I would like to thank Dr. Andrew Fitzgibbon for his tremendous supervision and guidance this year, and Dr. David Ranyard at Sony Computer Entertainment Europe for his interest and advice. The Visual Geometry Group has been a friendly and stimulating environment, particular thanks to Prof. Andrew Zisserman and Dr. Frederik Schaffalitzky. My gratitude to family and friends for their love and support.

Acknowledgements
# Contents

1 Introduction  

2 Literature review  

3 Implementation  

3.1 Overview  

3.2 Naive Attempt  

3.3 2D correspondence  

3.3.1 Dense flow correspondence  

3.3.2 Flow computation at interest points  

3.3.3 User-specified control points  

3.3.4 Flow field interpolation  

3.4 Obtaining correspondence for 3D models  

3.4.1 Resampling the 3D model  

3.4.2 First attempt: Resampling at regular intervals  

3.4.3 Second attempt: User defined resampling  

3.4.4 The model viewer revisited  

4 2D matching  

4.1 Overview  

4.1.1 Obtaining 3D from 2D  

4.2 Feature based matching  

4.2.1 Overview of the POSIT algorithm  

4.2.2 Application of POSIT to linear object spaces  

4.2.3 Finding initial estimates for $a$  

4.3 Pixel based matching  

4.3.1 Mutual information  

4.3.2 Implementing mutual information  

4.3.3 Testing mutual information’s suitability as a metric  

4.4 Search algorithms  

4.4.1 Gradient ascent method  

4.4.2 Levenberg-Marquardt method  

4.4.3 Downhill simplex method
4.5 Testing methodology, results and conclusions ........................................... 46
4.5.1 Finding starting conditions for a ......................................................... 46

5 Flexible bump map capture 50
5.1 Introduction ................................................................. 50
  5.1.1 Photometric stereo ......................................................... 51
  5.1.2 Existing techniques for bump map capture .............................. 52
  5.1.3 Flexible bump map capture ................................................... 53
5.2 Details ................................................................. 54
  5.2.1 Estimating camera position ..................................................... 54
  5.2.2 Nonlinear refinement of reprojection error .............................. 56
  5.2.3 Estimating light position ......................................................... 59
  5.2.4 Tracking the markers ............................................................ 60
  5.2.5 The capture process ............................................................. 61
5.3 Conclusions ............................................................. 63

6 Future Work ............................................................. 65

A Lectures, meetings, seminars ......................................................... 68
Chapter 1

Introduction

Increasingly, we see the fields of Computer Vision (CV) and Computer Graphics (CG) interacting and merging. With photorealistic rendering now the usual target for new CG techniques, it is still the machine parsing of real-world images that is the objective for CV. However, the relatively crude rendering environments from the early years of computing are now gone, and we begin to see how approximating algorithms from CG can now be used in CV to hypothesise about predicted visual properties in images. Analogous to this is the use of CV techniques to extract data for computer rendering, such as texture and shape capture.

With rigid-body, simple-surface modelling now photorealistic, perhaps the most challenging areas of CG remain in simulating organic, natural objects. This is particularly true of animals and people, which have complicated internal (bones, organs) and external surface structure (hair, skin). With a joint sponsorship from Sony Computer Entertainment Europe, the focus of this work is on real-time rendering, with a consideration of typical home or desk computing capabilities in the near-future. I aim to produce a system for the life-like generation, animation, and interpretation of human faces. This will involve research into the production of a *morphable model* for CG human faces, capable of generating large scale surface information for many different human faces from a small set of control parameters. We will also need to examine CV and CG techniques to simulate small scale features of skin, such as pores and wrinkles beyond the scale of the morphable model. With an eye on current graphics technology we observe that rendering power will never be infinite, and so our solution must be computationally feasible.

The second goal is to combine the morphable model with a *matching* technique, which will allow it to be aligned with a real-life face from captured video. Various interesting applications are apparent for a morphable model with such a matching system, including tracking, recognition, compression and real time animation:

- Face tracking/recognition—Given coefficients for a known face we can search for it in an image by reducing the error between the image and the rendered model. This will provide translational and rotational information about a face in a 2D image.
- Face alteration—Given a particular face $F$, we can progressively blend in other
faces with interesting properties to give \( F \) those properties, for example ‘fattening’, ‘thinning’, making a face more masculine or adding and altering expression.

- Face animation—Given two models of the same face with different expressions, (for example straight-faced and smiling), we can extract out an ‘expression vector’ which approximates the change from a straight face to that expression. This vector can then be applied to a different face model in gradually increasing amounts to animate that expression.

- Face generation—Given a suitably large input set of models, thousands of different faces can be constructed via linear combination. This has the potential to create virtual worlds (an example being a computer game environment) where every character can look different, eliminating the common ‘everyone is a clone’ problem present in many recent games.

- Face compression—By computing the nearest approximation to a given face in our model we need no longer store the 3D model of the face itself, just the coefficients of the model required to construct the approximation and its position in space. This has obvious application in very low-bandwidth video communication.

The following report describes the work carried out so far in the implementation and application of the morphable model outlined above. Chapter 2 contains a short review of relevant scientific literature. Chapter 3 details the process of constructing the model (more strictly a ‘linear object space’), which involves the development of a novel ‘3D via 2D’ correspondence technique. Chapter 4 describes the current state of the matching system, which covers the different function optimisation and 2D image matching techniques applied during its various revisions. As discussed above, the model alone will be insufficient to prove truly photorealistic rendering—we must look as well at the surface properties of human faces. As such, Chapter 5 discusses work carried out on this second strand of research, applying CV techniques to capture real world surface data from simple objects. Finally, Chapter 6 summarises the work carried out, as well as providing a plan for the continuation of my research over the following two years.
Chapter 2

Literature review

Human face rendering and animation

With our overall objective being the realistic simulation and animation of the human face, it is valuable to perform a review of past work in this area. Among the earliest attempts is Parke’s work on the creation of a parameterized polygonal representation of the human face [68], essentially an abstraction of human facial features. More realistic systems were to follow, a notable example being Waters’ muscle model for face animation, which used a spring model to simulate facial expression [90]. An alternative to physics-based models, image-based techniques such as that of [91] have also enjoyed success. Here, 2D and 3D motion is recovered via analysis of live performances from actors. Noh et al [64] produced a system capable of ‘expression cloning’ across 3D facial models, based on computing vertex correspondence with a 3D model undergoing an expression change. Simulation of expression on 2D images has also improved, with [54] using a model of facial structure to introduce realistic wrinkles during the warping of a facial image.

Recent attempts have improved on the coarse geometry of early work, using a variety of active and passive techniques to obtain more realistic models. Lee and colleagues were among the first to introduce very high resolution, texture mapped, Cyberware scanned, 3D models [47]. Guenther et al proposed a system for obtaining 3D face geometry from multiple 2D images [31]. Vetter and Blanz combined Cyberware geometry with a sophisticated ‘morphable model’ and image matching to produce a system capable of high quality visualisation of a large range of faces [9]. Fua introduces a technique for acquiring high resolution geometry from images using a generic face model and bundle adjustment [25], with Schlesinger and Flach’s improvements in feature matching techniques achieving remarkably good reconstruction from a single stereo pair of images [81].

Along with expression, visual simulation of speech is a popular research area. [11] uses a set of 2D mouth images captured from video, blending between them to simulate speech. This has been refined recently by Ezzat, Geiger, and Poggio, through the use of a 2D morphable model (analogous to [9]) of the human face, which can produce much more realistic ‘inbetween’ frames from a given set of mouth shape images [23]. Lee et al improve the simulation of human speech via a statistical analysis of human eye movement (saccades) during patterns of conversation, expressed as MPEG-4 facial parameters [46].
Although not directly related to our area of interest, simulation of hair is important for a true photographic realism. We omit a full discussion of work here, but note that recent papers such as [41] are very encouraging.

**Real world model capture**

Even though our main focus remains the acquisition of realistic face models, it is useful to take a brief look at significant work on 3D geometry capture in general. Whilst depth information can be gleaned via CV techniques, devices exist that can capture 3D information about a scene directly. Pentland and colleagues introduced a simple camera capable of capture of range information using defocusing techniques [70], a real-time refinement of this technique being [61]. With captured range data available a significant problem is the registration and combination of multiple range images into a coherent model—Turk et al provide an influential solution [85]. The development of modern computer vision techniques led to new ways to acquire range data from stereo pairs of images, an example being [38]. With time 3D model capture has been refined to capture more complex properties than just large scale geometry—Rushmeier et al use a coarse geometrical approximation, acquiring bump and texture maps to provide fine detail [77]. Sato and colleagues combine range and colour information to separates diffuse and specular lighting effects, leading to an automated system for retrieving texture and reflectance properties, as well as model geometry [80]. Perhaps the most interesting challenge seen recently is that of very large scale capture, such as ‘The digital Michelangelo project’ [50], which aims to obtain high resolution 3D data of full size statues.

**Imaged based rendering**

The systems described above all use large scale 3D geometry, perhaps with 2D approximations of surface detail to realise an object. An alternative but converging field is that of image based rendering—here a series of 2D images of an object are captured, processed and combined to produce novel views of the object. Historically, two very similar papers providing an implementation of this technique were published in 1996, by Levoy and Hanrahan [49] and Gortler et al[30]—the primary difference appears to be in Gortler’s use of coarse geometrical approximation to aid pose reconstruction, a technique further extended by Buehler et al[12]. Recently, a turn towards real-time Lightfield style rendering is noticeable, with Debevec and colleagues [20] applying projective texture mapping, and Chen providing a hardware based system using a relatively simple modern video card [15]. A recent interesting variation is Matusik et al’s use of opacity hulls to model highly specular or ‘fuzzy’ material, which uses a complex automated object capture rig [57].

**BRDF and BTF simulation**

Most work in computer simulation of objects can be reduced to modelling geometry with material properties, in particular properties relating to interaction with light. The standard approximating formula for a material’s light interaction properties is a bidirectional
CHAPTER 2. LITERATURE REVIEW

reflectance distribution function (BRDF), or a bidirectional texture function (BTF) for spatially varying materials. Perhaps the earliest attempt at a realistic approximation of a BRDF in computer graphics is Phong lighting [71], further refined by Blinn [10] and still very much in use today. Work approximating BTF’s have sometimes involved improving the approximation at surface scale, an early example being texture mapping to change diffuse surface reflectance colour. More recent techniques include bump mapping, as exemplified by Rushmeier et al [78], and horizon mapping [76]. Here, the coarse geometry of the 3D model is refined by applying a 2D map of surface normal variation. Horizon mapping offers the advantage of being able to simulate surface self shadowing effects, at higher computational cost. Liu et al’s work on BTF simulation from a sparse set of images of a real world object continues this theme [53].

The search for better approximations of material BRDFs continued, with Lafortune et al using a generalised Phong lobe model to simulate interesting phenomena such as specular highlights and retro reflection [44]. Interestingly, it is only fairly recently that practical implementations of wave theory have come about, with Stam’s surface diffraction shader system for anisotropic materials, such as the media side of a compact disc [84]. Much recent work has focussed on producing realistic reflections on an object via environment mapping, an example being [14]. Of particular relevance is BRDF measurement of human features and skin. Marschner et al’s work used an automated capture rig with knowledge of 3D surface geometry to provide an image based system for BRDF measurement of skin [56]. Debevec et al use another sophisticated capture device known as a light stage—the data from the light stage allows a wide range of different lighting conditions to be applied to a novel re-rendering of a face [19]. Improvements in hardware capabilities have allowed researchers to provide real time realistic BRDF simulation, first using a simplified BRDF with filtered environment maps [40], then multiple textures to represent relatively simple lighting conditions [58]. Work presented this year from Latta and Kolb, Ramamoorthi and Hanrahan has built on these techniques, extending them to include complex lighting scenarios [45, 75].

Transparent materials

Whilst BRDF and BTF approximations are useful for solid objects, most organic surfaces (in particular skin) have an additional property of translucency, where light can be transmitted around inside the surface causing effects such as colour bleeding. In order to represent this we need an even more sophisticated model, the bidirectional surface scattering reflectance distribution function (BSSRDF), as first introduced by Nicodemus and colleagues[63]. Here, the light emitted from each point is a function of the light absorbed across the rest of the surface. The complexity of BSSRDF modelling has meant that it is only relatively recently that CG approximations have begun to appear. Stam introduced a method based on calculated diffusion within highly scattering media [83]. Dorsey used photon mapping to simulate subsurface transport in weathered stone [22]. Both Jensen and Koenderink introduced more general shaders for transparent 3D objects, again based on diffusion [36, 42]. A major problem with these and previous implementations was their extremely high cost in terms of rendering on a standard Monte Carlo ray tracer—Jensen
and Buehler’s recent work promises to change this, with an emphasis on speed achieved by separating surface irradiance from internal scattering, combining surface irradiance samples in a hierarchical way to simulate light transport [35].

Real time graphics hardware

With increased graphic realism inevitably comes increased computational cost. Our requirement of real-time rendering for our CG face leads us to an examination of expected CG performance of the typical near-future PC. Even simple 3D operations such as perspective projection can require many arithmetic operations, leading to much work on how to optimise and accelerate geometry operations in hardware. Early systems were typically very specialised and expensive, applied in systems such as flight simulators—Clark’s geometry engine is often cited as an example of an early 3D accelerator [16]. The quest for photo-realistic rendering typically still requires a software solution, particularly for advanced BDRF and BSSRDF simulation. Software environments known as shaders are used to compute lighting properties, a well cited example being Pixar’s RenderMan shaders [86].

With improvements in technology came CPU functions aimed at geometry calculations, such as AMD’s 3DNOW! [1] and Intels SSE [34]. Even so, most would accept that it is the rise of the modern graphics co-processor that has provided the most change in desktop graphics power. Modern chip designs such as the ‘emotion engine’ (Kunimatsui et al [43]), provide cheap powerful hardware for home use. Research such as Peercy et al’s on multiple pass rendering on inexpensive graphics hardware now points to the fact that high quality software shaders can now be emulated on chip [69]. The introduction of programmable vertex shaders in hardware, such as the Nvidia Geforce 3 [51] along with sophisticated shader language compiling schemes such as [73] provided even more scope for real-time photorealistic rendering. This year has seen the unveiling of second generation programmable hardware, with manufacturers competing to provide a standardised language for pixel shaders [59, 65]. Finally, Purcell et al suggest the possibility of real time ray-tracing in hardware, based on expected progress over the next few years in the power and flexibility of graphics processing hardware [74]. We note that the accuracy and form of our CG representation of the human face in this thesis must be influenced by the abilities of the hardware on which it will run.
Chapter 3

Implementing a linear object class for 3D face models

3.1 Overview

A linear object class can be described as a set of objects, parametrized by scalar multiplication and addition operations. Each element in the class is constructed from a linear combination of ‘basis objects’, an essential requirement being that any combination of scalar multiplication and additions result in an object of the same form as the input set. As such, the linear object class is analogous to a vector space. Examples of object classes for which a linear class has been shown to be adequate include 2D images of faces and side-views of cars [37]. In this case, scalar multiplication is a pixel-by-pixel brightness operation, and addition pixel-by-pixel alpha blending—a pixel-for-pixel image correspondence is computed in order to ensure that all linear combinations of the input set produce output images in the intended ‘object space’.

For the purposes of this project, five detailed 3D head models were obtained from the Max-Planck-Institut, as used by Vetter and Blanz in [9] and [87]. The models themselves were obtained from a ‘Cyberware’ scanner, a cylindrical scanning device capable of obtaining 3D range and colour data from an object (in this case a subject’s head) placed in the scanner. Captured data is presented in the form of high resolution (512 by 512 pixels) texture and depth bitmaps, with X and Y coordinates in the bitmaps corresponding to rotation and vertical position respectively within the scanning space. This data was then manually edited to remove the subject’s shoulders, neck and hair (enclosed in a shower cap during scanning), as shown in Figure 3.1

The models themselves are all white Caucasian, two each of male and female. A fifth ‘average’ model was artificially created by Vetter and Blanz, representing the averaged head of approximately 200 scanned subjects of similar age and ethnical background. The captured depth maps were then triangulated to create 3D objects, represented as a collection of 3D vertices with a list of faces, each face indexing 3 vertices. Given the nature of the bitmap data provided by the Cyberware scanner every vertex of the triangulated model has corresponding 2D texture map coordinates in the scanned colour map. The
final models were each of the order of 85,000 vertices (170,000 triangles) each.

The intent for this work was to implement a working example of a linear object class for textured 3D face models, using OpenGL to provide a real time rendering environment. We observe that a basic requirement for 3D models to be in a linear class is for the models to be ordered such that same-indexed vertices correspond to the same feature across all the models. Without this condition (i.e. using arbitrary vertex correspondence), we can expect linear combinations of the input model set to produce non-valid output.

3.2 Naïve Attempt

With the initial hope that the provided 3D models were already in correspondence, a simple windowed GUI was developed, providing a 2D display of the 3D face model. A basic parser was written to read in lists of vertex and edge data from the provided data files, as well as texture map coordinates for every vertex—lighting, transform and rendering operations were all performed within OpenGL. Let us assume we are given 3D face models $A$ and $B$, each with $N$ 3D vertices, represented by $N$-vectors of $\mathbb{R}^3$ as $(A_0, \ldots, A_{N-1})$ and $(B_0, \ldots, B_{N-1})$. It is a simple matter to produce a linear object space via scalar multiplication and vector addition, texture mapping being ignored at this stage of the project. On the assumption that the models are already in correspondence their triangulation should be identical, and so we simply render using the first model’s list of faces. We observe a subtle but important requirement for the model: the scaling factors for the two basis face models must sum to 1. This can be considered analogous to maintaining a global scaling factor of one, i.e: maintaining a constant sized output model.

$$\text{OUT}_n = aA_n + bB_n, \ n = 0 \ldots N - 1, \ a + b = 1$$

Unfortunately, it soon became apparent that the provided models, whilst in global correspondence (i.e the face models were roughly the same size, position and facing the same direction), were not ‘vertex for vertex correspondent’. This meant that an attempted
Figure 3.2: Naıve Attempt. Combining non-vertex-for-vertex correspondent models results in a non-valid face model (left).

‘morph’ between faces (by starting at $a = 1, b = 0$ and gradually decreasing / increasing to $a = 0, b = 1$) resulted in an amusing effect whereby sections of the original face would blend into incorrect parts of the target face—for example the nose into the ear. Figure 3.2 shows how, as expected, linear combination without vertex correspondence results in models outside the space of valid human face models.

Closer inspection led to further evidence—the models themselves had different numbers of vertices and hence different ‘edge structures’ (were triangulated differently).

Vetter and Blanz [9, 87] suggest a method for computing 3D surface correspondence in a class of face models. This requires the models to be represented in the original cylindrical format of the Cyberware scanned data, computing a two dimensional flow field in $(y, \theta)$. The flow field is computed using a modified optic flow algorithm, adapted to minimize a vector of $(\text{Radius}(y, \theta), R(y, \theta), G(y, \theta), B(y, \theta))$, rather than $\text{Intensity}(x, y)$.

Aperture problems in areas of low contrast (such as cheeks) are solved by introducing a smoothing constraint.

The robust computation of optic flow between images is a well researched, but still difficult area in computer vision [4, 8]. We observe that, by expanding the algorithm to match Radius (depth) as well as separate R, G and B components the algorithm is now performing a limited kind of 3D shape matching. A short investigation into 2D optic flow techniques led to the conclusion that optic flow in general can be very delicate—this can only be compounded by the fact that we are trying to compute flow between similar but fundamentally different images, rather than the more typical application of flow between frames from a video sequence. The bijection between the ‘rotational coordinates’ of the captured depth map and the 3D points generated from it suggests a computationally
cheaper and conceptually simpler algorithm for computing vertex correspondence. We 
can naturally transform between these two representations without loss of information. 
By converting the 3D models back to their two dimensional cylindrical representation 
(range and colour bitmaps over \((y, \theta)\)), we are left with a simpler problem of computing 
2D pixel for pixel correspondence, as outlined in \([5, 6, 13, 37]\)—correspondence between 
the 2D bitmap representations should provide correspondence between the 3D models. 
We further note that, as the depth and colour maps are captured simultaneously, there 
must be a direct pixel-for-pixel correspondence between depth and colour maps—as each 
vertex in the 3D model has a corresponding 2D texture map coordinate we can use these 
as pixel locations for each vertex on the ‘depth map’. Our method thus becomes one 
of bucketing 3D vertices by their texture coordinates and extracting a sub-sampled 3D 
model. Array bucketing is used to provided a rough grouping on vertices in order to speed 
up sub-sampling.

3.3 2D correspondence

Papers such as \([7, 37, 88]\) discuss the process of computing 2D correspondence in the 
context of a linear object class for 2D images. Success was reported using a modified 
optical flow algorithm to compute a dense flow field from an average image \((A)\) to another 
\((B)\). This flow field simply provides a set of per-pixel vectors mapping pixels on \(A\) to 
pixels on \(B\)—bringing \(B\) into correspondence with \(A\) is then simply a matter of iterating 
through every pixel in \(A\), using the vectors to ‘look up’ the corresponding pixel in \(B\). 
Sub-pixel accuracy is obtaining using bilinear interpolation.

3.3.1 Dense flow correspondence

Various attempts were made to emulate this system, initially a simple pixel-for-pixel naïve 
search. Here we attempt to compute a dense flow field by scanning the image, storing a 
vector \(\text{Warp}(x, y)\) for every pixel \((x, y)\) in the image \(A\). The vectors directly represent 
the flow field, and are computed as below:

\[
\text{Warp}(x, y) = \begin{bmatrix} bx - x \\ by - y \end{bmatrix}
\]

where \((bx, by)\) is the pixel of closest colour in image \(B\) to \((x, y)\), and \(|\text{Warp}(x, y)| < R\) 
specifies a ‘search radius’.

The pixel-for-pixel search algorithm proved ineffective—at the pixel level the differ-
ent face texture maps tended to have quite different colour properties. Whilst our goal 
remains to compute a dense flow field, by analysing the texture maps to find patches (a 
group of pixels) that are particularly unique (and hence not easily confused), we hope a 
variation on the simple searching algorithm above can robustly match these patches. The 
sparse flow field found by matching patches can then be interpolated to give the required 
dense flow field.
3.3.2 Flow computation at interest points

The task of finding ‘interesting’, easily matchable areas in an image can be reduced via the consideration that these areas will be the ones of most sharp contrast, i.e: edges and corners within the image. To this end, a Harris corner detector [32] was implemented, which given an input image computes a ‘cornerness’ metric for each pixel \( I(x,y) \) based on the XY central difference of the image at that pixel; let \( lum(X) \) give the luminance of a RGB pixel \( X \) (defined explicitly in Eqn 4.2), then we define the XY central differences for a particular pixel:

\[
\begin{align*}
\Delta X_{xy} &= lum(I_{(x+1,y)}) - lum(I_{(x-1,y)}) \\
\Delta Y_{xy} &= lum(I_{(x,y+1)}) - lum(I_{(x,y-1)})
\end{align*}
\]

Now, let

\[
\begin{align*}
A_{xy} &= gauss(\Delta X_{xy}^2) \\
B_{xy} &= gauss(\Delta Y_{xy}^2) \\
C_{xy} &= gauss(\Delta X_{xy} \Delta Y_{xy}) \\
\text{cornerness}(I_{(x,y)}) &= |D| - 0.04 \text{Tr}(D)^2
\end{align*}
\]

...where \( gauss(X) \) represents a gaussian smoothing function over the matrix \( X \), and:

\[
D = \begin{bmatrix} A_{xy} & C_{xy} \\ C_{xy} & B_{xy} \end{bmatrix}
\]

Finally, we mark a particular pixel as a corner if it is positive, and a local maximum within the \( 3 \times 3 \) neighbourhood of pixels centered about it. Unfortunately, the nature of the input data set (smoothly textured faces, with rough cut edges) led to the failure of the Harris detector to reliably find internal interest points in the texture maps. Various attempts were made to improve the Harris detector, including introducing quadratic sub-pixel accuracy in determining local maxima, with little improvement. Fua reports similar problems with detecting interest points inside faces due to lack of texture [25]. Figure 3.3 shows an example of typical unsatisfactory output from the Harris detector.

3.3.3 User-specified control points

As an alternative to automatic interest point detection, manually defined ‘control points’ were introduced instead. Here, a standard set of 30 features present in all texture maps were identified and marked at manually to pixel level accuracy. We have a known correspondence between the control points in each texture map, and hence can trivially compute a sparse flow field between any two texture maps. Figure 3.4 demonstrates flow vectors from control points on two of the five faces to the control points on the average face.

3.3.4 Flow field interpolation

In order to warp texture map \( B \) into correspondence with texture map \( A \), the sparse set of flow vectors provided by the control points (\( \text{CPA}_{0...29} \) and \( \text{CPB}_{0...29} \)) must be interpolated into a dense field. Various interpolation methods exist, such as piecewise linear,
Figure 3.3: **Harris detector fails.** Facial texture maps tend to be internally ‘smooth’, and hence interest points (represented by white dots) are only detected around the edges of the face.

Figure 3.4: **Manually defined control points.** 30 manually defined control points at key features are used to provide a sparse flow field. Blue dots give the position of control points, with green lines indicating flow vectors between these faces and the average face.
cubic and nearest neighbour, inverse distance methods and natural neighbour—Amidror provides a useful survey [2]. We note that some of these methods are immediately less suitable for our purposes—piecewise systems like nearest neighbour can suffer discontinuities at ‘edges’, and do not extend beyond the convex hull of the control points simply. Ultimately, ease of understanding and implementation along with acceptable performance leads us to use a radial basis function with a Gaussian kernel.

Let the interpolated flow vector for a given position \([Px \ P y]^{\top}\) be \(\text{Warp}(Px, P y)\), and define \(\Delta X\) and \(\Delta Y\) to represent the sparse flow vectors:

\[
\Delta X = CPB_n.x - CPA_n.x \\
\Delta Y = CPB_n.y - CPA_n.y
\]

We wish \(\text{Warp}(Px, P y)\) to be equal to \([\Delta X_n \ \Delta Y_n]^{\top}\) when \([Px \ P y]^{\top} = CPA_n, (n = 0 \ldots 29)\), or a smooth interpolation of \(\Delta X\) and \(\Delta Y\) otherwise. The flow vector can be calculated by summing the sparse flow vectors multiplied by the negative exponential of the component-sum-squared difference of \([Px \ P y]^{\top}\) and \(CPA_n, \| CPA_n - [Px \ P y]^{\top}\|^2\). We note that a control point \(CPB_n\) has influence on the flow vector \(\text{Warp}(Px, P y)\) which is inverse to \(|CPA_n - [Px \ P y]^{\top}|\).

\[
\text{Warp}(Px, P y) = \sum_{n=0}^{29} \alpha_n \begin{bmatrix} Wx_n \Delta X_n \\ Wy_n \Delta Y_n \end{bmatrix}
\]

where \(\alpha_n = \exp - (\|CPA_n - [Px \ P y]^{\top}\|^2 / K_n^2)\)

Here, \(Wx\) and \(Wy\) are component weighting factors for a given control point position, with \(K\) effectively determining the ‘radius of influence’ of a control feature (generally between 15 - 70 pixels). At this stage, \(Wx, Wy,\) and \(K\) are constant across all texture map warping operations, hand modified to provide visually optimum results across the set of texture maps. Coloured control point markers were added to the texture maps for the purposes of testing, with output images overlaid in a paint package. Alpha blending was used to compare the alignment of coloured markers on the ‘target’ average texture map \(A\) and the output warped images. We observed that, whilst in general the correspondence between the texture maps had been greatly improved, areas with several closely positioned control points often failed to provide the necessary warp. In an attempt to facilitate the hand optimization of the radius and position of the control points, a ‘radius of influence’ diagram diagram was produced showing the interaction between the areas of effect of the control points (Figure 3.5).

The interaction between control point weightings and radii ultimately leads the simple scheme above to fail. The correct distribution of control point weightings can be considered an optimisation problem, with our criteria being the minimization of the total squared error in the system—‘error’ in this case being simply:

\[
\text{Error} = \sum_{n=0}^{29} \| \text{Warp}(CPA_n.x, CPA_n.y) - [\Delta X_n \ \Delta Y_n]^{\top} \|^2
\]

In order to ensure the best possible warping, we describe the intended warp as a large matrix \(D\), solving an additional pre-calculation to compute optimal weights. We note that
whilst $K$ remains global across the warping operations for all texture maps, the weight vectors computed below ($W_x$ and $W_y$) are now specific to which particular texture map (from faces $B \ldots E$) is being warped into correspondence with $A$.

Let

$$D_{ij} = e^{-\left(\frac{(CPB_i.x - CPB_j.x)^2 + (CPB_i.y - CPB_j.y)^2}{K^2_j}\right)} \quad \text{(for } i, j = 0 \ldots 29\text{)}$$

$$W_x = d^{-1} \times \Delta X, \ W_y = d^{-1} \times \Delta Y$$

Then,

$$\text{Warp}(P_x, P_y) = \sum_{n=0}^{29} \alpha_n \begin{bmatrix} W_{xn} \\ W_{yn} \end{bmatrix},$$

where $\alpha_n = \exp\left(-\left(\frac{||[P_x \ P_y] - CPA_n||}{K^2_n}\right)\right)$

With optimal weights computed and used as described above a much more successful warp was achieved. Specifically, the locations of the control points in the ‘target’ average image and their corresponding positions in the warped output images appeared identical. The 2D texture maps were thus successfully warped into correspondence via pixel-by-pixel application of the computed flow field.
CHAPTER 3. IMPLEMENTATION

3.4 Obtaining correspondence for 3D models

With the texture maps now warped so that they are pixel-for-pixel correspondent, the task remains to bring the 3D models into vertex-for-vertex correspondence. As discussed above in §3.2, our technique involves grouping together model vertices by their attached texture map coordinates, then subsampling the model by picking out a subset of each models vertices. Previously, texture map coordinates on different models had no direct relation, however, let us now warp each model’s vertex texture map coordinates equivalently to that model’s texture map warping operation; we can now claim that vertices with similar texture map coordinates correspond to the same facial feature across all models. This will allow us to subsample the models at equivalent features on the face, providing a vertex-for-vertex correspondence. All that remains to complete the process is to define a global triangulation for the subsampled basis faces. Two techniques for providing a triangulation will be defined, one automatic and one offering direct user specification.

3.4.1 Resampling the 3D model

Let us assume input 3D models A...E, where A is the ‘average face’ model. For the purposes of this discussion we make the assumption that the texture maps of the models are in 2D pixel-for-pixel correspondence (which is enforced in 3.3), and define for a given model X:

\[ X_n \text{ input vertices} \]
\[ Xv_{0...(Xn-1)} \in \mathbb{R}^3, \text{ representing } (Xn - 1) \text{ 3D vertex coordinates} \]
\[ Xt_{0...(Xn-1)} \in \mathbb{R}^2, \text{ denoting } (Xn - 1) \text{ 2D texture map coordinates (ranging from 0...1)} \]
\[ Xi_{(x,y)} \in \mathbb{I}^3, x, y = 0...511 \text{ denoting the texture map associated with the model (non integer values of } x \text{ and } y \text{ can be handled using bilinear interpolation)} \]

A preliminary step is to examine the vertex data of each input 3D models to determine the maximal and minimal texture map coordinates in two dimensions. These are stored in \( Xt_{x_{min}}, Xt_{x_{max}}, Xt_{y_{min}} \) and \( Xt_{y_{max}} \). We then construct a 2D array of searchable linked lists, each node being capable of holding vertex and texture coords for a single input vertex. Input vertex information (\( Xv \)) is inserted into the array using the vertices’ texture map coordinates \( Xt \) to determine which element of the array to insert into. Texture map coordinates must be scaled using \( Xt_{x} \) and \( Xt_{y} \) so as to efficiently use the whole space available in the array—for the purposes of this discussion we assume a 2D array of linked lists of user defined size \( Q \times Q \).

3.4.2 First attempt: Resampling at regular intervals

During the process of inserting vertices into the array, minimal and maximal texture map \( x \) coordinates are maintained for each row of the array, labelled \( X_{rxmin0...Q-1} \) and
Figure 3.6: **Regular resampling.** Model is resampled using extraction points on a regular grid scaled to the width of the face.

$X_{rx\text{max}0...Q-1}$. With all the vertices in the input model now held in the array, we extract $Q$ vertices from each of the $Q$ rows of the array. Given minimal and maximal texture map coords we can generate a set of *extraction points* across the array. The extraction points consist of $Q$ rows of $Q$ points at evenly spaced intervals across the horizontal segments of the model’s face, providing a regular grid style resampling over the face as shown in Figure 3.6.

The array is then scanned to find the $Q^2$ vertices with closest corresponding member within $X_t$ to the extraction points, minimizing sum-squared-difference. This is achieved by first calculating which 2D array entry contains the extraction point, then searching through the $3 \times 3$ grid of lists centred about this point within the array. It simply remains to triangulate the vertices into faces for rendering. This is trivial, and of course all 5 output models have the same regular triangulation, as demonstrated in Figure 3.7.

A first order solution to the correspondence problem, we find that, in the general case this method results in five output models with the same number of vertices, each with $Q$ horizontal segments of $Q$ vertices—intrinsically we have vertex-for-vertex correspondence.
CHAPTER 3. IMPLEMENTATION

3.4.3 Second attempt: User defined resampling

It follows naturally from the construction of the human head that the input 3D face models are of varying ‘horizontal segment’ size—this can be interpreted in the context of the 3D models as the diameter of the cross section of the face is non-constant with respect to vertical position. As such, the minimal and maximal texture map coords used in the array vary from row to row—this is particularly evident at the tip of the head near the hairline, and at the neck. With the system described above, each row in the array is allocated the same number of vertices ($Q$)—this is a sub-optimal arrangement as, logically, larger segments of the head should have more vertices to better match the larger sections. We also observe that the number of vertices required to maintain a specified level of accuracy is lower in smooth, largely continuous areas of the face, such as the forehead and cheeks. Whilst sophisticated vertex culling algorithms do exist\cite{52,27}, these are more focused at eliminating vertices on a solid model, rather than picking a suitable general triangulation for a linear object space.

Experience from real time environments, such as the gaming industry, indicates that vertex placement for critical applications in which ‘every vertex counts’ is still best performed manually; we choose to implement a system for allowing manual specification of the location of the extraction points for the linear model. A straightforward system captures the location of extraction points within the 2D array of lists of vertex information from mouse input. In order to achieve greater than pixel accuracy on highly detailed (non-smooth) sections of the model we increase the resolution of the system to $1/16$ pixel by enlarging the size of the texture map by a factor of 4 in both dimensions. Figure 3.8 shows a typical arrangement of extraction points, bright green pixels indicating intended vertex locations. Obvious advantages of this system over §3.4.2 include intuitive, easy editing with standard paint software, as well as completely user specifiable output model resolution.

The final stage in our revised resampling system is to provide a suitable triangulation...
for the specified extraction locations. At the very least, this triangulation must have the properties of completeness (every point is included in a triangle) and non-overlapping (no points are ever found within a triangle). A standard technique for the production of ‘sensible’ triangulations for 2D point clouds is the Delaunay triangulation, the implementation of which we borrow from O’Rourke’s publically available C code [66]. We observe that the relatively efficient implementation ($O(n^2)$) of Delaunay is actually a fairly advanced application of computational geometry techniques, using the $XY$ plane projection of the 3D convex hull of the points $[X_n, Y_n, X_n^2 + Y_n^2]^T$ (where $X_n,$ and $Y_n$ are the locations of the extraction points).

An easily corrected problem that affected both model resampling techniques was due to the occasionally non-convex nature of the 2D texture maps, resulting in the creation of triangles where no valid texture map or 3D data was present. A simple workaround for use with the user specified resampling was to introduce an additional pixel colour on the extraction map indicating ‘outside face’ areas. The output list of faces from the Delaunay algorithm is simply searched to remove any faces that contain edges overlapping this colour. Figure 3.8 shows the output from the Delaunay routine, modified to remove ‘rogue’ triangles.

### 3.4.4 The model viewer revisited

In order to examine the effectiveness of the method above the viewer GUI described in 3.2 was expanded and modified to support five way face morphing, representing 5 di-
Figure 3.9: Triangulation Problem. Triangulation of the extraction points falls outside of the facial area in the 2D texture map, which can be fixed by removing triangles with edges that overlap specially coloured marker pixels.

Execution of the system described above immediately led to the discovery of several issues of varying significance. Firstly, experiments with the value $Q$ led to the consideration of the ‘missing vertex problem’; a higher value of $Q$ leads to a larger vertex array, and hence generally shorter linked lists, speeding up the resampling process—unfortunately problems can occur at very large $Q$. During the extraction phase, we assume that the array will always be able to provide a vertex for each of the calculated extraction points (i.e. there must be at least one valid node in one of the lists in the $3 \times 3$ grid of array elements surrounding the extraction point). We see however, as $Q$ increases the array will become increasingly sparse—with a suitably high value if $Q$ it is possible that the method will be...
Figure 3.10: **Lighting normals.** Lighting normals do not interpolate via linear operation like 3D vertex data, as shown via 2D analogy to 3D geometry.

unable to find a vertex with a 3 x 3 neighbourhood. An obvious fix was simply to expand the search neighbourhood size. A further fact that became apparent was the necessity for recalculating surface normals for lighting during the intermediate stages of the morph—lighting normals do not interpolate simply like vertices, as a 2D analogy demonstrates in Figure 3.10:

We conclude the chapter with a small demonstration of the successful implementation of a 5 dimensional linear object space for 3D face models. Figure 3.11 shows a ‘morph’ between two basis faces, achieved by taking snapshots of the 3D model as we walk through model parameter space. It seems clear from the images that the intermediate 3D models all appear to be valid human faces, evidence that the linear object space does indeed meet our intended criteria.
Figure 3.11: The linear object model. A morph achieved by capturing 2D snapshots of the 3D model generated by a walk through model parameter space.
Chapter 4

Performing 3D from 2D using a linear object model for human faces

4.1 Overview

With a working implementation of a linear object space for human face models, we can now begin investigating several interesting applications. For clarity, let us formalise the linear object space. After the necessary computations described in §3 the model consists of a number of basis face models (currently 5), each model being a list of \( F \) 3D vertices and 2D texture map coordinates in index correspondence and a \( G \) by \( G \) 24 bit colour texture map. There is also a global triangulation for the model, a list of \( H \) vertex index triples defining a surface mesh.

\[
\text{MODEL} = \{ \text{E}_0 \ldots \text{E}_4 \} \text{ and } \{ \text{Tri}_0 \ldots \text{Tri}_{H-1} \}
\]

where \( \text{E}_X = [X_0 \ldots X_{F-1}, U_0 \ldots U_{F-1}, T] \), \( X_x \in \mathbb{R}^3, U_x \in \mathbb{R}^2, \text{Tri}_X \in \mathbb{I}^3 \),
\( T \) is an RGB image of size \( G \times G \), \( T(x,y) \in \mathbb{I}^3 \)

Now, we have a 5 dimensional ‘space’ of valid human face models, parameterized by the ‘model parameters’ \( \alpha_0 \ldots \alpha_4 \). As the 4 basis faces are in feature-for-feature correspondence, we can construct the particular vertices and texture map of a general face within the space via:

\[
\text{FACE}(\text{a}) = \sum_{q=0}^{4} \alpha_q \text{E}_q
\]
where \( \text{a} = [\alpha_0 \ldots \alpha_4]^T \)
\( \sum_{q=0}^{4} \alpha_q = 1 \)

Finally, although our model is in 3D, the majority of applications for it (such as viewing the model) require the model to be projected onto a 2D viewing plane (for example for real-time display on a computer monitor). This is accomplished via standard perspective projection of the vertex positions onto a viewing plane—relative position and orientation of the camera and model defines the viewed ‘pose’ of the model[24]. These 2D vertices are then drawn as texture mapped triangles, a similar per-triangle projection being applied.
to texture map information. Lighting conditions are simulated using a single Lambertian light source, surface normals for the triangles being calculated as necessary.

\[
\text{PROJ}_q(a) = K[R | t] \begin{bmatrix} X_q \text{ from FACE}(a) \end{bmatrix}
\]

(4.1)

where \( R \) is a \( 3 \times 3 \) rotation matrix,
\( t \) is the translation of the camera,
\( K \) is the internal calibration parameters of the camera:

\[
K = \begin{bmatrix} f & s & u_0 \\ 0 & af & v_0 \\ 0 & 0 & 1 \end{bmatrix}
\]

\( f \) is focal length;
\( (u_0, v_0) \) is principal point;
\( a \) is aspect ratio; \( s \) is skew

\( \text{PROJ}_q(a) \) is the projection of the model vertex on the viewing plane, with model parameters \( a \)

### 4.1.1 Obtaining 3D from 2D

A common objective in computer vision is the application of obtaining good approximating 3D models from the real world, with a wide variety of techniques having been developed. These can essentially be grouped into ‘3D scanning’, such as [60] and [80], where 3D information about the shape of the subject is captured directly using a range finding technique (typically using a striping technique such as [79]), or ‘3D from 2D’ where computer vision techniques are applied to a set of 2D captured images of the subject to produce a 3D model, examples being [25] and [19]. Both methodologies have relative merits, but we can observe that as a general rule ‘3D from 2D’ requires less extrinsic calibration and less sophisticated hardware setup, but full ‘3D scanning’ can produce more accurate results (at least, it is the output 3D models from high end systems that fall into this group that are most often used as a yardstick by which to judge the accuracy of a new capture system).

The parameterized nature of the linear model suggests a new ‘3D from 2D’ technique, as developed by Vetter and Blanz in [9, 87]. By applying the projection transformation above we can produce a 2D image of our parameterized 3D model—our objective will be to ‘tune’ our linear model to best approximate a particular person, as specified by one or more 2D images. This can be refined to a problem statement as follows:

Find the best pose \( (R, t) \) and model parameters \( (\alpha_0, A) \) such that the 2D camera image (the generated image) of the model looks most like the image of the subject (the target image).

Let us formalise our definition of an ‘image’, labelled \( J \). \( J \) is essentially a large matrix (array) of integer triples representing RGB pixel values, its width and height in pixels given by the functions \( \text{width}(J) \) and \( \text{height}(J) \) respectively. A particular pixel in \( J \) is referred to as \( J(x, y) \), with:

\[
J(0...\text{width}(J)-1,0...\text{height}(J)-1) = J
\]
Later it will be necessary to convert RGB pixels to luminance values and to that end we define a simple function for this:

\[
lum(Pix) = Pix.r \times 0.30 + Pix.g \times 0.59 + Pix.b \times 0.11
\]

where \( Pix.r \) is the red element of Pix, etc (4.2)

Of course, general 2D images of human faces can come from many different environments and capture devices, with many factors outside the current scope of the model (such as lighting, and low level surface detail) contributing to the overall appearance of the face. Even so, with six pose and five model parameters there is already an 11 dimensional space to be optimized, leading us to discount an exhaustive search immediately. Instead, our technique will be one of hypothesis testing—the model is rendered in 2D with a parameter set \( S^0 = [R_0, t_0, \{\alpha_{0,d}\}]^T \) which represents a ‘first guess’ at the optimal set. By performing some comparison of the generated output with the target image(s) we hope to be able to modify \( S^0 \) in a meaningful way to produce \( S^1 \), the process repeating iteratively until some termination criteria is met.

### 4.2 Feature based matching

The problem of finding optimal rotation, translation, scaling etc for a 3D object to match to a 2D image is not new. Techniques can effectively divided into feature based and pixel based methods. Feature based methods rely on matching certain key 2D locations \( T_i \) with particular points (often specific vertices) on the 3D model \( G_i \). By performing a camera projection of the object in a particular orientation (pose) the projected 2D location of \( G_i \) can be found, denoted \( g_i \). A simple error metric is just the summed distance of the elements of \( g_i \) and \( T_i \):

\[
Error = \sum_{i=0}^{N-1} \|g_i - T_i\|
\]

where \( N \) is the number of features to be matched.

Feature based matching of object pose can be found in closed form for 3 to 5 features [18], and via approximate iterative methods for larger numbers of feature matches—this of course depends on a known bijection between the 2D position of features in the target image and the 3D object. We note that a target 2D image does not intrinsically specify these locations. Much work has been carried out on face recognition, perhaps suggesting that automated detection of facial features in 2D images is possible, for example ‘eye detection’ [3]. Even so, it remains a difficult problem even for humans to consistently identify facial features across many different poses and lighting conditions—as an example consider the effect of lighting on perceived cheek bone position.

Algorithms such as POSIT [21] (described below) solve the pose estimation problem by clever manipulation of the equations defining the projection of model geometry onto a 2D camera plane, then iteratively improving estimates for pose and the homogeneous components of the features on the target image.
CHAPTER 4. 2D MATCHING

4.2.1 Overview of the POSIT algorithm

For the purposes of discussion, we assume that we wish to compute the optimal pose \([R \mid t]\) between \(K\) 3D vertices and \(K\) 2D image locations. Let the 3D vertex coordinates be expressed as \([X_k \ Y_k \ Z_k \ 1]^\top\) and the 2D image locations homogeneously as \([w_k x_k \ w_k y_k \ w_k]^\top\), \(k = 1 \ldots K\). We assume that the correspondence problem has been solved (the bijection between 2D vertices and 3D image points is known), ordering the two sets such that \([X_k \ Y_k \ Z_k \ 1]^\top \leftrightarrow [w_k x_k \ w_k y_k \ w_k]^\top\). Our task is thus to find \([R \mid t]\) such that the distance in pixels between the 2D image locations and the transformed, perspective projected 3D vertices is minimized.

By standard 3D projection equations, assuming square pixels and that the origin of the image plane is at the principal point:

\[
\begin{bmatrix}
  x_k \\
  y_k \\
  1
\end{bmatrix} =
\begin{bmatrix}
  f & 0 & 0 \\
  0 & f & 0 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  R_1^\top \\
  R_2^\top \\
  R_3^\top
\end{bmatrix}
\begin{bmatrix}
  t_x \\
  t_y \\
  t_z
\end{bmatrix}
\begin{bmatrix}
  X_k \\
  Y_k \\
  Z_k \\
  1
\end{bmatrix}
\]

(4.4)

where \(R_1^\top, R_2^\top, R_3^\top\) are three row vectors of the pose rotation matrix \(R\) or

\[
\begin{bmatrix}
  x_k \\
  y_k \\
  1
\end{bmatrix} =
\begin{bmatrix}
  f R_1^\top & ft_x \\
  f R_2^\top & ft_y \\
  f R_3^\top & ft_z
\end{bmatrix}
\begin{bmatrix}
  X_k \\
  Y_k \\
  Z_k \\
  1
\end{bmatrix}
\]

(4.5)

As the image locations are defined homogeneously (i.e: in terms of the multiplicative constant \(w_k\)) we can apply a multiplication of \(1/t_z\) to the perspective projection matrix without affecting the equality of Eqn 4.5. We also introduce a scaling factor \(s = f/t_z\) to simplify the representation of the equation.

\[
\begin{bmatrix}
  x_k \\
  y_k \\
  1
\end{bmatrix} =
\begin{bmatrix}
  s R_1^\top & st_x \\
  s R_2^\top & st_y \\
  s R_3^\top & st_z
\end{bmatrix}
\begin{bmatrix}
  X_k \\
  Y_k \\
  Z_k \\
  1
\end{bmatrix}
\]

(4.6)

\[
w_k = R_3 [X_k \ Y_k \ Z_k]^\top / t_z + 1
\]

(4.7)

If we are given knowledge of \(w_1 \ldots K\) we can construct a linear system of \(K\) equations via a rearrangement of Eqn 4.6, which can be solved to yield \(s R_1, s R_2, st_x, \) and \(st_y\). This is of course dependent on the rank of the matrix containing 3D vertices being at least 4, and being (pseudo)invertible. The geometrical interpretation of this requirement is that
the \( K \) 3D model points being matched must contain at least 4 noncoplanar vertices.

\[
\begin{bmatrix}
X_1 & Y_1 & Z_n & 1 \\
X_n & Y_n & Z_n & 1 \\
\vdots & \vdots & \vdots & \vdots \\
X_K & Y_K & Z_K & 1
\end{bmatrix}
= \begin{bmatrix}
w_{1x1} & w_{1y1} \\
w_{2x2} & w_{2y2} \\
\vdots & \vdots \\
w_{KxK} & w_{KyK}
\end{bmatrix} \begin{bmatrix}
sR_1 & sR_2 \\
st_x & st_y
\end{bmatrix}
\tag{4.8}
\]

The knowledge that \( R_1^\top \) and \( R_2^\top \) are two rows of a rotation matrix allows us to extract out the required values:

\[
s = \sqrt{|sR_1| |sR_2|} \quad t_x = (st_x)/s \\
R_1 = (sR_1)/s \quad t_y = (st_y)/s \\
R_2 = (sR_2)/s \quad t_z = f/s \tag{4.9}
\]

As stated above, to solve Eqn 4.8 above we require knowledge of \( w_{1...K} \). This is accomplished by iteratively improving our estimates of \( w_{1...K} \), \( R \), and \( t \), using \( w_{1...K} = 1 \) as a first estimate. We note that with this starting condition Eqn 4.6 corresponds to scaled orthographic projection, an approximation of perspective projection [18]. With a pose result for the first iteration we can then improve our initial estimate for \( w_{1...K} \) using Eqn 4.7, repeating this process until convergence is achieved.

### 4.2.2 Application of POSIT to linear object spaces

With some modification POSIT can be used to determine model parameters at the same time as pose parameters in a similar iterative way. Let us assume we wish to optimize model coefficients and pose for an \( N \) dimensional linear object space with \( K \times N \) 3D vertices (\( K \) feature vertices in each of the \( N \) basis models) to \( K \) 2D image points; we now have \([X_{(n,k)} \ Y_{(n,k)} \ Z_{(n,k)}] \top \) \( (n = 1 \ldots N \) and \( k = 1 \ldots K) \), with \( \alpha_{1...N} \) model coefficients to be optimized, such that:

\[
\begin{bmatrix}
w_{kxk} \\
w_{kyk} \\
w_k
\end{bmatrix}
= \begin{bmatrix}
f & 0 & 0 \\
0 & f & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
R_1^\top & t_x \\
R_2^\top & t_y \\
R_3^\top & t_z
\end{bmatrix}
\begin{bmatrix}
X_{(1,k)} & X_{(2,k)} & \ldots & X_{(N,k)} \\
Y_{(1,k)} & Y_{(2,k)} & \ldots & Y_{(N,k)} \\
Z_{(1,k)} & Z_{(2,k)} & \ldots & Z_{(N,k)}
\end{bmatrix}
\begin{bmatrix}
\alpha_1 \\
\alpha_2 \\
\vdots \\
\alpha_N
\end{bmatrix} \tag{4.10}
\]

Now, let

\[
B_k = \begin{bmatrix}
X_{(1,k)} & Y_{(1,k)} & Z_{(1,k)} & 1 \\
X_{(2,k)} & Y_{(2,k)} & Z_{(2,k)} & 1 \\
\vdots & \vdots & \vdots & \vdots \\
X_{(N,k)} & Y_{(N,k)} & Z_{(N,k)} & 1
\end{bmatrix}, \text{ and } a = \begin{bmatrix}
\alpha_1 \\
\alpha_2 \\
\vdots \\
\alpha_N
\end{bmatrix}
\]
Then by analogy to Eqns 4.6, 4.7 and 4.8:

\[
\begin{bmatrix}
a^\top B_1 \\
a^\top B_2 \\
\vdots \\
a^\top B_K \\
\end{bmatrix}
\begin{bmatrix}
sR_1 \\
st_x \\
sR_2 \\
st_y \\
\vdots \\
\end{bmatrix}
= 
\begin{bmatrix}
w_1 x_1 & w_1 y_1 \\
w_2 x_2 & w_2 y_2 \\
\vdots & \vdots \\
w_K x_K & w_K y_K \\
\end{bmatrix}
\]

(4.11)

Separating the equation into components:

\[
a^\top B_k \begin{bmatrix} sR_1 \\ st_x \end{bmatrix} = w_k x_k, \text{ and } a^\top B_k \begin{bmatrix} sR_2 \\ st_y \end{bmatrix} = w_k y_k
\]

where \(k = 1 \ldots K\)

(4.12)

or, equivalently by \((ABC)^\top = A^\top B^\top C^\top\):

\[
\begin{bmatrix} sR_1 \\ st_x \end{bmatrix} B_k^\top a = w_k x_k, \text{ and } \begin{bmatrix} sR_2 \\ st_y \end{bmatrix} B_k^\top a = w_k y_k
\]

(4.13)

Each iteration is now a three step process. Given initial estimates for \(w_1 \ldots K\) and \(\alpha_1 \ldots N\) we can use Eqn 4.11 to compute an estimate for object pose \((sR_1, sR_2, st_x, \text{ and } st_y)\) via pseudo-inverse of the matrix \([a^\top B_1 \ a^\top B_2 \ldots \ a^\top B_K]^\top\). Then a system of linear equations based on Eqn 4.13 is constructed to produce the improved estimate for model coefficients, \(a'\):

\[
a' = \begin{bmatrix}
sR_1^\top \\
sR_2^\top \\
\vdots \\
sR_K^\top \\
\end{bmatrix} B_k^\top \begin{bmatrix} a^\top B_1 \\
a^\top B_2 \\
a^\top B_3 \\
a^\top B_K \\
\end{bmatrix}^* \begin{bmatrix} w_1 x_1 \\
w_1 y_1 \\
w_2 x_2 \\
w_2 y_2 \\
\vdots \\
w_K x_K \\
w_K y_K \\
\end{bmatrix}
\]

(4.14)

where \(A^*\) is the psuedoinverse of matrix \(A\)

As discussed in §3, in order to maintain a constant model size \(\alpha_1 \ldots N\) must sum to one—this property can be maintained by simply scaling \(a'\) by \(1/\sum_{n=1}^N \alpha'_n\). The final stage in the iteration is to find \(s, R_3, \text{ and } T_z\) and improve on our initial guess for \(w_1 \ldots K\). This is done in a similar manner to standard POSIT, constructing a temporary set of \([X_k \ Y_k \ Z_k \ 1]\) for use with Eqns 4.7 and 4.9 as follows:

\[
\begin{bmatrix} X_k \\
Y_k \\
Z_k \\
1 \\
\end{bmatrix} = a'^\top B_k
\]

(4.15)

### 4.2.3 Finding initial estimates for \(a\)

The modified POSIT iteration now requires ‘initial guesses’ for the model coefficients \((\alpha_1 \ldots N)\), as well as \(w_1 \ldots K\). Again assuming scaled orthographic projection for the first iteration \((w_1 \ldots K = 1)\), we can use a rewriting of Eqn 4.12 to obtain initial conditions
for \(a_{1..N}\) as well as model pose (note how \(B_k\) is flattened, and brought to the left of the equation by pre-multiplying the two other left hand terms):

\[
\begin{bmatrix}
\alpha_1 sR_1 \\
\alpha_1 s_t x \\
\vdots
\alpha_{N/2} sR_1 \\
\alpha_{N/2} s_t x
\end{bmatrix}
\begin{bmatrix}
\alpha_1 sR_1 \\
\alpha_1 s_t x \\
\vdots
\alpha_{N/2} sR_1 \\
\alpha_{N/2} s_t x
\end{bmatrix}
\begin{bmatrix}
X(1,k) \\
Y(1,k) \\
\vdots \\
X(N,k) \\
Y(N,k)
\end{bmatrix}
\begin{bmatrix}
Z(1,k) \\
1 \\
\vdots \\
Z(N,k) \\
1
\end{bmatrix}
= w_k x_k
\] (4.16)

A linear system can then be constructed by ‘stacking’ \(4N\) of Eqn 4.16, combining multiple flattened \(B_k\)’s into one large matrix, and many \(w_k x_k\) into a vector. We can then inverse the large matrix of \(B_k\)’s, pre-multiplying both sides of the equation to yield \([\alpha_0 \ldots \alpha_N] sR_1\) and \([\alpha_0 \ldots \alpha_N] s_t x\). However, let us assume for the purposes of discussion that \(N = 2I, I \in \mathbb{Z}\). Then we can ‘split’ a between Eqn 4.16 and its analogous equation with \(w_k y_k\), giving two lower dimensional equations to solve of the form:

\[
\begin{bmatrix}
\alpha_1 sR_1 \\
\alpha_1 s_t x \\
\vdots \\
\alpha_{N/2} sR_1 \\
\alpha_{N/2} s_t x
\end{bmatrix}
\begin{bmatrix}
\alpha_1 sR_1 \\
\alpha_1 s_t x \\
\vdots \\
\alpha_{N/2} sR_1 \\
\alpha_{N/2} s_t x
\end{bmatrix}
\begin{bmatrix}
X(1,1) \\
Y(1,1) \\
\vdots \\
X(N/2,1) \\
Y(N/2,1)
\end{bmatrix}
\begin{bmatrix}
Z(1,1) \\
1 \\
\vdots \\
Z(N/2,1) \\
1
\end{bmatrix}
= w_1 x_1
\] (4.17)

Explicitly, the partner equation will involve \([X(n,k) Y(n,k) Z(n,k) 1]\), \(sR_2\), \(s_t y\), \(y_{1..2N}\) and \(\alpha_{N/2+1..N}\), \(n = N/2 + 1 \ldots N\). Finally, a rewriting of our solution vectors in matrix form allows us to decompose the solution vector into two independent vectors of smaller size, indicating that SVD decomposition can be used to extract out starting conditions for \(a_{1..N}\). Consider below the example for Eqn 4.17:

\[
\begin{bmatrix}
\alpha_1 sR_1 \\
\alpha_1 s_t x \\
\vdots \\
\alpha_{N/2} sR_1 \\
\alpha_{N/2} s_t x
\end{bmatrix}
\begin{bmatrix}
\alpha_1 sR_1 \\
\alpha_1 s_t x \\
\vdots \\
\alpha_{N/2} sR_1 \\
\alpha_{N/2} s_t x
\end{bmatrix}
\begin{bmatrix}
\alpha_1 \\
\alpha_1 \\
\vdots \\
\alpha_{N/2} \\
\alpha_{N/2}
\end{bmatrix}
\begin{bmatrix}
sR_1 \top \\
s_t x
\end{bmatrix}
\] (4.18)

With larger dimensional spaces it is clear this solution will quickly become extremely computationally expensive, but perhaps more importantly we see from Eqn 4.17 that we require \(k \geq 4\) and \(k \geq 2N\) in order to have sufficient equations to determine a solution. Of course, the ‘space’ of human faces that can be represented by the model is determined by the dimensionality of the linear space—the greater the size of the input basis set of models the greater number of combinations possible. Vetter and Blanz suggest that for effective
general matching of faces a basis set of 100 faces is sufficient [9]—with a modified POSIT algorithm in use to perform the matching we would need an enormous 200 uniquely identifiable features on the human face to match with, a task almost certainly impossible either via human or machine vision techniques. Even so, this algorithm is expected to be useful in providing initial estimates for parameters.

4.3 Pixel based matching

With the non-rigid nature of our 3D model complicating feature based image matching methods, we turn instead to pixel based methods. A more ‘brute force’ approach, here the closeness of the generated and target images is found by a comparison on a pixel-by-pixel basis. The most straightforward metric is just luminance difference, converting full colour images to luminance maps as necessary:

$$\text{Metric} = - \sum_{x=0}^{\text{width}(T)-1} \sum_{y=0}^{\text{height}(T)-1} \left| \text{lum}(G(x,y)) - \text{lum}(T(x,y)) \right|$$

(4.19)

where $G,T$ are the generated / target images.

Testing indicates Eqn 4.19 is a poor metric for image correspondence. Figures 4.1 and 4.2 show 3D plots of Eqn 4.19 applied on a rendered target face image and a set of generated images—each plot varies two parameters across 21 different positions, giving 121 generated images per set. The generated sets used in the first 3 plots vary 2 of the 6 pose coefficients at a time, using a target image of the model in the central position. We observe that an ideal metric would be smoothly increasing as we move towards the globally correct pose and model coefficients (the ideal pose in the center of the graph with no variation from the target). Our overall objective being to perform image matching between synthetic and real world images, it seems prudent to examine the effect of noise on the target image on the metric; a further concern is the effect of lighting variation on image correspondence. The following 6 plots apply the same variation in pose parameters, but add artificial noise of 20 and 50 pixels to the target image. The next shows the metric’s behaviour as the position of lighting relative to the face model is translated over a range of values in X and Y. Finally, in the last 2 plots we repeat the study of the effect of rotation in X and Y, but use target images of the model rendered using offset lighting positions.

An immediate observation is that noise, even as large as 50 pixels, seems to have little effect on Eqn 4.19—this is evident in the similarities between the first 9 plots. Whilst pixel difference appears to provide a good metric for translational variance (plots 1, 4, and 7), it is less effective for rotation in X and Y (plots 2, 5, and 8). Furthermore, the inverted valley-like shape of plots 3,6, and 9 indicates translation in Z seems to have little effect at all on the metric. The worst performance occurs in plots 11 and 12, in which we test the ability of Eqn 4.19 to match the model to target images with offset lighting positions. It seems likely that only a fairly rough estimation of lighting conditions will be possible when matching to real world images, and so this is a particularly unsatisfactory result. We therefore begin consideration of alternative image correspondence metrics.
Figure 4.1: **Testing pixel difference as a correspondence metric** 1 Plots 1 - 3, top row from left to right: target image of model in ‘0-position’ compared to model over X and Y translation, X and Y rotation, Z translation and rotation. Plots 4 - 6, bottom row from left to right: as for plots 1 - 3, except using 20 pixels of random noise on target image. Note that the peak in each graph is in the correct (central) position.
Figure 4.2: **Testing pixel difference as a correspondence metric** 2 Plots 7 - 9, top row from left to right: as for plots 1 - 3, except using 50 pixels of random noise on target image. Plot 10, bottom left: effect of translation of light position in X-Y plane. Plots 11 - 12, model rotation in X and Y axis, with target model image using light offset of 1,1 and -2,-2 respectively. We observe that non standard target lighting results in incorrect location of the metric’s peak value.
4.3.1 Mutual information

An alternative metric to the pixel difference method is ‘mutual information’, a technique already successfully applied to the calculation of optimal alignment of images via 2D affine transformation ([28] and [89]). Essentially, the mutual information between two images is maximised when they are identical, and decreases as the images become more dissimilar. The mutual information between two images is calculated in terms of individual and joint image entropy. Let us first define the measure of entropy within an image, which is in turn based on general entropy theory [67] for discrete random variables:

Let \(X\) be a discrete random variable, with events \(X_0, \ldots, X_n\), and probability of a particular event \(X_i\) equal to \(P(X_i)\). It follows naturally that the probability of all events sums to one:

\[
\sum_{i=0}^{n} P(X_i) = 1
\]

Then, let the entropy of \(X\) be defined as \(H(X)\):

\[
H(X) = -\sum_{i=0}^{n} \log(P(X_i)) P(X_i)
\] (4.20)

Entropy can be considered a measure of ‘randomness’, essentially how uniformly behaved a random variable is. Entropy is maximized by a random variable having many similarly probable outcomes, or minimized by an ‘unfair’ distribution of likelihood. Figure 4.3 gives two contrasting examples; a fair 20 sided dice (high entropy) and a biased coin (low entropy).

We now consider the interpretation of an image \(J\), as described above in §4.1 as a discrete random variable. Let \(L_J\) be the set of all integer luminance values \(J(x, y) \in J\). Let \(w = \text{width}(J)\) and \(h = \text{height}(J)\), then the probability of a particular pixel having luminance \(l \in L_J\) is simply:

\[
P(l) = \frac{1}{w \times h} \sum_{x=0}^{(w-1)} \sum_{y=0}^{(h-1)} 1, \text{ if } J(x, y) = l
\]

\(0, \text{ otherwise}\)

Analogous to entropy of a discrete random variable, image entropy \(H(J)\) can be defined:

\[
H(J) = -\sum_{l \in L_J} \log(P(l)) P(l)
\] (4.21)

Finally, we define joint image entropy in terms of \(J\) and \(K\), where \(K\) is another image of size \(w\) by \(h\), with \(L_K\) being the set of all luminance values \(K(x, y) \in K\). Then

\[
H(J, K) = \sum_{l_J \in L_J} \sum_{l_K \in L_K} \log(P(l_J, l_K)) P(l_J, l_K)
\]

\[
P(l_J, l_K) = \frac{1}{w \times h} \sum_{X=0}^{(w-1)} \sum_{Y=0}^{(h-1)} 1, \text{ if } J(x, y) = l_J \text{ and } K(x, y) = l_K
\]

\(0, \text{ otherwise}\) 

(4.22)
The mutual information of images \( J \) and \( K \) is defined:
\[
I(J, K) = H(J) + H(K) - H(J, K)
\] (4.23)

It is perhaps difficult to interpret this equation as to the nature of the metric provided, however probability and entropy theory [67] suggest an alternate form of the equation that is perhaps more intuitive:
\[
I(J, K) = H(J) - H(J|K)
\]

This suggests \( I(J, K) \) is the reduction in entropy of \( K \) given \( J \) [67], i.e: the reduction in uncertainty about \( K \) due to the knowledge of \( J \). Furthermore by simple expansion:
\[
H(J, K) = H(K, J) \text{ hence } I(J, K) = I(K, J)
\]

This perhaps explains the motivation behind the choice of term ‘mutual information’: \( J \) always provides the same amount of information about \( K \) as \( K \) gives about \( J \).

### 4.3.2 Implementing mutual information

With the theory given above we can now begin to consider implementation specific issues. As discussed above in section 4.1, our global technique will be one of iterative hypothesis and testing—if mutual information is to be used as a metric for 2D image comparison, we can expect to be computing the mutual information between target(\( T \)) and a generated (\( G_0 \ldots n \)) image at every iteration. Because of this, we must aim to optimize the
CHAPTER 4. 2D MATCHING

calculation of \( I(T, G_x) \). Eqns 4.21 and 4.22 specifying image entropy initially appear to require \( O(|L|width(X)height(X)) \) time, searching through the image to find pixels of a particular colour for every existing luminance value. A much more efficient alternative requiring only one pass through the image \( O(LX + width(X)height(X)) \) is to build a histogram counting the frequency of occurrence of each possible luminance value in the image. These values can then be divided by \( width(X)height(X) \) to give \( l_z \in L_X \) for a given luminance value \( l_z \) in image \( X \). An analogous technique can be used for computing the joint entropy \( H(T, G_x) \), via a 2D frequency histogram. Given that the 3D to 2D matching process will use static target images, it is a simple observation that \( H(T) \) is constant throughout the matching process. Eqn 4.24 gives pseudocode for calculation of mutual information between two images, \( G \) and \( T \).

\[
\text{ASSUME } width(G) = width(T), \text{ height}(G) = height(T)
\]

\[
\begin{align*}
&\text{LET } \text{LUM}(I, X, Y) = \text{integer part of } \text{lum}(I(x, y)) \\
&\text{THIST}[0 \ldots 255], \text{GHIST}[0 \ldots 255], \text{JHIST}[0 \ldots 255][0 \ldots 255] = 0 \\
&HT, HG, HJ = 0, \text{SIZE} = width(T) \times height(T) \\
&\text{FOR } X = 0 \ldots width(T) \\
&\qquad \text{FOR } Y = 0 \ldots height(T) \\
&\qquad \quad \text{THIST}[\text{LUM}(T, X, Y)] + = 1 \\
&\qquad \quad \text{GHIST}[\text{LUM}(G, X, Y)] + = 1 \\
&\qquad \quad \text{JHIST}[\text{LUM}(T, X, Y)][\text{LUM}(G, X, Y)] + = 1 \\
&\text{NEXT }
\text{NEXT }
\text{FOR } X = 0 \ldots 255 \\
&\quad HT + = \log(\text{THIST}[X]/\text{SIZE}) \times \text{THIST}[X]/\text{SIZE} \\
&\quad HG + = \log(\text{GHIST}[X]/\text{SIZE}) \times \text{GHIST}[X]/\text{SIZE} \\
&\text{FOR } Y = 0 \ldots 255 \\
&\quad HJ + = \log(\text{JHIST}[X][Y]/\text{SIZE}) \times \text{JHIST}[X][Y]/\text{SIZE} \\
&\text{NEXT }
\text{NEXT }
\end{align*}
\]

\[\text{MUTUAL} = HT + HG - HJ\] (4.24)

A final issue with the application of mutual information is one of having a sufficiently large sampling. Figure 4.4 shows several examples of 2D frequency histogram used in the calculation of \( H(T, G_x) \), with colour temperature increasing with frequency. Each is based on the same 8 bit depth (256 grey levels) target and generated luminance images, but a varying degree of quantisation has been applied to the luminance space of each image before calculating the histogram. A typical image may be of the order of \( 640 \times 512 = 327,680 \) pixels in size—in the case where no quantisation has been applied, we have 65,536 elements in the histogram, only 5 ‘samples’ per element. This is exacerbated by the fact that the ovoid shape of the human face will usually result in the generated image having large section’s of non-object (black) pixels, further skewing the distribution of the sample set. Figure 4.4 demonstrates this problem—the central, most interesting area of the histogram appears near empty because the scale of the colour tem-
perature must range from 0 to the relatively very large frequencies of the \((0, 0 \ldots 255)\) and 
\((0 \ldots 255, 0)\) elements. By quantizing the luminance space we group luminance values 
together, producing a coarser but more statistically valid histogram.

4.3.3 Testing mutual information’s suitability as a metric

A series of simple tests were run in order to determine the effectiveness of Eqn 4.23 as a 
correspondence metric, analogous to the testing of Eqn 4.19. Again images of the linear 
model with particular pose and model coefficients and lighting conditions were used as 
a target \((T)\), and the mutual information between \(T\) and a series of generated images 
with varying parameters calculated. As discussed above, an ideal metric would smoothly 
increase as each parameter tends towards the correct value, located at the center of the 
graph as in Figures 4.1 and 4.2. Figure 4.5 below shows a section of the images used, as 
well as the corresponding histograms used to compute mutual information.

Figure 4.7 shows 6 plots of Eqn 4.23’s behaviour. The first three show the effect of 
model pose on the metric, calculating correspondence between a centrally posed target 
and a range of translated and rotated images of the model. The general surface shape of 
plots 1 - 3 on Figures 4.7 and 4.1 seems similar, if not slightly improved in that there 
seems to be greater separation from the metric at the central, correct position and its 
surrounding values. The final three plots repeat the lighting experiments of plots 9 - 12 
in Figure 4.2, which is where we see the greatest improvement over the pixel difference 
metric. With noise on the target image having negligible effect on the mutual information 
metric, we are pleased to see acceptable behaviour when matching pose varying image 
sets to differently lit target images (plots 5 and 6 in Figure 4.7). For further comparison, 
Figure 4.6 collates the output produced by the two metrics (mutual information and pixel 
difference) under different target lighting conditions, and overlays plots of the second, 
most extreme, lighting test. Metric values are appropriately scaled so that the plots both 
range from 0 \ldots 1—It seems clear that mutual information provides superior behaviour.
Figure 4.4: **Effect of different levels of luminance space quantization on joint histogram** The colour temperature guide must be scaled to fit all values—as we decrease quantization the relative size of the center elements is reduced compared to the elements in the top row and left column, leading to the histogram’s centre becoming more obscure.
Figure 4.5: **Mutual information varying with pose** Mutual information is calculated between each of the generated images with a target image (identical to the central image). The joint histogram of each of the target/generated image pairs is shown beneath, note how the histogram converges to diagonal as correspondence is achieved.
Figure 4.6: Comparison of mutual information and pixel difference metrics. Computed correspondence metrics between rendered target image with offset light position and generated set with pose variation of rotation in X and Y. First row: pixel difference, second row: mutual information, third row: target images used, fourth row: two views of plots with most severe lighting offset, appropriately scaled and overlayed. Mutual information provides more informative behaviour, peaking correctly at the central position.
Figure 4.7: **Testing mutual information as a correspondence metric** 2 Plots 1 - 3, top row from left to right: target image of model in ‘0-position’ compared to model over X and Y translation, X and Y rotation, Z translation and Z rotation. Plot 4, bottom left: effect of translation of light position in X-Y plane. Plots 5 - 6: model rotation in X and Y axis, with target model image using light offset of 1,1 and -2,-2 respectively.
4.4 Search algorithms

With mutual information providing an error metric for correspondence, the final component of the 3D to 2D matching system is a suitable non-linear optimization algorithm to perform the hypothesis-testing within the pose and parameter space of the model. This is a problem of non-linear minimization. Figure 4.8 shows a 2D analogy of the many-dimensional problem of model correspondence. From our initial starting parameters (in Figure 4.8 (0, 0)) we wish to search through the 2D parameter space to find the globally maximum position in terms of the metric value at that point in parameter space (in this case near (1, 1)). The simplest method to use is a ‘coordinate ascent’ search. A starting set of parameter values \( S^0 \in P \) and particular ‘search direction’ \( p \) is chosen from the space of parameters \( P \) (in the example of Figure 4.8, \( P = \mathbb{R}^2 \)), such that \( p \) is aligned with a coordinate axis. \( P \) is linearly searched using a constant step size \( \lambda \) in both directions along \( p \), using \( S^0 \) to provide the other parameter values so a metric can be computed. For the next iteration, \( S^{i+1} \) (equal to \( S^0 \) with the starting value for \( p \) replaced with the newly found improved value) is used as a starting position, and a different parameter \( p \) chosen to be searched. The algorithm terminates when no ‘movement’ in parameter space results in an improvement in metric value, a possible minor extension being to reduce step size \( (\lambda) \) or use bisection in an attempt to obtain the calculated parameter values to a higher degree of accuracy.

1. Assume given \( S^i \), step size \( \lambda \), search direction \( p \), metric \( f() \) and termination criterion \( count \). Let \( S^U = S^D = S^i \)

2. Repeat

3. While \( f(S^U + \lambda p) > f(S^U) \), \( S^U = S^U + \lambda p \)

4. While \( f(S^D - \lambda p) > f(S^D) \), \( S^D = S^D - \lambda p \)

5. \( S^{i+1} = f(S^U) \), if \( f(S^U) > f(S^D) \)

6. \( S^{i+1} = f(S^D) \), otherwise

7. \( p = \text{permute}(p) \), where \( \text{permute}([a \ b \ldots x]^\top) = [b \ldots x \ a]^\top \)

8. Until \( count = \text{Dim}(p) \)

In order to evaluate the effectiveness of search algorithms to control the iterative matching process, a testing framework was developed. This essentially consisted of using a rendered target image with known pose and model parameters, perturbing these parameters in a random direction by a small value to provide suitable ‘slightly-off ideal’ starting parameters for the search algorithm. Search algorithm effectiveness was quantified by simple sum-squared-difference comparison of the model and pose parameters.
Figure 4.8: **Coordinate ascent algorithm in two dimensions** The algorithm ‘walks’ through parameter space in small steps, moving in one dimension at a time.
output at algorithm termination and those used to generate the target. Table 4.10 below gives results for all the search algorithms tested. Even without extensive testing, it is a straightforward observation that Eqn 4.4 suffers from two important problems; lack of guaranteed convergence, and extreme susceptibility to local minima—we discount Eqn 4.4 as a viable search algorithm immediately.

4.4.1 Gradient ascent method

A natural improvement over coordinate ascent is the gradient ascent method—here the search direction is not limited to a single axis, now being dependent on the gradient of the metric function at the starting point of each iteration. This technique is based on the principle of steepest ascent, at each iteration we follow the direction which leads to the locally maximal metric value. Consider the relation $I = f(S)$, then $S$ is a particular set of pose and model parameters, $f$ is a non-linear function which generates an image of a the linear object model using the specified parameters $S$, then calculates the mutual information ($I$) between the target and generated image. Now let the starting point for the gradient ascent iteration be $S^i$, then:

$$S^{i+1} = S^i - \alpha \left[ \frac{\delta f(S^i)}{\delta S^i.1} \cdots \frac{\delta f(S^i)}{\delta S^i.(|S^i|)} \right]^T$$  \hspace{1cm} (4.25)

where $\alpha$ is some small constant step size

$S.x$ is the $x$th element of vector $S$

So far, our proposed gradient ascent method requires the use of the first derivative of $f$; the complex, non continuous nature of $f$ makes the explicit computation of $f'$ impossible. As an alternative we can use finite differences to approximate $f'$ at $S^i$ with respect to a particular dimension by construction of $h$, a 0 vector except for a small 'step size' in that dimension only. A choice of finite difference techniques can be applied, as described below in Eqn 4.26. We note that use of forward differencing ($D_F(h)$) minimizes the number of function calls required, but provides a worse approximation than central differencing ($D_C(h)$), which requires roughly twice the number of function evaluations.

$$D_F(h) = \left( f(S^i + h) - f(S^i) \right) / |h|$$

$$D_C(h) = \left( f(S^i + h) - f(S^i - h) \right) / (2|h|)$$  \hspace{1cm} (4.26)

4.4.2 Levenberg-Marquardt method

A popular iterative search algorithm is the Levenberg-Marquardt method (LM) [55, 72], essentially a slight variation on Newton iteration [72]. Although not suitable for use with the mutual information metric, it can be used with with a form of the pixel difference metric, and its popularity makes it worthy of our interest. For the purposes of clarity we define LM in the context of our specific application, but a good general discussion of both Newton and LM iterative methods can be found in [33]. Instead of maximising a function with a single metric value, LM acts to minimize the RMS error of a vector of residuals output from a function. Let $P_x = f(S)$, then $f$ is the function that generates an image
from a set of pose and model parameters and compares it with a target. \( \mathbf{P} \), the \textit{measurement vector}, is a vector of luminance differences between the generated image and target, one element per pixel. It follows naturally from the definition of pixel difference that the best possible fit will be when \( \mathbf{P} \) is a 0 vector, i.e. all pixels in the target image are identical to the generated image. Let this ideal target be represented as \( \mathbf{P}^\star \). Our aim is thus to find \( \mathbf{S}^\star \):

\[
\mathbf{P}^\star = f(\mathbf{S}^\star) + \epsilon, \text{ such that } \|\epsilon\| \text{ is minimized} \tag{4.27}
\]

where \( f \) is a non-linear metric function.

As above, we begin our iterative method with an initial estimate for \( \mathbf{S}^0 \), which we label \( \mathbf{S}_0 \). This is refined under the assumption that \( f \) is ‘locally linear’, whereby we can define \( \epsilon_0 \) by:

\[
\epsilon_0 = \mathbf{P} - f(\mathbf{S}_0) = \mathbf{P} - f(\mathbf{S}_1) = \epsilon.
\tag{4.28}
\]

\( J \) is thus the Jacobian matrix \( \mathbf{J} = \frac{\partial \mathbf{P}}{\partial \mathbf{S}} \). Our iterative method must calculate an improved estimate for \( \mathbf{S}^\star \), \( \mathbf{S}_1 \) which we demand \( \mathbf{S}_1 = \mathbf{S}_0 + \Delta \) and from Eqn 4.28 must minimize:

\[
\mathbf{P}^\star - f(\mathbf{S}_1) = \mathbf{P}^\star - f(\mathbf{S}_0) - \mathbf{J} \Delta = \epsilon_0 - \mathbf{J} \Delta \tag{4.29}
\]

Thus, \( \mathbf{S}^\star \) is obtained from the initial estimate \( \mathbf{S}_0 \) via repeated approximation \( \mathbf{S}_{i+1} \) = \( \mathbf{S}_i + \Delta_i \), where \( \mathbf{J} \Delta_i = \epsilon_i \), \( \mathbf{J} \) being the Jacobian \( \frac{\partial \mathbf{P}}{\partial \mathbf{S}} \) at \( \mathbf{S}_i \) and \( \epsilon = \mathbf{P} - f(\mathbf{S}_i) \). The final step is to solve the linear least-squares problem \( \mathbf{J} \Delta = \epsilon \). The augmented normal equations are solved with this \( \lambda \) to give a value for \( \Delta \). If this results in a decrease in error when applied to Eqns 4.28 and 4.29, \( \Delta \) is accepted and \( \lambda \) is divided by 10 (by convention) at the start of the next iteration. If the current value of \( \lambda \) results in a \( \Delta \) that increases error, \( \lambda \) is multiplied by 10 and a new \( \Delta \) computed—this iterative sub-step continues until a value for \( \Delta \) is found that results in a reduction of error. A justification for this sub-step can be found by consideration of the effect of varying \( \lambda \) on the algorithm’s behaviour. When \( \lambda \) is very small the method is very similar to Newton’s iteration, when \( \lambda \) is very large LM behaves in a similar manner to a gradient descent method as described above.

Although LM represents a significant enhancement over standard Newton iteration, results were variable. The algorithm terminated with relatively few iterations, and in general over simple comparisons (i.e. with a normally lit target image) it was quite effective,
even when started with conditions comparatively far away from the correct pose. A change in lighting conditions appeared to have a serious negative impact on the algorithm’s performance, and it was felt that as well as being let down by the simplistic pixel difference metric, the LM method was probably insufficient to fully examine the function space. This was particularly apparent with the optimisation of model parameters ($\alpha_0$..., $\alpha_4$), which can result in a very subtle variation in pixel difference. Tables 4.10 and 4.11 summarise the test results for matching using the LM iterative method.

### 4.4.3 Downhill simplex method

A rather different search methodology is offered by the Nelder-Mead *downhill simplex method* [62], perhaps to an extent made popular by the inclusion of a slightly modified version with the colourful name ‘Amoeba’ in the successful [72]. Amoeba is certainly less efficient than LM in terms of the number of times $f(S)$ must be evaluated, but as is shown by test results below, offers a more thorough examination of the pose and model parameter space. Amoeba works by maintaining a non-degenerate geometrical figure called a ‘simplex’, which when existing in $N$ dimensional space has $N + 1$ vertices (along with interconnecting edges or polygonal faces). Of course, our function $f(S)$ is many-dimensional, but for the purposes of clarity and diagram simplicity let us limit this discussion to the optimization of a non-linear two dimensional function $g$. We observe that at the time of writing we must maximise over an 11 dimensional space (6 pose and 5 model parameters), however this is likely to increase greatly in future as we expand our basis set of models—with the 100 models suggested by Vetter and Blanz we have a 106 dimensional function to optimize. Helpfully, the amoeba function in lower dimensions has a direct geometrical interpretation, Figure 4.9 shows how, with three two-dimensional vertices the simplex of $g$ is a triangle, and how the various stages of the algorithm affect the simplex.

At the start of each iteration the algorithm will have knowledge of $N + 1 = 3$ vertices, $S^0$...$S^2$, and their function values $g(S^0)...g(S^2)$. These are ordered, and renamed such that $g(S^0) \geq g(S^1) \geq g(S^2)$. Given we wish to maximise $g(S)$, $S^0$ is considered the ‘best’ vertex, and $S^2$ the ‘worst’. The centroid of the $N$ best vertices is found, $\bar{S}$:

$$\bar{S} = \frac{1}{N} \sum_{n=0}^{N-1} S^n$$  \hspace{1cm} (4.31)

Next, the *reflection* vertex $S^r$ is found, $S^r = \bar{S} + \alpha (\bar{S} - S^N)$. With a suggested value of 1 for $\alpha$, geometrically the reflection vertex is the reflection of the worst vertex in the line (or plane in higher dimensions) of the other vertices. If $g(S^0) \geq g(S^r) \geq g(S^N)$ (i.e. it improves the simplex but is not superior to the best vertex) then $S^r$ is integrated into the simplex, taking the place of the worst vertex and the iteration is over.

If $g(S^r) > g(S^0)$ then it is a new best vertex. Observing that a move in the direction of $S^r$ has resulted in a favourable change in function value we compute the *expansion* vertex, $S^e = S^r + \beta (S^r - \bar{S})$ to see if a further move in that direction is helpful ($\beta$ is typically 1). If $S^e$ is strictly ‘better’ than $S^r$ ($g(S^e) > g(S^r)$) then $S^e$ is takes the place of the $S^n$, otherwise $S^r$ is used as above—in either case the iteration is done.
Conversely, if \( g(S^r) < g(S^{n-1}) \) then a move in the direction of \( S^r \) has not resulted in an improvement over the worst vertex in the simplex. The contraction vertex is computed \( S^c = S + \zeta(S - S^n) \), with \( \zeta = 0.5 \) this is effectively the same as \( S^r \) but scaled to be closer to the known best vertices. If \( g(S^c) > g(S^n) \) then \( S^c \) improves the simplex, and is incorporated taking the place of \( S^n \) to finish the iteration.

In the last case where \( S^c \) still fails to improve on \( S^n \), we shrink the simplex (also known as multiple contraction), moving all the vertices closer in to the best one:

\[
S^i = S^0 + \eta(S^i - S^0), \quad \text{with} \ \eta \ \text{equal to} \ 0.5 \ \text{as standard.} \quad (4.32)
\]

The amoeba algorithm is so called because of the behaviour of the simplex as it ‘crawls’ through the space of the function. Each of the possible steps is designed to maintain the non-degeneracy of the simplex with expansion and contraction changing the relative ‘step size’ of the algorithm. We observe that amoeba requires a full simplex as an initial starting condition, which can be constructed from a single initial guess parameter set \( S^0 \) as follows:

\[
S^i = S^0 + \lambda_i \text{Un}_i, \quad (4.33)
\]

...where \( \text{Un}_i \) are the \( N \) dimensional unit vectors, and \( \lambda_i \) are ‘step sizes’ for each corresponding parameter, more precisely an estimate of the ‘characteristic length scale’ of the function in each dimension. Here, \( \text{Un}_i \) are constructed such that a movement in parameter space of \( \lambda_i \text{Un}_i \) results in approximately the same change in function value. Termination criteria are usually defined in terms of a minimum move tolerance, i.e. if the
newly added vertex is within a certain distance of the removed vertex, or the change in function value at $S^0$ in the simplex is sufficiently small over consecutive iterations.

### 4.5 Testing methodology, results and conclusions

As discussed above, the various search algorithms were compared for effectiveness by running the algorithm a number of times using starting pose and model parameters a known small offset from those of the rendered target image, the metric determining success being difference between the algorithms optimized output parameters and those used to render the target. At the time of testing the model had 11 parameters, three rotation, three translation and five model parameters ($\alpha$ in the discussion above). It is well known that initial conditions ($S^0$ above) have a large influence over the outcome of Newton’s iteration, and to an extent LM optimization. In an attempt to quantify the significance of starting position on our search algorithms, tests were conducted with $S^0$ having increasingly larger variation from the target image’s parameters. We see matching performance degrading with increase in the size of the starting position offset, and note that under ‘simple’ matching conditions, LM and amoeba offer similar results. Even so, a change in target lighting conditions leaves Amoeba with mutual information clearly superior, at the cost of many more function evaluations when compared to LM with pixel difference and coordinate ascent methods. To quantify the difference, a typical LM optimization requires around 250 evaluations, with amoeba using 650.

Section §4.1 states that eventually our intended subject matter for performing matching will be real life captured images, almost certainly from a variety of different sources. Real life images are quite different to the pre-rendered target images in use at the moment, but one property that can be simulated quite easily is noise. As we might expect from the investigations carried out in §4.3.3, results below indicate that a noisy target image has a negligible effect on matching.

#### 4.5.1 Finding starting conditions for $\alpha$

As a byproduct of developing a test-bed environment for the search algorithms, various issues relating to the starting position for the algorithms have occurred. It is apparent from Table 4.11 that a good first estimate for model pose will, in general, result in a better match. At the moment the synthetic nature of the testing allows easy specification of ‘not quite perfect’ starting parameters, but when real life images are to be matched this will be more difficult. An interactive system for specifying a first estimate on pose is straightforward conceptually [9], however providing a first guess at model parameters is more difficult. There is no intuitive link between model parameters and the output face, save that ‘increasing the value of parameter N will make the models appearance more N-like’. After some experimentation an automated solution arose, using the mutual information between the target and each of the basis faces to determine the relative quantities of the model parameters; For specified starting pose $S^0$ and target image $T$ let $G_n, n = 0\ldots4$ be the image of basis face $n$ with pose component from $S^0$, then to find starting model
parameters \( \{ \alpha_0 \ldots \alpha_4 \} \) of:

\[
\{ \alpha_x \}_0 = \frac{I(T, G_x)}{\sum_{q=0}^{4} I(T, G_q)}
\]  

(4.34)

Concluding the chapter, the groundwork for a successful matching system has been laid. Results are encouraging for synthetic target images, and further work must concentrate on improving matching of real world rather than CG targets. Lighting condition optimisation is a concern, as well as obtaining more face models to expand the matching potential of the morphable model. A re-examination of feature based matching techniques discussed in §4.2 may also be useful, perhaps leading to a suggested ‘feature patch’ matching algorithm. This could potentially increase speed and accuracy by selectively applying pixel based matching only on high contrast ‘interest’ sections of the face (such as eyes, mouth and nose), as defined by the vertex structure of the model and 2D features marked on the target.
CHAPTER 4. 2D MATCHING

### Table of results 1

RMS error values for runs of the different non-linear optimisation schemes with various starting parameters.

<table>
<thead>
<tr>
<th>Rotation (All $\times 10^{-2}$)</th>
<th>Translation</th>
<th>Model</th>
<th>Starting offset</th>
<th>Target type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Coordinate Ascent method</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>7.0</td>
<td>0.00</td>
<td>normal</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>3.8</td>
<td>0.01</td>
<td>target</td>
</tr>
<tr>
<td>1.9</td>
<td>2.1</td>
<td>5.7</td>
<td>0.02</td>
<td>image</td>
</tr>
<tr>
<td>2.8</td>
<td>2.8</td>
<td>5.0</td>
<td>0.03</td>
<td>lighting</td>
</tr>
<tr>
<td>4.0</td>
<td>3.2</td>
<td>6.4</td>
<td>0.04</td>
<td>offset</td>
</tr>
<tr>
<td>3.7</td>
<td>4.8</td>
<td>8.5</td>
<td>0.05</td>
<td>by (1,1)</td>
</tr>
<tr>
<td>2.5</td>
<td>0.0</td>
<td>1.2</td>
<td>0.00</td>
<td>target</td>
</tr>
<tr>
<td>1.1</td>
<td>0.8</td>
<td>15.8</td>
<td>0.01</td>
<td>image</td>
</tr>
<tr>
<td>1.6</td>
<td>1.8</td>
<td>24.1</td>
<td>0.02</td>
<td>lighting</td>
</tr>
<tr>
<td>3.0</td>
<td>3.0</td>
<td>26.0</td>
<td>0.03</td>
<td>offset</td>
</tr>
<tr>
<td>3.2</td>
<td>3.5</td>
<td>31.4</td>
<td>0.04</td>
<td>by (-2,-2)</td>
</tr>
<tr>
<td>3.9</td>
<td>4.7</td>
<td>24.8</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>8.7</td>
<td>0.00</td>
<td>target</td>
</tr>
<tr>
<td>1.1</td>
<td>1.1</td>
<td>8.2</td>
<td>0.01</td>
<td>image</td>
</tr>
<tr>
<td>1.4</td>
<td>2.1</td>
<td>10.6</td>
<td>0.02</td>
<td>lighting</td>
</tr>
<tr>
<td>2.6</td>
<td>2.7</td>
<td>21.1</td>
<td>0.03</td>
<td>offset</td>
</tr>
<tr>
<td>3.2</td>
<td>4.1</td>
<td>20.9</td>
<td>0.04</td>
<td>by (-2,-2)</td>
</tr>
<tr>
<td>4.2</td>
<td>4.6</td>
<td>14.3</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>10.9</td>
<td>0.00</td>
<td>target</td>
</tr>
<tr>
<td>1.1</td>
<td>1.0</td>
<td>7.2</td>
<td>0.01</td>
<td>image</td>
</tr>
<tr>
<td>1.9</td>
<td>2.0</td>
<td>8.3</td>
<td>0.02</td>
<td>with</td>
</tr>
<tr>
<td>2.6</td>
<td>3.1</td>
<td>5.9</td>
<td>0.03</td>
<td>50 pixels</td>
</tr>
<tr>
<td>3.1</td>
<td>3.9</td>
<td>9.0</td>
<td>0.04</td>
<td>of noise</td>
</tr>
<tr>
<td>4.0</td>
<td>4.8</td>
<td>12.9</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td><strong>Levenberg-Marquardt method</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>2.6</td>
<td>5.7</td>
<td>0.00</td>
<td>normal</td>
</tr>
<tr>
<td>0.8</td>
<td>2.0</td>
<td>5.4</td>
<td>0.01</td>
<td>target</td>
</tr>
<tr>
<td>0.7</td>
<td>1.9</td>
<td>5.9</td>
<td>0.02</td>
<td>image</td>
</tr>
<tr>
<td>1.2</td>
<td>1.3</td>
<td>4.8</td>
<td>0.03</td>
<td>lighting</td>
</tr>
<tr>
<td>0.9</td>
<td>2.1</td>
<td>5.4</td>
<td>0.04</td>
<td>offset</td>
</tr>
<tr>
<td>1.1</td>
<td>1.5</td>
<td>5.9</td>
<td>0.05</td>
<td>by (1,1)</td>
</tr>
<tr>
<td>0.4</td>
<td>37.6</td>
<td>21.7</td>
<td>0.00</td>
<td>target</td>
</tr>
<tr>
<td>1.0</td>
<td>37.7</td>
<td>20.8</td>
<td>0.01</td>
<td>image</td>
</tr>
<tr>
<td>1.5</td>
<td>37.7</td>
<td>22.2</td>
<td>0.02</td>
<td>lighting</td>
</tr>
<tr>
<td>2.9</td>
<td>39.1</td>
<td>20.3</td>
<td>0.03</td>
<td>offset</td>
</tr>
<tr>
<td>3.6</td>
<td>38.8</td>
<td>22.2</td>
<td>0.04</td>
<td>by (1,1)</td>
</tr>
<tr>
<td>7.7</td>
<td>40.7</td>
<td>26.6</td>
<td>0.05</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.10:
### Table of results 2

RMS error values for runs of the different non-linear optimisation schemes with various starting parameters.

<table>
<thead>
<tr>
<th>Rotation (All $\times 10^{-2}$)</th>
<th>RMS Error</th>
<th>Translation</th>
<th>Model</th>
<th>Starting offset</th>
<th>Target type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Levenberg-Marquardt method</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.4</td>
<td>55.7</td>
<td>49.5</td>
<td>0.00</td>
<td>target</td>
<td></td>
</tr>
<tr>
<td>7.1</td>
<td>53.1</td>
<td>48.8</td>
<td>0.01</td>
<td>image</td>
<td></td>
</tr>
<tr>
<td>7.2</td>
<td>51.8</td>
<td>49.2</td>
<td>0.02</td>
<td>lighting</td>
<td></td>
</tr>
<tr>
<td>10.7</td>
<td>58.7</td>
<td>49.3</td>
<td>0.03</td>
<td>offset</td>
<td></td>
</tr>
<tr>
<td>11.3</td>
<td>60.3</td>
<td>49.6</td>
<td>0.04</td>
<td>by (-2,-2)</td>
<td></td>
</tr>
<tr>
<td>5.9</td>
<td>42.7</td>
<td>49.1</td>
<td>0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>2.4</td>
<td>14.1</td>
<td>0.00</td>
<td>target</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>3.9</td>
<td>11.1</td>
<td>0.01</td>
<td>image</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>4.1</td>
<td>13.9</td>
<td>0.02</td>
<td>with</td>
<td>50 pixels</td>
</tr>
<tr>
<td>1.6</td>
<td>6.0</td>
<td>13.3</td>
<td>0.03</td>
<td>of noise</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>5.4</td>
<td>14.7</td>
<td>0.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>4.9</td>
<td>13.5</td>
<td>0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Amoeba method</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.5</td>
<td>1.1</td>
<td>0.00</td>
<td>normal</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>1.0</td>
<td>6.7</td>
<td>0.01</td>
<td>target</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>1.9</td>
<td>3.6</td>
<td>0.02</td>
<td>image</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>3.1</td>
<td>10.3</td>
<td>0.03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>3.8</td>
<td>9.4</td>
<td>0.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>5.0</td>
<td>14.2</td>
<td>0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>12.9</td>
<td>0.00</td>
<td>target</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>1.0</td>
<td>13.5</td>
<td>0.01</td>
<td>image</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>2.1</td>
<td>18.2</td>
<td>0.02</td>
<td>lighting</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>3.1</td>
<td>16.8</td>
<td>0.03</td>
<td>offset</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>4.1</td>
<td>16.1</td>
<td>0.04</td>
<td>by (1,1)</td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>5.1</td>
<td>18.3</td>
<td>0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.5</td>
<td>3.6</td>
<td>0.00</td>
<td>target</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>1.2</td>
<td>9.2</td>
<td>0.01</td>
<td>image</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>2.1</td>
<td>11.3</td>
<td>0.02</td>
<td>lighting</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>2.8</td>
<td>8.6</td>
<td>0.03</td>
<td>offset</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>3.8</td>
<td>17.9</td>
<td>0.04</td>
<td>by (-2,-2)</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>4.8</td>
<td>14.8</td>
<td>0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>0.5</td>
<td>16.3</td>
<td>0.00</td>
<td>target</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>1.4</td>
<td>8.0</td>
<td>0.01</td>
<td>image</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>2.1</td>
<td>12.7</td>
<td>0.02</td>
<td>with</td>
<td>50 pixels</td>
</tr>
<tr>
<td>0.4</td>
<td>2.7</td>
<td>12.7</td>
<td>0.03</td>
<td>of noise</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>3.6</td>
<td>16.7</td>
<td>0.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>5.0</td>
<td>18.5</td>
<td>0.05</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 5

Flexible bump map capture

5.1 Introduction

With the previous chapters concentrating on the use of a linear object space to produce relatively large scale 3D vertex information for 3D face models, this chapter describes work in a different but related strand on simulation of micro-scale surface effects. With small scale colour information already provided via texture mapping, we are particularly interested in adding geometric detail such as wrinkles and pores. Of course, in an ideal situation there would be no need to separate between micro and macro shape, just rendering using a high enough resolution 3D mesh to incorporate small scale detail. This remains impractical for a real-time environment for the immediately foreseeable future simply because of the enormous processing requirements of such a high resolution mesh. Of course, we can never expect a vertex mesh alone to be able to simulate complex surface effects such as those expressed by a spatially varying BRDF [53]. Here we are beginning to consider the effects of a material at an atomic level, even to an extent its effect on the path of a single photon. In fact, for accurate simulation of human skin we must consider the effect of translucency (photon transmission through materials), using a more advanced abstract approximation known as a BSSRDF [63]. Despite the obvious complexity of analysis, there has been much successful research into the capture and simulation of materials. Examples include laser scanning, lighting capture[19] and the subject of this chapter—the recovery of material properties expressed as BRDF samples[17, 48, 56] or bump maps[10, 78].

Bump mapping is a generic term grouping techniques which use 2D texture to apply 3D detail to a surface, an incremental improvement over more standard flat surface approximation. Bump mapping is most useful in the context of lighting approximation (such as Lambertian) where it assists in the simulation of a surface’s true BDRF. The core idea is that shapes and materials may be geometrically approximated relatively coarsely (in our morphable model, by a triangular mesh of 3D vertices), with 2D texel maps applying intricate surface detail. Currently, most renderers using bump mapping can be grouped into one of two categories:

- Displacement mapping: Here each texel specifies a (signed) micro distance from the
CHAPTER 5. FLEXIBLE BUMP MAP CAPTURE

3D object to which it is applied. Renderers can store this information more compactly and/or render it more quickly than is possible with a dense polygon mesh.

- Normal mapping: Each texel defines the normal to the surface of the object at that point, to be combined with the direction of the general surface normal at that point to give an overall local normal. Performing lighting calculations using this modified normal creates the impression of surface texture or roughness without explicit geometric modelling of surface detail.

Rendering of 3D objects with bump mapping is rapidly becoming a computationally feasible operation, with relatively affordable modern graphics pipelines directly supporting bump mapping within hardware, providing per-pixel lighting calculation superior to traditional per-vertex operations. Bump maps are often generated procedurally, using strategies which are designed to emulate the generic behaviour of certain classes of surface variation. This tends to be unsuitable for human faces, which require more specific surface modelling. 2D bump maps for faces are thus either manually constructed, or captured from actual subjects. With manual creation of bump maps a difficult and time consuming process, our strategy will be to sample surface shape, or more accurately, surface normal estimates by observing the effect of lighting variation on a subject[78, 93]. This chapter describes a first prototype of such a system, which can capture bump maps from near-planar surfaces.

5.1.1 Photometric stereo

By considering the standard Lambertian lighting model with no ambient light, we observe that the normal to a surface at a given location can be found by observation of the intensity of the point on the image over different lighting conditions. By recording the intensity of the image of the object illuminated with a single light at three different known positions, solution of a system of linear equations yields the normal at that point. Below \( L_o \) represents a constant—light intensity multiplied by Lambertian reflectivity, \( l_{1...3} \) denotes three sets of light directions, \( n_{1...3} \) the normal vector, and \( i_{1...3} \) the observed intensities at the points on the surface up to scale \( \alpha \). For additional clarification, we note that in the equation below a particular individual light direction will be of the form \([l_{n,1} l_{n,2} l_{n,3}]\).

\[
L_o \begin{bmatrix} l_{1,1} & l_{1,2} & l_{1,3} \\ l_{2,1} & l_{2,2} & l_{2,3} \\ l_{3,1} & l_{3,2} & l_{3,3} \end{bmatrix} \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix} = \begin{bmatrix} \alpha i_1 \\ \alpha i_2 \\ \alpha i_3 \end{bmatrix}
\]

If relative light and object positions are known, and the camera is fixed relative to the object, these equations are readily solved. In the case where the Lambertian approximation is poor, or shadowing must be accounted for, extensions to the model allow accurate reconstruction of the surface normals. Our goal in this paper is to satisfy the geometric conditions: in testing we use only the simplest lighting model. Although more complex models would be expected to provide more accurate normals, the simple model is adequate to test our geometry.
5.1.2 Existing techniques for bump map capture

Rushmeier et al[78] describe a system for bump map capture from a physical object. Their experiment setup is representative of standard bump map capture systems, using a fixed camera and object with a set of identical lights at known relative positions to the object to provide the required illumination directions. Figure 5.1 sketches such a setup. The accuracy of this model is of course limited by the ability of the Lambertian lighting model to predict actual lighting effects. Three common properties of physical objects which can produce non-Lambertian conditions are specular highlights, reflection and self-shadowing. Highlights and shadows are avoided by discarding particularly low or high intensity values. For accurate results, images must be captured under several lighting configurations, requiring a complex rig with limited portability. Rushmeier et al use 5 lights, and suggest that at least this number are required for accurate capture. In their extension of the process to a moving rig[77], the rig retains 5 lights, rigidly attached to the camera, and is still somewhat cumbersome.
5.1.3 Flexible bump map capture

We propose an alternative capture rig, requiring only a single light and camera, which is moved relative to the sample (or vice versa). Its limitation is that the sample being analysed will be modelled as a single planar surface. This is of course very different to the complex shape of the human face, leading us to use a planar sample object rather than an actual human subject for our tests. Even so, we note that sections of the human face are roughly planar (such as the cheeks and forehead), suggesting that a piecewise planar approximation could be used—this would fit in with the general scheme of bump mapping, which as described above would rely on dense normal maps being applied to coarse planar approximations. Figure 5.2 illustrates two equivalent embodiments of the capture system. For our experiments, we use a fixed light and camera, with the sample being moved; but for more flexible capture, rigidly attaching a light to the camera would allow an extremely portable capture system. A summary of the system follows, the rest of the paper describing these steps in more detail.

In operation, a (video) sequence of images of the object is taken, with the relative position of the camera and light fixed. Four calibration markers are placed on or near the sample, and by tracking the markers the relative position of the light/camera rig is calculated, and an inverse perspective transformation applied to re-render the image of the object as fronto-parallel (see Figure 5.3). The resulting image sequence is analogous to the situation of a static camera and object with several lights[78]. We observe that, in terms of the relative geometry of the capture rig, the two situations: static object, moving rig; and static rig, moving object are equivalent. Furthermore, the relative positions of camera and light are fixed, so a single calibration step suffices for capture of a large number of light-direction samples.
Figure 5.3: Inverse perspective mapping. The markers placed on the sample are tracked from frame to frame, and the object re-rendered to appear fronto-parallel. The resulting images approximate the appearance of a fixed object with moving light source, so photometric stereo may be applied to compute normals.

5.2 Details

The following subsections detail the design and construction of a prototype of our system. We divide the discussion into two parts. In the first, we do not refer to light source position, and describe only the recovery of camera position relative to the object’s markers. This will allow us, in §5.2.3, to describe an easy technique for calibrating the relative position of the camera and light source. The guiding principle of these steps is that any measurements or construction required of the user should be as insensitive to human error as possible. For example, we assume it is straightforward to accurately place markers on a measured rectangle, but not to measure the distance from the light source centre to the (poorly defined) optical centre of the camera. For concreteness we shall describe the system from the point of view of the left hand system in Figure 5.2, with a fixed camera and light, and moving sample on a planar backing plate.

5.2.1 Estimating camera position

Reference to figure 5.3 will reveal four high-contrast markers placed on the sample’s planar backing plate. The positions of these markers is assumed known (although assuming them approximately known is an easy extension). This section describes how the camera-to-sample transformation is computed, which in combination with the calibration information of the next section, gives the sample-to-light vector. Several techniques are
possible for the recovery of camera position from known 3D points, and the book by hartley and Zisserman[33] provides a review. We use the technique of Simon et al[82], which is fast and accurate for coplanar markers.

We may temporarily choose world coordinates in the sample’s plane coordinate system, which we arbitrarily define to be the \( xy \) plane. Denote the four marker positions in that plane by \( \mathbf{p}_{1..4} \) where \( \mathbf{p}_i = (p_i, q_i, 0, 1) \top \) is the representation in homogeneous coordinates. We note also that we may write these as homogeneous 2D points in the sample plane by omitting the third component, giving \( \bar{\mathbf{p}}_i = (p_i, q_i, 1) \top \).

We are given a video sequence of images of the object in various positions, and observe the 2D image coordinates \( \bar{x}_{1..4} \) of the projections of the 3D points. The camera position is defined by the \( 3 \times 4 \) projection matrix

\[
\mathbf{P} = \mathbf{K} [\mathbf{R} | \mathbf{t}]
\]

where \( \mathbf{R} \) is a \( 3 \times 3 \) rotation matrix, and \( \mathbf{t} \) is the translation of the camera. The matrix \( \mathbf{K} \) represents the internal calibration parameters of the camera, as defined in Eqn 4.1. We assume that \( \mathbf{K} \) is approximately known—for the DV camcorder used in our experiments, it was sufficient to guess a focal length of \( 1000 \), and assume square pixels \( (a = 1, s = 0) \), with principal point at the image centre. We will need to work with the columns of \( \mathbf{R} \), which we denote by \( \mathbf{r}_{1..3} \), so \( \mathbf{R} = [\mathbf{r}_1 | \mathbf{r}_2 | \mathbf{r}_3] \).

Observing the action of \( \mathbf{P} \) on the markers \( \mathbf{p}_{1..4} \), we may write

\[
\bar{x}_i = \mathbf{K} [\mathbf{r}_1 \mathbf{r}_2 \mathbf{r}_3 \mathbf{t}] \begin{pmatrix} p_i \\ q_i \\ 0 \\ 1 \end{pmatrix}.
\]

Noting that the third column of \( \mathbf{P} \) is always multiplied by the (zero) \( z \) coordinate of \( \mathbf{p} \), the transformation becomes a 2D homography

\[
\bar{\mathbf{x}}^i = \mathbf{K} [\mathbf{r}_1 \mathbf{r}_2 \mathbf{t}] \bar{\mathbf{p}}_i = \mathbf{H} \bar{\mathbf{p}}_i
\]

Where the \( 3 \times 3 \) matrix \( \mathbf{H} \) parametrizes the homography. Efficient linear methods exist [33] to compute \( \mathbf{H} \) from the four point correspondences \( \bar{x}_i \leftrightarrow \bar{p}_i \).

On computing \( \mathbf{H} \), the remaining column of \( \mathbf{R} \) is then computed as follows. In the absence of noise, the matrix \( \mathbf{M} = \mathbf{K}^{-1} \mathbf{H} \) has columns \( [\mathbf{r}_1 \mathbf{r}_2 \mathbf{t}] \) so \( \mathbf{r}_3 \) may be computed as the cross product \( \mathbf{r}_1 \times \mathbf{r}_2 \). In practice, due to inaccuracies in the measurements of the image positions \( \bar{x}_i \), the columns of \( \mathbf{M} \), now denoted \( [\mathbf{m}_1 \mathbf{m}_2 \mathbf{m}_3] \) will not satisfy the orthonormality constraints on the columns of a rotation matrix: \( \|\mathbf{m}_1\| = \|\mathbf{m}_2\| = 1 \) and \( \mathbf{m}_1 \cdot \mathbf{m}_2 = 0 \). A more accurate estimate of the rotation may be obtained by first truncating \( \mathbf{M} \) to the special form where these constraints are satisfied. Figure 5.4 illustrates the process, i.e. construction of an orthonormal basis set from a pair of 3D directions. As the homography is defined only up to scale (as it is a mapping between projective spaces), the matrix may be assumed scaled so that \( (\|\mathbf{m}_1\| + \|\mathbf{m}_2\|) = 2 \). Then the following steps yield a matrix which satisfies the appropriate constraints.
Figure 5.4: Constructing an orthonormal basis. Given a pair of 3D directions \((M1, M2)\), the closest orthonormal pair \((R1, R2)\) and their cross product \(R3\) are computed.

1. Normalize \(m_1, m_2\) i.e. \(m_i = m_i/\|m_i\|\).
2. \(r_3 = m_1 \times m_2\). Normalize \(r_3\).
3. \(tmpa = m_1 + m_2\). Normalize \(tmpa\).
4. \(tmpb = r_3 \times tmpa\). Normalize \(tmpb\).
5. Set \(r_1 = tmpb + tmpa\). Normalize \(r_1\).
6. Set \(r_2 = tmpb - tmpa\). Normalize \(r_2\).

Assembling the columns \(r_{1,2,3}\) into \(R\) completes the process.

5.2.2 Nonlinear refinement of reprojection error

The preceding paragraphs describe the estimation of camera position from measured 2D point positions, and provide reliable solutions via closed-form (or provably convergent) algorithms such as eigenvalue computation or the singular value decomposition[29]. However, the estimates can be significantly improved at small computational cost by a maximum likelihood estimation of the parameters[33]. This is a process of nonlinear minimization of the forward model which generates the 2D point tracks. This process optimizes the error in the system not by least-squares approximations of matrices as above, but at its source, namely the image plane. The free parameters of the system are the six
parameters of camera pose, and one for focal length. Given the known marker positions and image points, written \( \vec{x}_i \) in non-homogeneous coordinates, the objective function to be minimized is

\[
\epsilon(R, t, f) = \sum_{i=1}^{4} \| \vec{x}_i - \pi(K[R \ t] \ p_i) \|^2
\]

Where the projection function \( \pi([x, y, z]^\top) = (x/z, y/z) \). Parameterization of the rotation \( R \) avoids the singularities of Euler angles or equivalent three-parameter forms by using a quaternion. The gauge freedom introduced by this overparameterization is handled via the Levenberg-Marquardt algorithm\[26\], as discussed in §4.4.2. This minimization of the reprojection error directly reduces the image-plane difference between the projected 3D points and their associated 2D markers, computing the optimal transformation in terms of registering the images.

Typical performance is exemplified by the images in Figure 5.3, where the RMS distance of markers to the reprojected points is typically around 5 pixels before the nonlinear optimization, reducing to approximately 1.5 afterwards.
Figure 5.5: **Calibration of light-source positions.** Green crosses show the detected points, red the reprojections from the model—these can be found at the corners of the backing plate and the centre of the image of the light bulb. The geometric model is estimated by minimizing the reprojection error of all five points in all images simultaneously.
5.2.3 Estimating light position

The system as described to this point covers the estimation of camera position and rectification of the image into an approximation to the fronto-parallel view. The remaining task is to compute the position of the light source for each rectified image, which is the focus of this section. The primary constraint on light source position is that the relative positions of light source and camera are fixed during the acquisition—only the relative position of the light and camera system changes. Therefore, if we know the position of the light source in camera coordinates before acquisition, we can transform this position to sample coordinates to obtain a light-source direction for each of the registered fronto-parallel views.

Given accurate measuring equipment, and accurate knowledge of the 3D position of the effective optical centre of the camera, it would be possible to physically measure the light’s position in camera coordinates. However, such measurements are inconvenient, time-consuming to obtain, and extremely difficult to do accurately. Fortunately a more accurate solution can be obtained with less effort by adopting an image-based calibration procedure. By placing a mirror on the sample backing plate, the plate can be moved in front of the camera so that the reflection of the light source is visible. Figure 5.5 shows several images of this procedure. In each image, the position of (the reflection of) the light source is manually specified using the mouse.

From one such image, the relative position of the sample plane and the camera coordinate system can be computed as described in §5.2.1, and it is a matter of straightforward geometry to go from the 2D coordinates of the image of the light centre to determine the 3D line (in camera coordinates) on which the light source must lie. From two such images, two such 3D lines are obtained, in camera coordinates, both of which include the light source—Figure 5.6 shows a typical arrangement. In exact geometry, the intersection of these lines provides the light-source position, and calibration is complete. We observe that the calibration remains valid as long as the parameters of the light-camera system (most importantly their relative position) remains constant. This means that calibration can be carried out ‘off-line’ for portable capture rigs, such as the second system in Figure 5.2.

In practice, of course, the lines are not accurately computed, due to inaccuracies in the localization of the backing plane corners and the manual indication of the light-source centre. By using more than the minimum of two images, a least-squares estimate[39] of the intersection point can be computed, which will improve accuracy. However, the estimate will be biased unless the errors in the 3D lines are accurately modelled. The optimal estimate is obtained[33] using an extension of the nonlinear estimator described in §5.2.2.

Suppose $V$ views are available. The parameters of the system are: $V$ camera positions $(R_i, t_i)_{i=1}^V$, the camera focal length $f$, and the light position $L$. These parameters determine the light position with optimal camera transformations. Let $\vec{x}_{[k]}$ be the image of $p_i$ in the $k$th image. The reflection of the light through the $xy$ plane may be written as a constant transformation matrix $F$ premultiplying $L$, and the 2D image coordinates of the center of the light in image $k$ are denoted $\vec{x}_{[k]}$. The objective function to be minimized is
then:

\[ \epsilon(R_1 \ldots R_V, t_1 \ldots t_V, f, L) = \sum_{k=1}^{V} \epsilon_k, \]

with \( F = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \)

where \( \epsilon_k \), the error in image \( k \), is

\[ \epsilon_k = \left( \sum_{i=1}^{4} \frac{||x_{ik} - \pi (K[R_k t_k]p_{ik})||^2}{\text{Reprojection error of point } i \text{ in view } k} \right) + \frac{||x_{ik5} - \pi (K[R_k t_k]FL)||^2}{\text{Reprojection error of light in view } k} \]

This minimization has the effect of optimally distributing the 2D localization error of the 3D components. The 3D rays are guaranteed to intersect, as they all emerge from the light source \( L \). This means that \( ad \ hoc \) schemes for 3D error propagation are not needed.

To demonstrate the accuracy of light position estimation, \( V = 6 \) different images of the mirror were used. Figure 5.5 is used to demonstrate the effectiveness of the non-linear optimisation. In each calibration image, green crosses indicate \( x_{ik} \), and red crosses \( \pi (K[R t_k]p_{ik}) \). Again the reduction in average reprojection error is from 7 pixels at the initial estimate to around 1 to 2 pixels at convergence. Execution time for the calibration stage is a few seconds on a modern desktop PC.

### 5.2.4 Tracking the markers

Whilst digital still cameras offer much better resolution than a DV camcorder, one major advantage of video over stills is that it is easy to capture large quantities of frames in...
Figure 5.7: **Calculated light direction.** The light direction, projected into the fronto-parallel view, is superimposed on the rectified images. This is the input to a photometric stereo algorithm which recovers surface normals. Comparing the light source direction to the shadow directions indicates that the direction is qualitatively well estimated.

very little time. By automating the detection of the position of the high-contrast markers in each sample image, we can easily capture data for hundreds of camera positions. A simple ‘brute force’ search for the marker points has proved relatively effective so far. This works by storing the images and positions of the markers in the first image (the positions of the markers having been manually defined for the first frame). The search then examines each possible location for the markers in the following images and the locations which give minimal root-sum-squared pixel difference from the new image are returned. The computational overhead of this method can be large, and is reduced by limiting the search to within a specified radius of pixels from the position of the center of the marker in the previous image. One possible improvement to this method is to use an adaptive search, i.e. to update the stored images of the markers after finding them in the new image. This could allow for change in the perceived shape of the markers caused by perspective effects as the sample is moved, but has potential problems with cumulative error—if a marker is tracked incorrectly then its image will be updated falsely, eventually leading to ‘drifting’ type effects.

5.2.5 **The capture process**

Our experimental setup followed the first arrangement suggested in figure 5.2, i.e. a static light and camera with moving sample. Equipment comprised a Canon DV camcorder, tripod, a standard office angle-poise lamp, along with a roughly 140mm by 110mm rectangular sample. A handle was attached to the back of the sample so it could be manipulated within the camera view manually.

Calibration was carried out under room lighting. For capture, we needed the capture sequence of the sample to be as close to zero ambient light as possible, so only the desk lamp illuminated the scene.

Identification of the corners of the mirror in the image is facilitated by the having the observed size of the mirror in the image as large as possible. A reasonably sized mirror proved rather ungainly to manipulate, and so instead a small much lighter mirror was
Figure 5.8: **Calculated normal map.** Computed surface normals are superimposed onto the five captured images of the sample used to generate the map. Because of the deviation from Lambertian imaging conditions, there is a systematic error in the normals, but their local consistency gives us confidence that flexible capture is as accurate as traditional, laboratory-bound systems.

attached to the reverse of a larger plastic case.

Once the light position relative to the camera $L$ and focal length $f$ has been computed, the system is calibrated. We are now able to calculate the direction of the light for any point $Q$ on the object, given the camera position. (We observe that $Q$ will be of the form $Q = (x, y, 0, 1)^T$):

$$\text{Direction} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix}^{-1} \mathbf{L} - \mathbf{Q}$$

Figure 5.7 shows the set of inverse perspective mappings from Figure 5.3, overlaying a 2D projection of the calculated direction to the light from the center point of the sample.

The final stage is to collate the inverse perspective mappings and use them to create normals as described in §5.1. The density of the field is in real terms limited by the resolution of the camera—the quality of the reconstructed fronto-parallel view is reduced as the angle between the object and the camera increases. Even so, Figure 5.3 shows that a reasonably good re-rendering is possible despite fairly severe foreshortening in the image.

For the purposes of demonstration, a 20 by 18 grid of normals was constructed from 3 images of the sample. These were then overlaid on the input sample image for inspection. As noted by Rushmeier et al[78], highlights and shadowing mean that some of the calculated intensity values must be rejected, leaving some positions on the object with fewer than the required three values for normal computation. Inclusion of additional images of the sample provides the necessary data—a topic for further investigation is how
Figure 5.9: **Rendered normal map.** By applying the Lambertian lighting model (computing the angle between the light and surface normal for each texel in the normal map) we can render the normal map and overlay it in the corresponding image in the input sequence. Red pixels indicate there was insufficient data to compute a normal at that point, showing the need for greater than 5 input images.

to optimise the choice of which fronto-parallel views are used in the normal calculation. Even a short captured sequence will contain many hundreds of images—we would intend to maximise the range of object orientations used during the computation of the surface normals, whilst avoiding images containing extremely large object-camera angles due to the suspected loss in re-rendering quality. Typical processing time on a desktop PC is around 5 seconds per image, leading to a total time of approximately 2 hours for a 60 second sequence.

### 5.3 Conclusions

We see in Figure 5.8 a set of well formed calculated normals, with a generally smooth continuity of direction over the board. A more effective way to examine the results is to re-render the sample as a normal mapped quadrilateral, which can be observed in Figure 5.9. Examination of these images reveals a relatively good representation of the sample—the sample has been successfully approximated as discussed in §5.1. As with all bump mapping techniques, secondary surface effects such as self-shadowing are not modelled. This characteristic is shared by our computed bump maps—the rendered surface is less convincing at acute angles to the camera, such in as the second image in Figure 5.9.

Although non-Lambertian effects on the surface mean there is a systematic error in normal direction at the extremities, we are confident that this error is common to both fully
calibrated[78] and flexible bump map capture. Ongoing work includes the implementation of more sophisticated techniques to estimate the normal map in order to accurately quantify the system’s efficacy.

An interesting extension is to the case where the marker positions on the planar surface are known only approximately. This might happen if only archive footage is available, and four arbitrary high-contrast points are tracked. Then approximate world coordinates for the markers can be guessed, but must be refined in order to obtain an accurate normal map. Given enough images, we can in fact include the positions of the markers in the nonlinear optimization, noting that \( p_1 \) and \( p_2 \) may be arbitrarily assigned the origin and \( x \)-axis direction, but the remaining pair must be added to the parameterization of the error function \( \epsilon \).

This chapter has shown that recent systems for bump map capture using calibrated light sources and static cameras may be made more flexible by allowing the movement of a light/camera rig, if the surface to be sampled is of low curvature. I have introduced a novel technique for calibration of such a rig and demonstrated the computation of surface normals from the system’s output. Future enhancements are expected to allow the extension of the system to highly curved samples, such as the human face, by application of the approximate geometry of the morphable model.
Chapter 6

Future Work

My overall project goals for the next two years can be summarised as completing the work on matching the model to images of real faces, extracting new geometry from images, extracting micro scale surface information from images, and if time permits an exploratory look at animation of speech and expression via the morphable model. These points are expanded below.

Improving the matching system

As mentioned in the conclusion of Chapter 4, the matching system in use at the moment should be considered a rather ‘brute force’ approach. Vetter and Blanz propose a much more sophisticated algorithm requiring many more computationally quicker iterations, comparing small chunks of the rendered face with the target image. It is possible that a version of the modified POSIT equation could be used to provide a rough starting alignment between a set of easily marked key features in the target face (corners of mouth, eyes) and the corresponding vertices in the morphable model. We note that we can ensure that the model contains correctly corresponding vertices by simple use of the manual vertex placement system, as described in §3.4.3. By only modifying and matching certain interesting areas of the morphable model we can drastically cut down on the processing time required to recalculate and render, as each single triangle can be calculated and drawn independently. A further concern with the matching system as it stands is lighting. A system for light placement, and perhaps optimization within a small locus of its specified position may be useful. We note that previous experiments have shown that mutual information is fairly tolerant of lighting change, and hence may not be a suitable metric for lighting adjustment parameters.

Shape recovery from stereo

With a working matching system for single images, it should be possible to apply it to stereo and multiple view matching. The position of the model can be used to assist in calculating stereo correspondence between images, hopefully providing enough information to give a dense mapping between pixels in paired images. By calculating relative camera
position we can use knowledge of parallax effects to extract out depth information, and hence new 3D models to be added to the morphable model. Texture map extraction follows fairly naturally once we have a 3D model for the images being matched, and will be necessary for truly realistic model capture. It is through this technique that we hope to add to and expand the model to include many different human faces.

Surface detail recovery

Again, with a sufficiently large number of basis faces and a high quality matching system it should be possible to produce good approximations to a large set of target images. Continuing the work from Chapter 5, we can use this model to give a high quality approximation of the surface geometry of a face in an image, or video. We already have the basic system required to extract detailed surface normal information from a video sequence of a planar surface under relative light-camera movement, and literature indicates refinements of this technique can be used to glean even more information, such as a Horizon map \[76\], or approximation of an actual BTF \[53\]. It is proposed that the morphable model can provide the necessary geometrical information to make extraction of this kind of detail from the complex, curved shape of the human face possible.

Animation

By capturing images of the same human subject with different expressions it should be possible to capture 3D facial models of similar expressions also. By computing a correspondence between these models it should be possible to quantise the difference between the two models, and apply this to other models to create the same expression. Although initially we would be concerned with basic emotions such as happy, sad, angry etc, a sophistication of this work would be to capture a human speaking, perhaps investigating the possibilities of synching animation to a sound track, as accomplished with a 2D morphable model by Ezzat et al \[23\].
**Timetable for future work**

The following table provides a rough estimate of expected time requirements for the project. Naturally, elements of work are expected to overlap each other, as reflected in the schedule.

<table>
<thead>
<tr>
<th>Sep 02</th>
<th>Matching System:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Refine matching system to work with real life images</td>
</tr>
<tr>
<td>Dec 02</td>
<td>Shape Recovery:</td>
</tr>
<tr>
<td></td>
<td>Introduce stereo matching, and baseline recovery. Use model to aid geometry extraction</td>
</tr>
<tr>
<td>Mar 03</td>
<td>Surface Detail Recovery:</td>
</tr>
<tr>
<td></td>
<td>Use matched model to provide approximation of faces geometry for small scale BTF extraction</td>
</tr>
<tr>
<td>Jun 03</td>
<td>Animation:</td>
</tr>
<tr>
<td></td>
<td>Capture model under different expressions. Apply this to other faces, perhaps to speech</td>
</tr>
<tr>
<td>Sep 03</td>
<td>Thesis write up</td>
</tr>
<tr>
<td>Dec 03</td>
<td></td>
</tr>
<tr>
<td>Mar 04</td>
<td></td>
</tr>
<tr>
<td>Jun 04</td>
<td></td>
</tr>
</tbody>
</table>
Appendix A

Lectures, meetings, and seminars attended

Lecture courses attended

- Prof. A. P. Zisserman’s *Two dimensional signal analysis*, for 3rd year engineering science undergraduates. Topics covered include 2D image processing, stereo camera systems and baseline recovery, camera geometry, image perspective effects.

- Prof. B Kouvaritakis *Optimisation and Linear Algebra*, for 3rd year engineering science undergraduates.

- Prof. D. W. Murray and Prof A. P. Zisserman’s *Computer Vision*, for fourth year engineering science undergraduates. Topics covered include tracking, camera geometry, rectification, vanishing points, image homographies.

Project meetings

- Weekly meetings with my supervisor, Dr. Andrew Fitzgibbon. Meetings with SCEE supervisor, Dr. David Ranyard on August 2001, April and June 2002. Frequent progress updates via email and presentation of completed work electronically.

Seminars and reading groups

- Weekly attendance of the robotics department’s *Computer Vision Reading Group*, during academic term time this year. Diverse range of topics including image separation, genetic algorithms, markov chains, detecting human motion, background subtraction, rapid human face detection etc.

- Frequent attendance of the robotics department’s seminars during term time. Visiting academic talks about their interest. Diverse range of computer vision related topics such as ‘View Synthesis with Occlusion Reasoning’, ‘Monotone Problems of Optimal Labelling: Applications to Texture Segmentation and Stereo ’ and ‘Face Processing with Bunch Graphs and Rigid Body Motion and Grouping’
Bibliography


[34] (Intel Corporation). IA-32 intel architecture software developer’s manual volume 2: Instruction set reference.


