA vectors modelling 90% of the total model variance.

We implemented the view-based AAM and light field deformable model in MATLAB. We outline our implementation and provide timing results in Section 7.5.

7.2 Comparison to a View-Based AAM

To compare our method to a view-based AAM we built a single-view 2D AAM and compared it against a feature-based light field deformable model. Each model was constructed using all fifty subjects, and was matched to various views of two people, the pose of the person in each view unknown. The resulting fits are displayed in

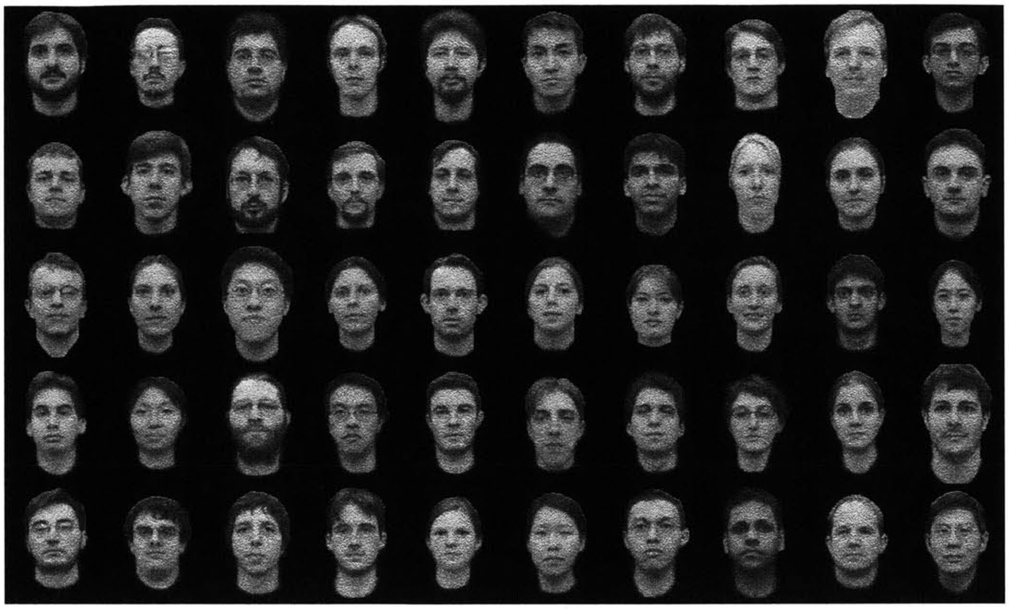


Figure 7-1: Fifty subjects used to train and test our light field deformable models.

Figures 7-8 and 7-9. In Figure 7-8 the person is wearing glasses which self-occlude the subject in extreme views of the camera array. These self-occlusions are difficult to model using a view-based AAM, where inter-pose variation is modelled as shape. Also note that the view-dependent texturing effects in the persons glasses are preserved by the light field deformable model, but are lost by the view-based AAM even though the person remains in the model.

In Figure 7-8 the performance between the view-based AAM and light field deformable model is gaged by how well they model the subject's eyes when glasses are present. In the case of a view-based AAM, close inspection of the eyes shows that the view-dependent specularities of the glasses are lost by the model and that the presence of glasses also introduces error in the fit as the eyes are awkwardly warped. Note this is not the case when fitting the subject without glasses. To emphasize this difference we performed the same experiment in color, since in grayscale light patches about the eye are sometimes confused for specularity in the glasses. The results of this experiment are shown in Figure 7-3, where we show the matches of each subject from the side pose of the first row of Figure 7-8. The fit of each color model is similar to that of the grayscale models of Figure 7-8, however, the specularity preserved by

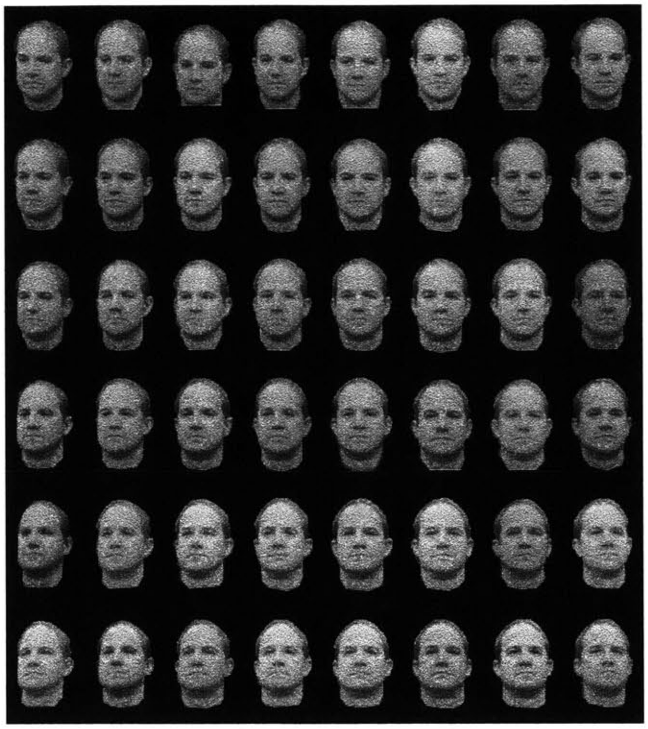


Figure 7-2: Example 6 × 8 light field captured using the light field camera array [41].

the light field deformable model and the errors introduced by the view-based AAM are more apparent.

The difference in performance between each model is explained by how they model pose variation. The view-based AAM blends the texture and shape of multiple poses at a given local-linear model. Thus, one would expect that inter-pose self-occlusion and view-dependent texture would not be properly modelled using this technique, unless many such local linear models are introduced rendering the model inefficient. The light field deformable model represents appearance in 4D, thus the shape and texture of each pose are kept separate and pose is an external parameter of the model. As a result the light field deformable model can easily handle the view-dependent texture and self-occlusions introduced by the glasses whereas the view-based AAM cannot.

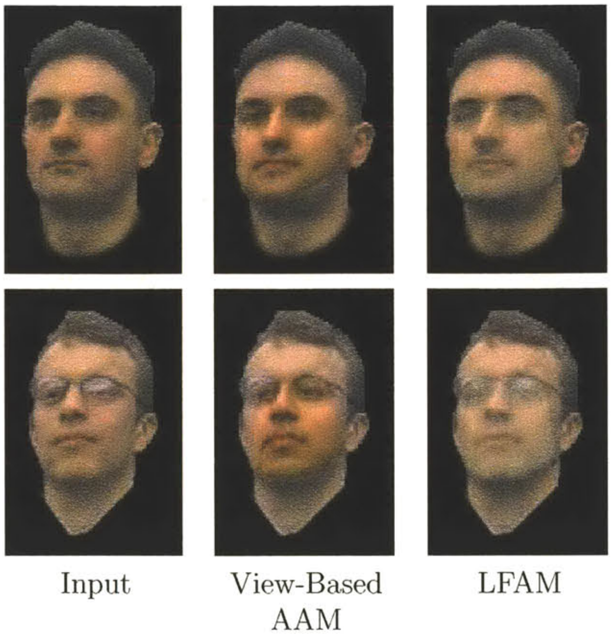


Figure 7-3: Comparison of Figures 7-8 and 7-9 using color models. [7]

7.3 Matching to a Light Field

In this section we demonstrate the ability of a light field deformable model constructed using either optical flow or feature-based shape vectors to be optimized over light fields of objects. We show results that optimize these models over images of objects in the following section.

7.3.1 Optical Flow Based Model

We built an optical flow based light field deformable model using 48 of the 50 subjects of Figure 7-1. Figure 7-4 displays select views of the light fields and resulting model fits of the two subjects kept out of the model. In the figure the model fit is superimposed onto ground truth. The figure illustrates the model's capability to generate convincing light fields of a novel input object. These fits, however, exhibit some error due to the ambiguity in the computed optical flow fields and the use of forward warping. Such errors are absent from the synthesized light field of the next subsection.

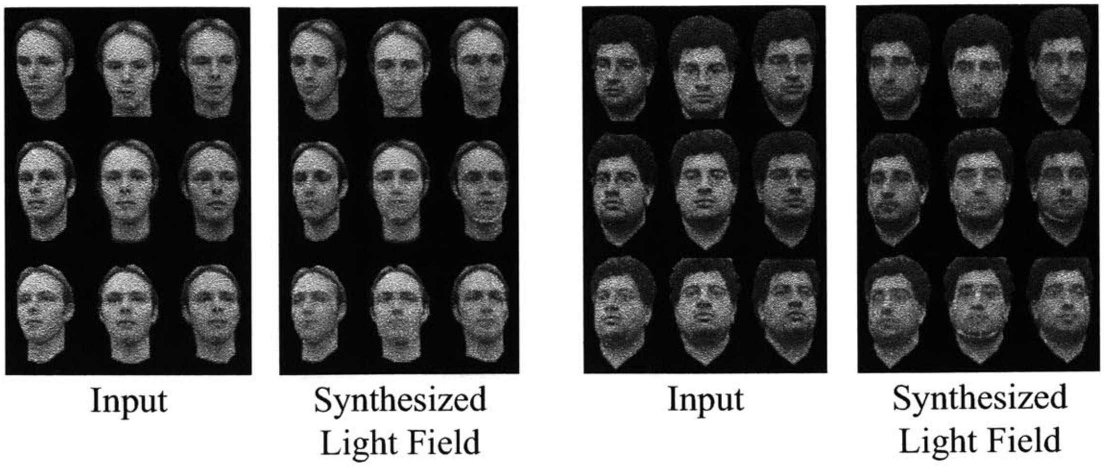


Figure 7-4: Optical flow based light field deformable model optimized over light fields of two subjects outside of the model database.

7.3.2 Feature-point Based Model

We built a feature-based light field deformable model using the same 48 of 50 subjects from the previous sub-section. Figure 7-5 displays select views of the synthesized light fields super-imposed onto ground truth using this model. Similar to the optical flow based model, this figure demonstrates that the feature-based model is able to generate convincing light fields of objects outside of the model database. Note that the fits of the feature-based model are more smooth and contain less error. The optical flow based technique relies on the online computation of optical flow for matching and uses forward warping for synthesis. The feature-based approach uses direct search to optimize the model that does not depend on online computed shape features to fit the model. Thus, although the fits of each figure are similar, those of the feature-based model appear more smooth. We further discuss these differences in more detail in the following chapter.

7.4 Matching to an Image

In this section we present light fields synthesized from 2D images of objects with unknown pose. We begin by providing empirical evidence for the use of a quadratic

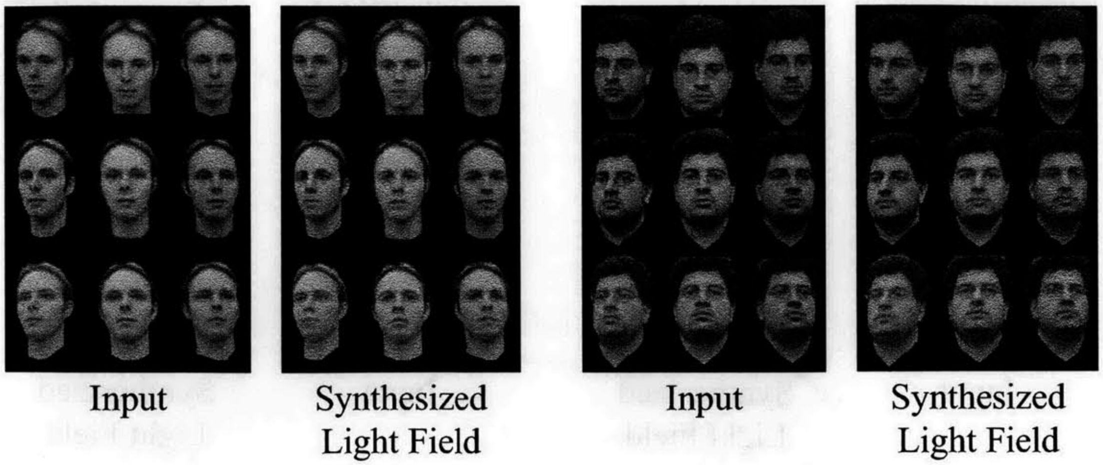


Figure 7-5: Feature-point based light field deformable model optimized over light fields of two subjects outside of the model database.

fit in the automatic pose estimation algorithm of Chapter 6. We then show light fields synthesized using both the optical flow based and feature-based light field deformable models optimized over 2D images of objects with unknown pose.

7.4.1 Automatic Pose Estimation

The automatic pose estimation algorithm of the previous chapter used a quadratic fit about a 3×3 neighborhood of the model light field to estimate object pose, where the center of this neighborhood corresponds to the discrete view of the model light field whose pose is closest to that of the input image. Let the RMS error of the model fit at estimated pose $\hat{\Theta}$ be given by the function $E(\Theta, \hat{\Theta})$, where Θ is the object's true pose. In this section we demonstrate that $E(\Theta, \hat{\Theta})$ is well approximated by a quadratic.

To accomplish this, we traversed the inner views of the model light field and computed $E(\Theta, \hat{\Theta})$ about a 5×5 neighborhood using the central view as input such that $\Theta = \Theta_c$, where Θ_c is the pose of the central view. The average value of $E(\Theta, \hat{\Theta})$ computed over 16 inner views of the model light field is displayed in Figure 7-6 from three different viewing points. The function is displayed in Figure 7-7 with a quadratic fit superimposed onto it. The fit quadratic is also displayed separately. As

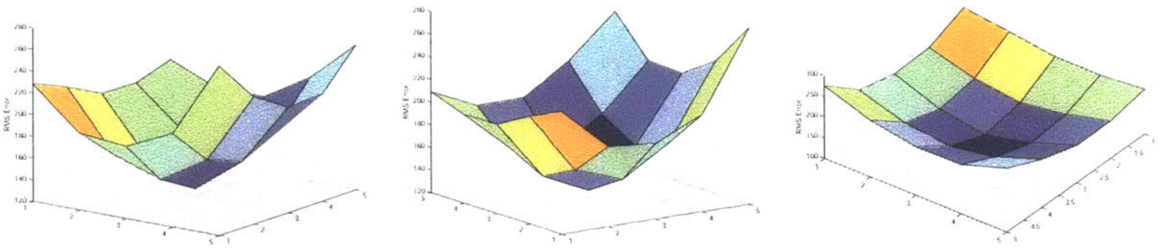


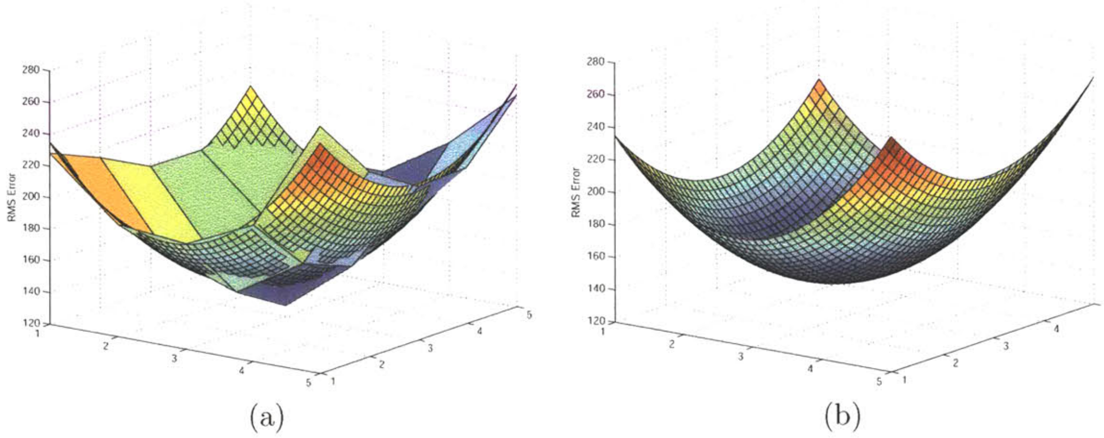
Figure 7-6: Average value of $E(\Theta, \hat{\Theta})$ computed about a 5×5 neighborhood of the model light field displayed from three different viewpoints.

demonstrated by these figures $E(\Theta, \hat{\Theta})$ is indeed well approximated by a quadratic fit. Note this experiment assumed that $E(\Theta, \hat{\Theta})$ is independent of the value of Θ. We believe this is a fair assumption, however, since the local, relative pose variation with respect to a central view should not strongly depend on the absolute pose of the central view.

Overall we found the use of a quadratic fit to work well in our experiments. Occasionally the fit would be well outside the 3×3 neighborhood of the converged view. In this case we performed a weighted average to estimate the object's pose. This event was rare, however, and in most cases a quadratic fit was used.

7.4.2 Optical Flow Based Model

We built an optical flow based light field deformable model using 46 of the 50 subjects of Figure 7-1 and fit the model to 2D images of 4 subjects kept out of the model, at various unknown poses. The resulting model fits and select views of the synthesized light fields superimposed on ground truth, along with the ground truth light field of each subject are provided in Figure 7-10. In the figure, the model is fit to each subject at two different poses for comparison, one of the poses frontal. Note our model was able to infer a full 4D light field from a single image with unknown object pose, the pose automatically estimated using our model. Comparing the synthesized light fields across poses of the same subject, one finds that although the model is optimized at different views the resulting light fields are quite similar. We demonstrate optimizing the feature-based light field deformable model over 2D images next.



(a)

(b)

Figure 7-7: Quadratic fit of $E(\Theta, \hat{\Theta})$ of Figure 7-6: (a) the quadratic superimposed onto $E(\Theta, \hat{\Theta})$, (b) the fit quadratic displayed on its own.

7.4.3 Feature-point Based Model

We built a feature-based light field deformable model using the 46 subjects of the previous sub-section. Figure 7-11 demonstrates fitting this model to 2D images of the 4 subjects kept out of the model, at various unknown poses. In Figure 7-11 the model is fit to two different poses for comparison, one frontal; select views of the synthesized light fields superimposed over ground truth are also displayed along with the ground truth light field of each subject. Similar to the optical flow based model, the feature-based model is able to generate convincing object light fields from single 2D images of novel objects captured under unknown pose.

Comparing Figures 7-10 and 7-11 one finds that each model performs quite similarly: the synthesized light fields resulting from each model are approximately the same. Such performance is expected since each model is trained on the same training set and, as seen from Chapter 6, each model is designed with the same framework using PCA. Close inspection of each figure shows that there are some minor differences between the fit of each algorithm, due to the different optimization techniques employed by each model as well as the use of different shape features. For example, the optical flow based model has difficulty about the edges of the face due to ambiguity in the optical flow, however, as illustrated by the figures these errors are minor. We

Routine	Average Execution Time (sec)
Light Field Fit	26
Discrete Pose Fit	1
Arbitrary Pose Fit	19
Fit to Unknown Pose	120

Table 7.2: Execution times of the routines used to optimize the optical flow based light field deformable model. All times are rounded up to the nearest second.

further discuss this issue in the following chapter.

Figures 7-10 and 7-11 seem to suggest that our model does not represent glasses too well (see the second subject). We believe, however, that this is a different issue than that discussed in Section 7.2 where we compared our model against a view-based AAM. In Section 7.2 the subject was in the model and thus the glasses and view-dependent specularities were recovered. In these fits the subject is outside of the model and thus it is more difficult to optimize over subjects wearing glasses, especially because the models contain both subjects with and without glasses and the glasses worn across subjects have different surface reflectance properties and unconstrained shape (see Figure 7-1). To properly handle the variation in appearance due to the presence/absence of glasses, ideally we would extend our approach to have the ability to separately cluster the examples with and without glasses and then match to each cluster separately. The investigation of such non-linear models is an interesting area of future work that is discussed in the next chapter.

7.5 Algorithm Implementation and Performance

We have implemented both the optical flow based and feature-based models in MATLAB. The implementation of each model is organized into routines that build the model, to a light field, fit the model to a discrete pose of the model light field, to an arbitrary pose within the model light field and a pose estimation module that utilizes the image matching routines to optimize the model over an image of an object with unknown pose. We report execution times for the above components of each model.

Tables 7.2 and 7.3 display execution times for the optical flow based and feature-

Routine	**Average Execution Time (sec)**	**Average Number of Iterations**
Light Field Fit	45	6
Discrete Pose Fit	4	5
Arbitrary Pose Fit	11	6
Rendering Jacobian	7	NA
Fit to Unknown Pose	225	NA

Table 7.3: Execution times and iteration counts for the routines used to optimize the feature-based light field deformable model. All times and iteration counts are rounded up to the nearest second or count.

based light field deformable models respectively, run on a Pentium 4, 2.0 GHz processor with 768 MB of memory. The training times for these models are omitted as they are on the order of a few hours and can be performed offline. The average execution times reported in these tables were computed over the model fits displayed in the figures of this chapter. In these tables the execution time of each component is reported. We discuss each of these components below.

Of all the routines, those that involve rendering take the longest. Another bottleneck is the piecewise image warping. In our implementation, both the rendering and warping engines were implemented using C/C++ for reasonable performance, however, they were done without the use of optimized graphics hardware. With the use of graphics hardware both of these operations can be performed in real-time [34, 25, 6].

Observing the average number of iterations required to match the feature-based model to a light field or 2D image, one finds that these numbers are similar to previously reported iteration counts for algorithms that employ direct search [8, 34]. The direct search algorithm used by the feature-based model is known to exhibit real-time performance [34]. The main difference between their implementation and ours is that we use a rendering engine to synthesize the model at arbitrary poses. As discussed above, this module can be made real-time using graphics hardware. We therefore expect that light deformable models, if efficiently realized, can be optimized in real-time.

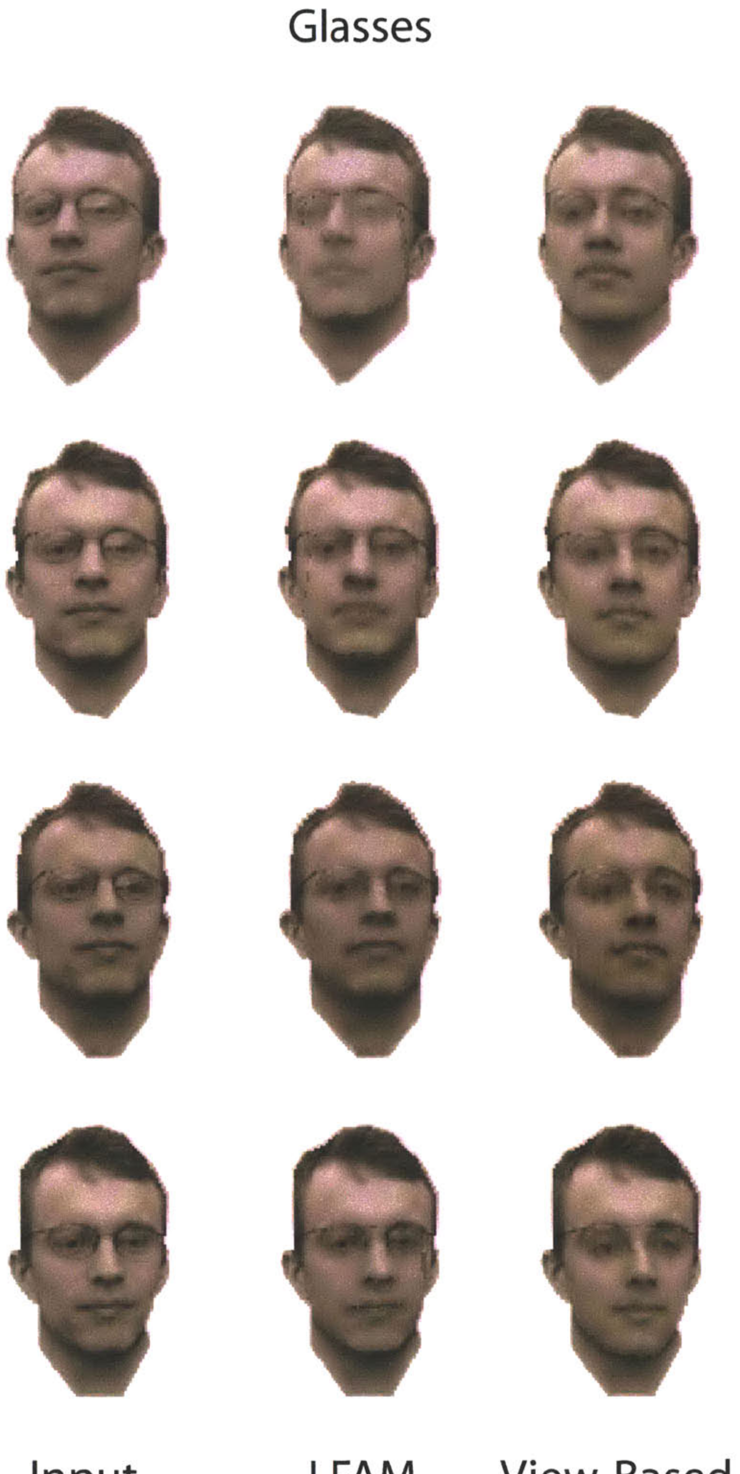


Figure 7-8: Comparison of a light field deformable model to a view-based AAM. The left column shows the input, the right column the best fit with a 2D AAM, and the middle column the light field fit. When glasses are present the 2D method fails and the light field appearance model succeeds.

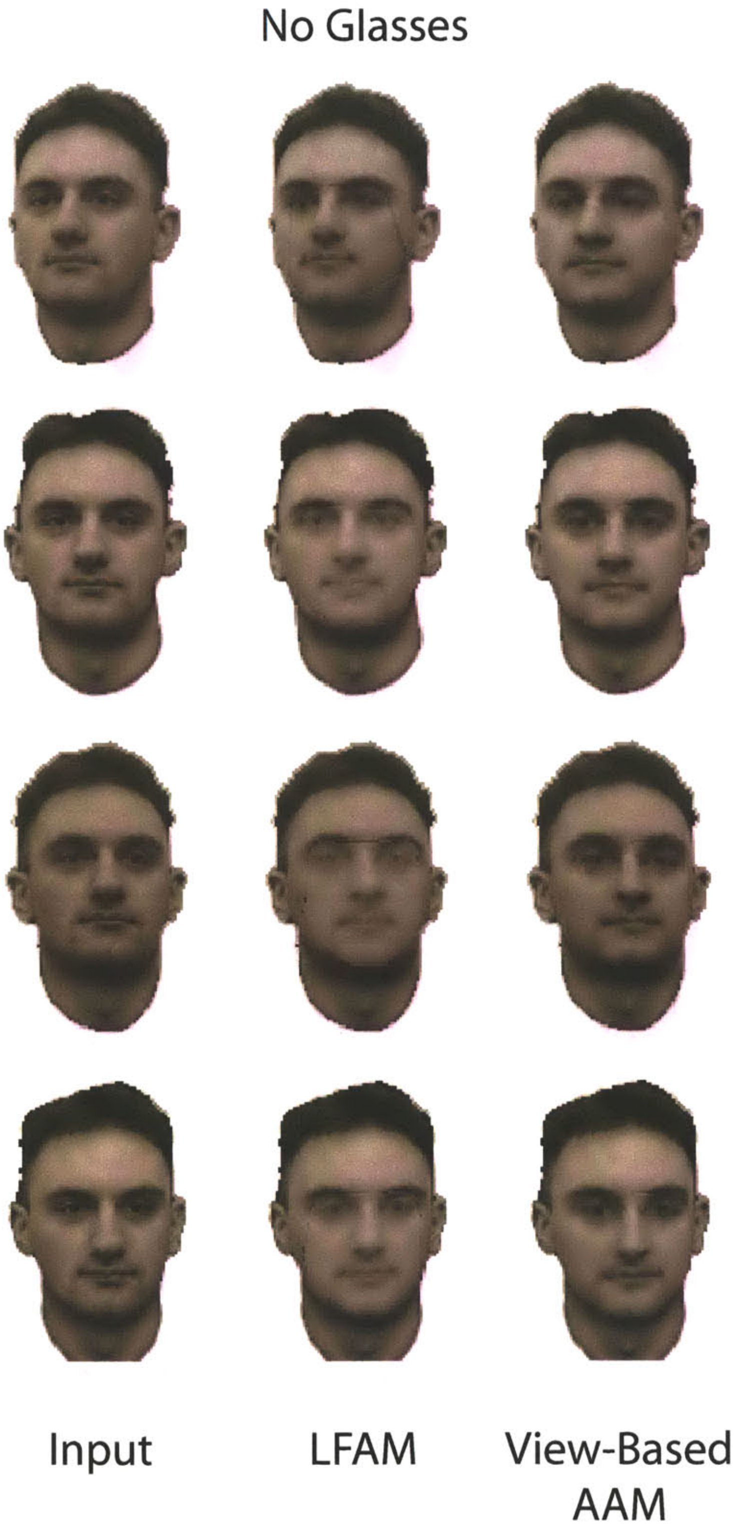


Figure 7-9: Comparison of a light field deformable model to a view-based AAM. The left column shows the input, the right column the best fit with a 2D AAM, and the middle column the light field fit. The 2D and light field appearance models both exhibit qualitatively good fits when the surface is approximately smooth and lambertian.

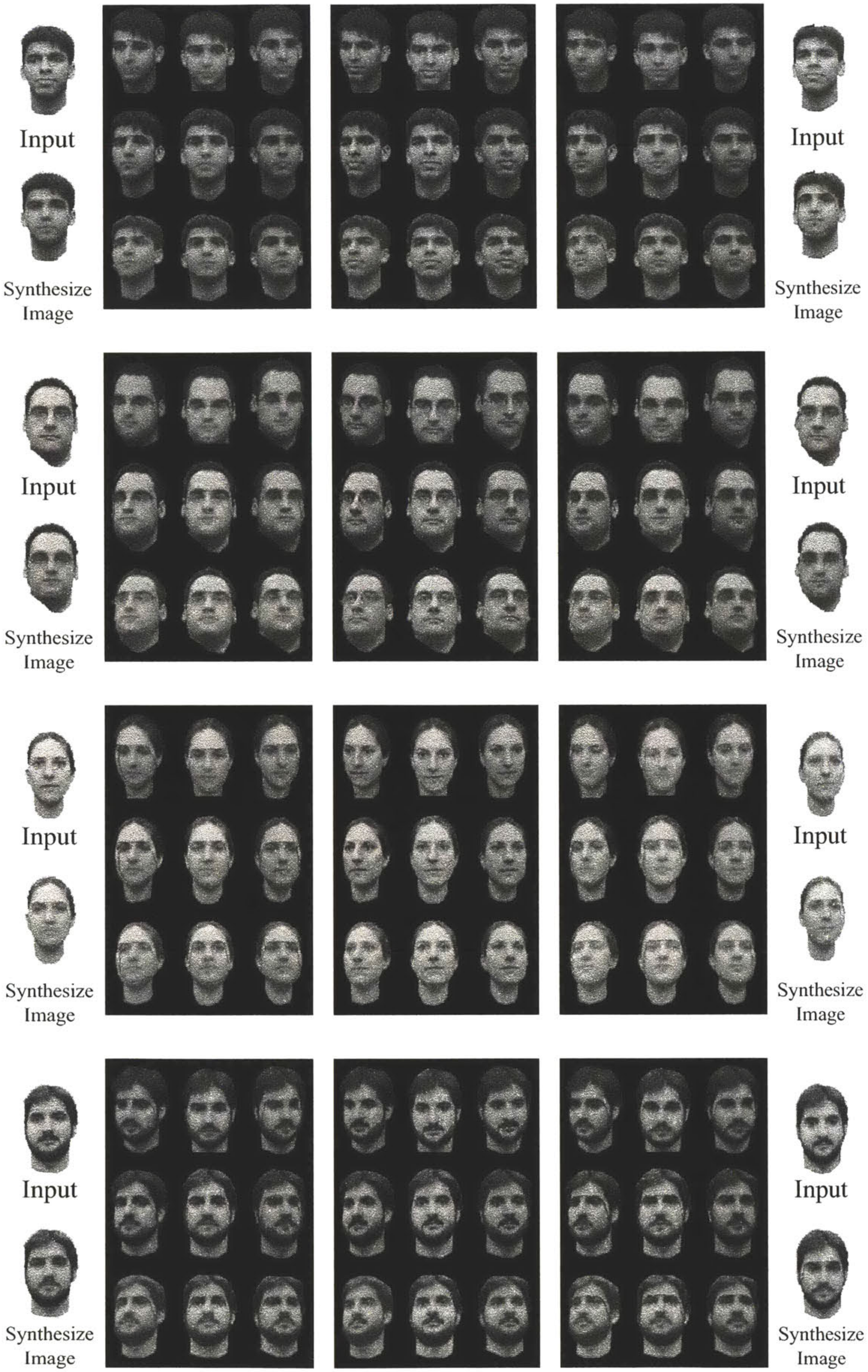


Figure 7-10: Optical flow based light field deformable model optimized over images of objects with unknown pose. The model was optimized over 4 subjects removed from the model database. Our method is able to synthesize convincing light fields from a single input image. Optimizing the model over different poses of the same subjects gives light fields that are strikingly similar.

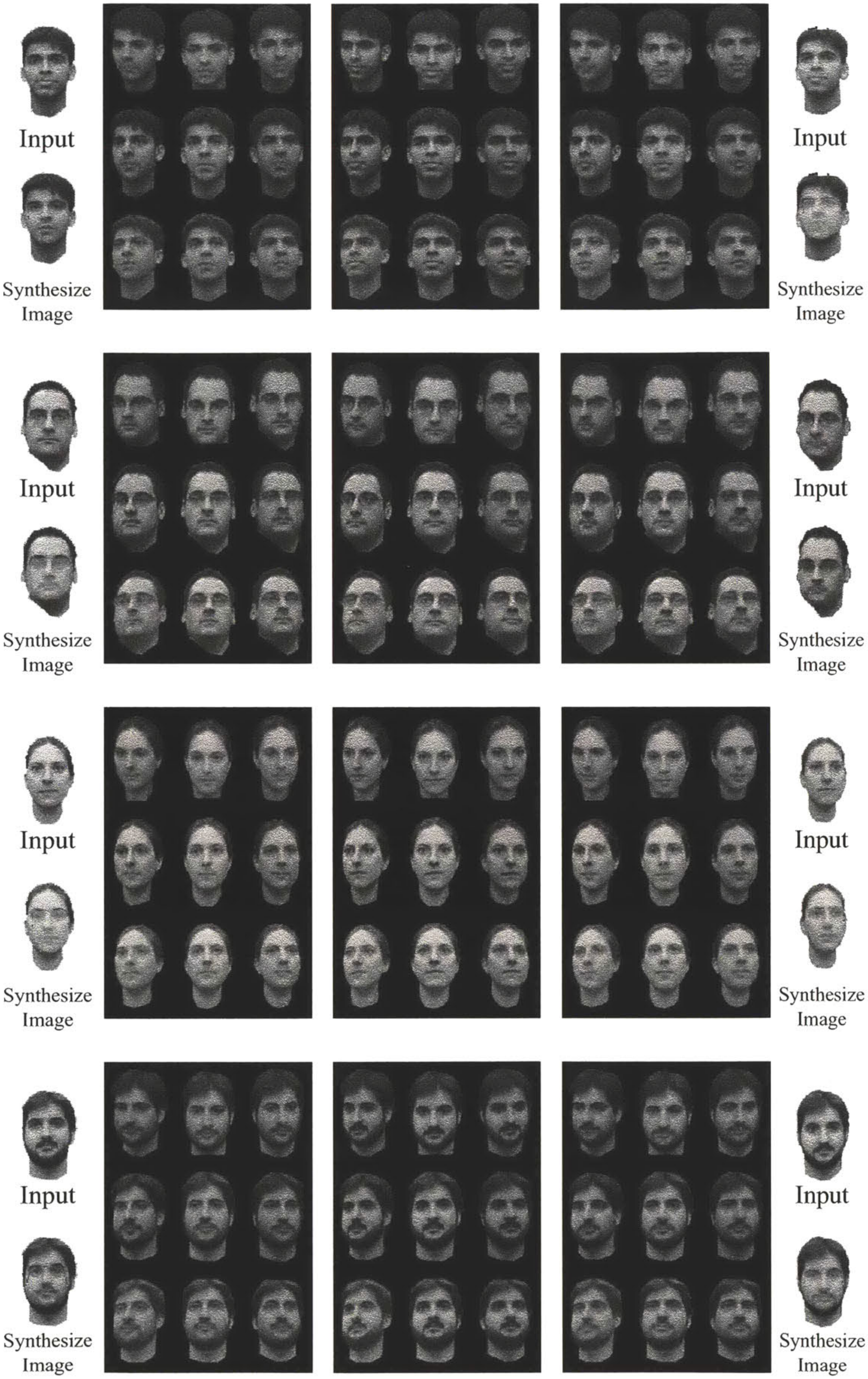


Figure 7-11: Feature-point based light field deformable model optimized over images of objects with unknown pose. The model was optimized over 4 subjects removed from the model database. The feature-based method obtains higher fitting accuracy compared to the optical flow based method.

Chapter 8

Discussion and Future Work

This chapter provides an overview of the work performed and discusses possible applications and extensions of the ideas presented in this thesis. The contributions of this thesis are summarized in Section 8.1. Applications of light field deformable models include 3D animation and pose-invariant face recognition. These and other applications of our work are presented in Section 8.2. Finally, in Section 8.3 we outline how our method can be improved and extended, and discuss interesting avenues for future work.

8.1 Contributions

In this thesis we have presented the concept of a *light field deformable model*. Light fields offer a 4D representation of appearance that model the scene with densely sampled imagery and, unlike other image-based rendering approaches, they model the scene without the use of any scene geometry. We have shown that light field deformable models can easily model object classes exhibiting complex surface reflectance and geometry. Also, our method is able to easily optimize over the pose of the imaged input object, as pose is kept as an external parameter to the model.

To realize our model we introduced the notion of a *light field appearance manifold* defined over 4D shape and texture vectors of prototype object light fields. We then demonstrated how to match light fields or 2D images of novel objects with unknown pose to this manifold. The light field appearance manifold was defined using both manually specified point features and automatically computed optical flow shape vectors. A different optimization algorithm was proposed for each model, however, both methods used the same pose estimation framework.

We adopted a method similar to [3] to optimize the optical flow based model, where online optical flow shape vectors are computed to fit the model. To fit the feature based model we extended the direct search algorithm of [8] to function over the domain of light fields. With our method the Jacobian has light field intensity gradients as its columns and to match the model to an image, the Jacobian is rendered at the pose of the input view.

In our experiments we demonstrated the construction of a light field deformable model using both optical flow and feature point based shape vectors and fit each model to light fields and 2D images of novel objects with unknown pose. We compared our method to the view-based AAM [9], a 2D approach that models pose variation by defining local-linear models in the different regions of pose space, such that in each local-linear model the resulting appearance model is well approximated as linear. When an object class exhibits complex surface reflectance with view-dependent texturing effects and/or complex geometry these models tend to break down as many local-linear models become warranted rendering such models in-efficient.

In contrast, we demonstrated that light field deformable models can easily handle such phenomena. We showed this by matching to a subject with glasses. With our model the glasses were recovered and the view-dependent texturing effects preserved, whereas the view-based AAM had difficulties. In particular the view-based AAM was unable to represent the view-dependent specularities in the glasses, and the inter-pose self occlusion caused by the presence of glasses introduced errors in the fits resulting from the view-based AAM. The main reason for the difference in performance between the two approaches is that the view-based AAM blends images from many poses at a single local linear model, whereas the light field appearance model does not. Instead with our approach object appearance is represented using 4D and thus it can easily model the complex surface reflectance and geometry introduced by the presence of glasses.

Light field deformable models have many advantages over existing 2D and 3D shape and texture appearance models. Two dimensional AAMs [8] and MMMs [24] cannot model large changes in pose variation, as this results in non-linear differences in appearance. Extensions to these models such as the view-based AAM [9] are able to model pose but as discussed above have difficulty modelling object classes that do not exhibit smooth, Lambertian surfaces. Pose variation is easily handled with 3D deformable models [4] where pose is kept as an external parameter of the model. Such methods, however, use simply textured 3D meshes and rely on high accuracy range scans. Our model represents object appearance in 4D with light fields, a purely image based approach, that does not require knowledge of any scene geometry. As such, light field deformable models can easily represent object classes with complex surface reflectance and geometry, unlike existing 2D and 3D approaches.

In this thesis we have presented most of the ground work for defining 4D deformable models using light fields. We believe there are still many important and interesting extensions to our work, such as BRDF modelling to handle arbitrary lighting and the use of alternative IBR techniques that require less imagery. We discuss these and other possible improvements/extensions to light field deformable models in Section 8.3.

8.2 Applications

Light field deformable models have numerous applications in computer vision and graphics. We list and briefly discuss some of these applications below.

In computer graphics, light field deformable models are useful for the 3D animation of virtual characters or avatars [14]. With a light field deformable model, a 4D representation of appearance can be constructed for a particular object class (e.g. faces) and the coefficients of the model labelled with class specific information as was done in [4]. Provided an image of an object, a light field of the object could be recovered along with class specific model parameters. An animator could then adjust these parameters, that for the case of people, could make the subject look more young/old, male/female, etc. The animator could also display and animate the object from previously unseen views. Note the difference between using our model and the model in [4] is that the object classes can exhibit more complex surface reflectance and geometry giving the animator more freedom in the choice of object class to represent.

Other applications of light field deformable models include object segmentation and non-rigid body tracking. Two dimensional deformable models have proven to be useful in these tasks [34, 32]. Using a 4D deformable model, the input object can take any pose, thus providing a more flexible solution to these problems.

Light field deformable models also have application in pose-invariant object recognition. In [20], Gross et. al. showed how *Eigen light fields* can be used to achieve accurate pose-invariant face recognition. In their paper, they specify feature points on the training and input images to normalize their data to a common coordinate frame. They then performed PCA on the normalized light field data to construct and optimize their model. Light field deformable models model both light field shape and texture variation and thus using our approach the input image need not be normalized, since the shape of the input object is recovered using our model. We believe extending their work to use a light field deformable model will make the recognition processes more automated.

8.3 Future Work

This thesis presents most of the ground work for light field deformable models, however, we believe that there are many ways in which our model can be improved and extended to increase its performance and utility. We begin this section by discussing some of these improvements and extensions. We then discuss other interesting avenues of future work related to shape and texture appearance models that we hope to address in the near future.

There are a number of improvements that could be made to our existing algorithm.

One improvement is to use stochastic gradient descent [24] or direct search in place of the online optical flow computation to optimize the optical flow based light field deformable model. This would avoid errors introduced by the input optical flow field. These techniques are also less sensitive to scene clutter. Naturally, another source of error is in the flow fields used to represent the shape of the prototype light fields. In [24], Jones and Poggio present an algorithm that normalizes the shape vectors to correct for error in the optical flow. This algorithm could also be applied to our model.

Another improvement is in the pose estimation algorithm. To estimate the pose of the input object we fit a quadratic to the model error values centered about the converged discrete pose of the model light field. Although this worked well in our experiments, a more general technique would be to incorporate pose estimation as part of the direct search, where we would learn image intensity gradients parameterized over object pose. We believe that this would give more accurate sub-view pose estimates and would result in improved light field synthesis.

In [27], Matthews and Baker present a provably optimal direct search technique for optimizing a 2D AAM. The direct search algorithm of Cootes and Taylor assumes that the model Jacobian is constant with respect to the value of the model parameters. In their work, Matthews and Baker demonstrate that this assumption is false in general and present an efficient direct search algorithm based on inverse compositional image alignment that is provably optimal. To optimize the model, they first separate the model objective into two parts, one that is independent of appearance variation and the other independent of shape variation and they optimize each part separately. To optimize over shape they update the model warp field as oppose to shape parameters by composing warp fields; they show that using this method the computed shape Jacobian is always evaluated at zero using this method and thus can be safely assumed constant and pre-computed. Given the shape parameters the second part of the objective, involving texture variation, is optimized using linear least squares.

This new direct search technique has several advantages over previous methods. In addition to being provably optimal, it achieves greater efficiency by separately optimizing over shape and texture. Matthews and Baker also show that by separately optimizing shape, the corresponding model Jacobian can be analytically computed to give much better results. Extending light field deformable models to use this new direct search algorithm is an interesting avenue for future work as it can increase model fitting accuracy and efficiency.

The head database used in our experiments suitably demonstrated the capabilities of a light field deformable model. It would be interesting, however, to collect multi-light slab light field databases of objects that exhibit more complicated surface reflectance and/or geometry, captured over a larger pose variation. Such databases would further emphasize the strengths of a light field deformable model and may lead to interesting applications in computer graphics.

A drawback of such databases, and light field deformable models in general, is the amount of imagery necessary to represent objects across large pose variation. An interesting area of future research is the investigation of alternative IBR methods for the construction of deformable models. With these methods, a rough geometric proxy can be used to decrease the number of images necessary to model the scene. Such approaches may not exhibit the full capabilities of a light field deformable model in the extent to which they can represent non-Lambertian objects and complex geometry, however, the practical advantages of these models may tradeoff the decrease in representation power.

Another exciting extension of our work would be the incorporation of BRDF models for representing objects imaged under varying illumination. In [16], Georghiades et. al. prove that the space of images of an object under all possible illuminations is defined by a convex cone. Depending on the complexity of the object's BRDF the appearance of an object under all illuminations can be represented quite compactly by picking images that lie on the boundary of this cone. It would be interesting to see how this approach generalizes to light fields of objects and what implications this may have for defining light field deformable models that represent objects under varying illumination.

In the more direct future we are investigating deformable models built to handle object dynamics and topological deformations. In doing so, we are studying the use of non-linear manifold learning methods that can segregate the appearance manifold into its meaningful components and restrict model search to the valid portions of the manifold. We hope that this work in coordination with our work on light field deformable models will enrich the utility of shape and texture appearance models.

Bibliography

[1] E. H. Adelson and J.R. Bergen. *Computation Models of Visual Processing*, chapter The Plenoptic Function and the Elements of Early Vision. MIT Press, Cambridge, 1991.

[2] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In *Computer Graphics (Proceedings of SIGGRAPH 92)*.

[3] David Beymer and Tomaso Poggio. Face recognition from one example view. Technical Report AIM-1536, September 1995.

[4] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3D faces. In *SIGGRAPH*, pages 187–194, Los Angeles, 1999.

[5] M.E. Brand. Morphable 3D models from video. In *CVPR*, May 2001.

[6] Chris Buehler, Michael Bosse, Leonard McMillan, Steven J. Gortler, and Michael F. Cohen. Unstructured lumigraph rendering. In *SIGGRAPH*, pages 425–432, 2001.

[7] C. Mario Christoudias, Louis-Philippe Morency, and Trevor Darrell. Light field appearance manifolds. In Tomas Pajdla and Jiri Matas, editors, *The 8th European Conference on Computer Vision*, volume 3024 of *Lecture Notes in Computer Science*, pages 481–493, Prague, Czech Republic, June 2004. Springer-Verlag.

[8] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. *Lecture Notes in Computer Science*, 1407:484–98, 1998.

[9] T. F. Cootes, G. V. Wheeler, K. N. Walker, and C. J. Taylor. View-based active appearance models. *Image and Vision Computing*, 20:657–664, 2002.

[10] T.F. Cootes and C.J. Taylor. Statistical models of appearance for computer vision. Technical report, University of Manchester, October 2001.

[11] Trevor Darrell. Example-based image synthesis of articulated figures. In MIT Press, editor, *Neural Information Processing Systems*, volume 11, 1998.

[12] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *Proceedings of SIGGRAPH 96*, pages 11–20, August 1996. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

[13] G. Edwards, C. Taylor, and T. Cootes. Interpreting face images using active appearance models. In *3rd International Conference on Automatic Face and Gesture Recognition*, pages 300–305, 1998.

[14] Tony Ezzat, Gadi Geiger, and Tomaso Poggio. Trainable videorealistic speech animation. In *Proceedings of ACM SIGGRAPH 2002*, San Antonio, Texas, July 2002.

[15] P. Fua and C. Miccio. From regular images to animated heads: a least squares approach. In *ECCV*, pages 188–202, Springer,Berlin, 1999.

[16] Athinodoros Georghiades, Peter Belhumeur, and David Kriegman. From few to many: Generative models for recognition under variable pose and illumination. *IEEE PAMI*, 23(6):643–660, June 2001.

[17] Jonas Gomes, Bruno Costa, Lucia Darsa, and Luiz Velho. *Warping and Image Morphing of Graphical Objects*. Morgan Kaufmann, 1998.

[18] Colin Goodall. Procustes methods in the statistical analysis of shape. *Journal of the Royal Statistical Society*, 53(2):285–339, 1991.

[19] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *Computer Graphics*, 30(Annual Conference Series):43–54, 1996.

[20] R. Gross, I. Matthews, and S. Baker. Appearance-based face recognition and light fields. *IEEE PAMI*, 26(4), April 2004.

[21] Intel. *Intel Math Kernel Library (MKL).* http://www.intel.com/software/products/mkl/.

[22] I. T. Jolliffe. *Principle Component Analysis.* Springer-Verlag, New York, 1986.

[23] Michael Jones and Paul Viola. Fast multi-view face detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 2003.

[24] Michael J. Jones and Tomaso Poggio. Multidimensional morphable models. In *ICCV*, pages 683–688, 1998.

[25] Marc Levoy and Pat Hanrahan. Light field rendering. *Computer Graphics*, 30:31–42, 1996.

[26] B. D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

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

[28] B. Moghaddam and A. Pentland. Probabilistic visual learning for object recognition. *IEEE PAMI*, 19(7):696–710, 1997.

[29] H. Murase and S. Nayar. Visual learning and recognition of 3-d objects from appearance. *IJCV*, 14(1):5–24, 1995.

[30] Frederic H. Pighin, Richard Szeliski, and David Salesin. Resynthesizing facial animation through 3d model-based tracking. In *ICCV*, pages 143–150, 1999.

[31] S. Romdhani, S. Gong, and A. Psarrou. A multi-view nonlinear active shape model using kernel pca. In *British Machine Vision Conference*, pages 483–492, 1999.

[32] Stan Sclaroff and John Isidoro. Active blobs. In *ICCV*, Mumbai,India, 1998.

[33] Steven M. Seitz and Charles R. Dyer. View morphing. *Computer Graphics*, 30(Annual Conference Series):21–30, 1996.

[34] M. B. Stegmann. Analysis and segmentation of face images using point annotations and linear subspace techniques. Technical report, Technical University of Denmark, DTU, August 2002.

[35] Trolltech. *QT*. http://www.trolltech.com.

[36] M. Turk and A. Pentland. Eigen faces for recognition. *Journal of Cognitive Neuroscience*, 3(1), 1991.

[37] Thomas Vetter, Michael J. Jones, and Tomaso Poggio. A bootstrapping algorithm for learning linear models of object classes. Technical Report AIM-1600, 1997.

[38] Bennett Wilburn, Michael Smulski, Hsiao-Heng Kelin Lee, and Mark Horowitz. The light field video camera. In *Proceedings of Media Processors 2002, SPIE Electronic Imaging*, 2002.

[39] Matusik Wojciech, Christopher Buehler, Ramesh Raskar, Leonard McMillan, and Steven J. Gortler. Image-based visual hulls. In *SIGGRAPH*, 2000.

[40] George Wolberg. Image morphing: a survey. *The Visual Computer*, 14(8-9):360–372, 1998.

[41] Jason C. Yang, Matthew Everett, Chris Buehler, and Leonard McMillan. A real-time distributed light field camera. In *Eurographics Workshop on Rendering*, pages 1–10, 2002.

[42] Zhunping Zhang, Lifeng Wang, Baining Guo, and Heung-Yeung Shum. Feature-based light field morphing. In *Conference on Computer graphics and interactive techniques*, pages 457–464. ACM Press, 2002.