Performance-driven Facial Animation

Garoe Dorta-Perez, Ieva Kazlauskaite, Richard Shaw

CM50247 - Visual Effects

Unit Leader: Dr Darren Cosker

University of Bath

May 2015

Signature of Author . . . . . .

Garoe Dorta-Perez, Ieva Kazlauskaite, Richard Shaw

Contents

Chapter 1

Introduction

"*Performance capture is a technology, not a genre; it's just another way of recording an actor's performance*." - Andy Serkis

A talented painter or sculptor is able to imagine and reproduce the subtle details of a human face. Hours of training and endless manual adjustments are required before an arbitrary shape resembles a facial expression. Furthermore, a human eye is trained to notice subtle changes in the expression of another human being, which also applies to animated humans, thus even the smallest discrepancies in an animated model are easily detected. Though artists are able to produce good quality and appealing results, the limitations in budget and time so prominent in the entertainment industry motivate the development of more automated, faster and cheaper methods.

The naive approach that allows for faster performance is the keyframe animation - though it is based on manual input, keyframe animation requires fewer frames as the intermediate expressions are interpolated over. However, this approach is still very laborious and time consuming. With the emergence of more sophisticated systems and software for motion capture, development of methods for performance-driven animation has received significant attention from both the industry as well as academia.

Chapter 2

Previous Work

2.1 Facial Animation

The first attempts at using performance-driven facial animation in academia date back to late 20^{th} century while in industry it was first used for guidance rather than production, for instance the facial motion of Gollum in the *Lord of the Rings* was based on the motion capture data of the performance of the role actor Andy Serkis [PSS02]. Another notable example is the pipeline used in the making of *the Polar Express*; the motion capture data was used to construct a multi-layer facial rig that was used by an artist to create the final facial animation [Ben05], examples are shown in Figure 2-1. Though many other examples exist, often the exact technology and methods used in production are not disclosed.

In the last two decades, nearly two hundred academic papers that include words *face*, *animation* and *motion capture* were published in computer graphics and vision journals [Sco]. The increase in the quality of the results is explained by the advancements in both the capturing technology and the computational methods. We shall concentrate on the latter; for a brief discussion of the capturing methods see Section 3.1.

(a) Golum in Lord of the Rings (Image from New Line Cinema)

(b) Train conductor in Polar Express (Image from Warner Bros.)

Figure 2-1: Motion capture examples in the VFX industry.

The work of Guenter *et al.* provides the first detailed and unified system for capturing and reconstructing facial performance [GGW*98]. Their work is mostly focused on motion tracking and production of labelled three-dimensional motion of the dots on the actor's face. Once a sequence of moving dots is acquired, the author's face is scanned to obtain a polygonal mesh that corresponds to the geometry of the face. The motion of the sparse dots is described in terms of offsets from the neutral position. Then the vertices in the mesh are moved by calculating a linear combination of the offsets of the nearest dots, i.e. in each frame a weighted sum of the change in the dot position is calculated and added to the neutral position of the given vertex. The weights are chosen to be zero everywhere except in the one-ring neighbourhood where the weights depend on the distance of the vertex to the dots; the weights must sum to one. An additional stationary ring of dots is added on the edge of the face to ensure there is no motion outside the face. The algorithm suffers from noise introduced during in the tracking, the reconstruction and the three-dimensional scanning. Moreover, the choice of the nearest dots for each vertex in the mesh is not trivial due to the irregular distribution of the tracked dots. The method that assigns the blends has a number of manually adjusted parameters and does not guarantee to find the best set of reference dots. Additionally, the areas around the eyes and lips require a special treatment; the dots above and below the problematic areas are marked and are not allowed to be blended. The authors point out that a number of artefacts are visible in the resulting animation; some of these flaws are attributed to the poor quality of the facial scan, the fact that the reconstruction method is not robust to jitter and incorrect placement of the tracked data.

At around the same time Pinghin *et al.* proposed a method for synthesizing facial expressions form photographs [PHL*98]. The authors concentrate their attention on the capture and reconstruction part of the process. First, a three-dimensional model of the face pulling a number of different expressions is constructed, then the animation is created by morphing these meshes and producing the intermediate motion. The meshes are topologically consistent and the features are marked manually, resulting in matching feature sets. Then linear interpolation between corresponding vertices in the different meshes is used to get a smooth morphing. Compared to the method proposed by Guenter *et al.*, the approach of Pinghin *et al.* puts more emphasis on the capture, reconstruction, and creation of good quality meshes for different expressions resulting in an almost trivial animation production process.

Noh and Neumann addressed the problem of having to create a new model for a new animation by exploiting an existing database of high quality animations [NN01]. Given a new target model, dense correspondence between the source model from the database and the target is estimated using a set of manually selected matching points. The volume morphing is performed using a weighted linear combination of Radial Basis Functions (RBF). However, some manual input is required when matching the features; for this authors employ a set of heuristic rules. Then a cylindrical projection is used to transfer the vertices from the source onto the target model, thus the source vertices are embedded in the target surface. The animation is created by applying the adjusted target motion vector to the source. The adjustments involve scaling, and change of direction according to the curvature of the source surface. Moreover, if the two models have very different geometries, then small neighbourhoods are used to determine the local changes that were imposed during the morphing of the source surface. As in most facial

models, special attention is paid to the area around the mouth; the edges of the lips in the two models are aligned, and the motion transformations take into account the motion of both lips to ensure that the duplicated vertices move consistently. The presented approach has a number of limitations; the model requires manual matching, the eyelids, the teeth and the tongue are not incorporated in the model, and it is only able to reproduce the motion of the source model and cannot produce novel movements.

Building on previous work on synthesis of facial expressions, by Joshi *et al.* address the problem of automating the creation of blendshapes [JTDP03]. The proposed method segments the face into characteristic parts that can then be modified independently to make a desired expression thus simplifying the process of creating new blendshapes. The authors argue that one of the advantages of controlling small regions is the increased number of expressions that may be produced using the blendshapes. Each frame in the keyframe animation is produced by calculating the linear combination of blendshapes in each region. A linear elasticity model is used to deform the surfaces. A related method, proposed by Pyun *et al.* generates a new facial expression by combining emotional and verbal key expressions [PKC*03]. The main limitation of this model is that a set of key expressions have to be designed by an artist.

In some situations the sets of blendshapes is only available for the target model. Choe and Ko propose a method that constructs a set of source key shapes given the target key shapes [CK05]. First, a mapping between the source and the target coordinate systems is constructed. Then the method iteratively refines the set of shapes using the captured source data. The authors build on their previous work, and use a muscle actuation basis as opposed to the surface-based key shapes. The elements in the actuation basis are linearly independent and they span the corresponding space, forming a complete basis for the facial expressions [CLK01]. However, the actuation basis is much harder to construct in comparison to the surface-based set of shapes and the relation between the motion of facial muscles and the resulting motion of the facial tissue is not straight-forward.

Vlasic *et al.* proposed an novel approach to facial animation that is based on a multilinear model [VBPP05]. The work offers an alternative approach to blendshape models; it involves creating a large dataset that encodes various features, for instance the identities, expressions and location of vertices. Each dimension of the data tensor corresponds to a unique feature, and due to the independence of modes, each feature may be varied without affecting the others. Such model can be extended to include an arbitrary number of features, and may be used for motion retargeting or actor replacement, when a database of three-dimensional scans is available. Moreover, the probabilistic interpretation of the model is related to probabilistic principal component analysis, and is able to deal with missing data.

A number of dimensionality reduction techniques have been used to produce facial animation; for example Blanz and Vetter apply principal component analysis (PCA) to create a controllable low dimensional model [BV99]. Deng *et al.* use PCA to find a correspondence between the motion capture data are the manually tuned blendshape model [DCFN06]. An extension to nonlinear dimensionality reduction techniques was presented by Wang *et al.* [WHL*04]. The proposed method separates the time-varying facial expressions and the individual style associated with the performer. Then the locally linear embedding (LLE) framework is used to

find a nonlinear mapping from the high dimensional space onto a low dimensional manifold. The LLE method is based on an assumption that small neighbourhoods around each data point may be treated as linear patches. Using the resulting generative model, new expressions or an animation may be produced by sampling from the low dimensional manifold. Dimensionality reduction is used when no blendshape model is available, or when different features of the data need to be extracted.

Most of the previously described methods are unable to capture and reproduce the fine details on the actor's face; Bicket *et al.* introduced a method that is able to create detailed wrinkles on the target model [BBA*07]. The approach is based on decomposition of the model into coarse features from the motion capture sequence, and fine features from the accompanying video. The novel part of their method relies on capturing, analysing and reproducing the wrinkled surface. In the tracking stage, uniform B-Splines are fitted to each fold of the skin. Then the shape of each fold is estimated by exploiting the self-shadowing effect, and finding the gradient that describes the shape of each fold. Moreover, the coupling behaviour is modelled separately. New wrinkles are synthesised by minimising the associated non-linear shell energy; the geometrical model, proposed by Grinspun *et al.* estimates the local curvature of a surface, and is able to capture the behaviour of thin flexible structures [GHDS03]. The facial model with detailed wrinkles relies on the quality of the captured data; in particular, the areas that are prone to small-scale deformations have to be painted in a way that minimises diffuse reflection.

One of the most successful attempts to produce a photo-realistic facial animation is known as the Digital Emily project [ARL*09]. The proposed pipeline involves collecting high resolution data, building a detailed facial rig of the actor's face, producing an animation from the video data, and reproducing the high quality results. The blendshape model was created using approximately 30 three-dimensional scans; since each scan contained more than one discrete shape, a splitting algorithm had to be used to obtain a larger set of controllable shapes. Additional subtle effects, such as the motion of the deformation of the eyelid when the eyes are closed were included to increase the realism of the results. The animation is produced using a number of example poses, and generating predictions for the required pose of the digital model. The entire production process is very time consuming and requires manual input from skilful artists and animators. The authors do not provide a detailed description of the method but their controllable facial rig is publicly available.

One recent development was proposed by Bouaziz *et al.*, who develop a method for real-time facial animation [BWP13]. The new method does not rely on the construction of a three-dimensional expression model associated with the actor prior to the face animation stage. The presented method uses a consumer-level device that captures both the intensity and the depth (RGB-D cameras); previous research by Baltrušaitis *et al.* indicates that the use of multimodal data improves the quality of feature tracking [BRM12]. To achieve real-time results, Bouaziz *et al.* use a template blendshape model that is constantly updated to better match the face of the performer. This update includes two steps. Firstly, PCA is used to construct a low dimensional representation of a large set of different facial meshes, then any new face is constructed by merging the average face with a linear combination of the orthonormal basis vectors produced by PCA. Secondly, surface deformation field is used to further refine the blendshape model.

Then the tracking, retargeting and production of an animation is performed in parallel with the optimisation algorithm that aims to personalise the blendshape model.

Concurrently, Li *et al*. combined blendshape models with facial tracking to animate a target character face [LYYB13]. The authors employ per frame correctives to achieve run time shape correction. The data is captured using RGB-D cameras. The initial state of this method consists of capturing a three-dimensional model of an actor's face, and PCA and a large database of captured facial data is used to construct a generic face. A blendshape model is constructed using the deformation transfer algorithm, proposed by Sumner and Popović [SP04]. Then the motion is produced using blendshapes, and it is refined using an orthogonal adaptive basis. This basis is constructed using PCA, and it contains the initial blendshapes as well as a set of corrective shapes; these additional basis elements correspond to expressions that are not contained in the original shape set. The corrective shapes capture the fine-scale details. Due to the non-linear nature of the resulting blendshape model, Laplacian deformations are used to align the tracked data to the three-dimensional model. Then the expectation-maximisation algorithm is used in the adaptive PCA space to iterative improve the space of the corrective shapes. Specifically, given a number of sufficiently diverse input samples and the initial guess of the corrective space, the algorithm estimates the coefficients of the corrective space. Then, the algorithm finds the model that best explains the samples given the corrective coefficients. After each two cycles of the algorithm, the newly acquired corrective space is orthonormalised, and used to decrease the error during the tracking. This method is capable of producing appealing visual results in real-time. However, it requires a scan of the actors face, and the corrective shapes are not directly used during the retargeting.

A different direction was chosen by Garrido *et al*., who aim to reconstruct high quality three-dimensional models using data only from a monocular video [GVWT13]. The method relies on a pre-designed blendshape model, and uses optical flow to produce three-dimensional motion. Meanwhile, Xu *et al*. improved the retargeting of facial animation from an actor onto a digital model [XCLT14]. The authors use a multi-scale approach, where a targeted optimisation algorithm is used to achieve best results at the course and the fine scale. Moreover, the proposed method provides the user with a set of control tools that are used to alter the automatically produced results. This blendshape model produces high-quality visual results but it still requires significant input from the user.

2.2 Skin Rendering

Rendering realistic skin is a challenging task. As social beings we interact with interact with other individuals on a daily basis, which has made human perception quite sensitive to skin appearance, even more so with human faces. Skin is composed of several layers with different properties, to accurately simulate skin the light transport between this layers has to be simulated. The full effect of light scattering between two points on the surface can be modelled using a Bidirectional Surface-Scattering Distribution Function (BSSRDF).

Weyrich *et al* [WMP*06] proposed a two-layer model for skin rendering, the outer layer

simulates the air-oil interface and the inner layer models the subsurface scattering in the skin. The authors considered the scattering to be homogeneous, with this assumption they measured the skin BRDF of several subjects in a light dome, while the scattering was sampled at three points in the face with a custom made sensor. The BRDF data was fitted to a Blinn-Phong and a Torrance-Sparrow isotropic models, and the scattering was fitted with a single transport coefficient. Donner *et al* [DWd*08] also proposed a two-layer model, however the authors allow for the layers to be heterogeneous. With this addition they are able to introduce the effects of haemoglobin, veins and tattoos. Emotional induced haemoglobin variations have also been explored [JSB*10]. The authors measured the haemoglobin distributions of several subjects in different poses, then a linear combination of the captured data would determine the final haemoglobin distribution for a new sequence. Recently, Iglesias *et al* [IGAJG15] introduced a five-layer model to handle skin ageing. Haemoglobin, collagen and fat changes with age are modelled using the different layers.

Normal maps are use to alter the normals of the scene objects during rendering. This technique is used to add geometric detail to an object at rendering time without actually changing the geometry. Normal maps for skin rendering are usually captured using expensive light domes with a number of synchronized cameras [GTB*13, WMP*06].

Another technique to increase the quality of a face render is to scale the resolution of the textures being used. According to citeTian2011 super-resolution techniques can be classified into four classes: *frequency-domain*-based, *interpolation*-based, *regularization*-based and *learning* -based.

Frequency-domain-based super-resolution methods started with the seminal work of [TH84], the authors transformed the image using the discrete Fourier transform, where it is possible to formulate a system of equations relating the discrete and the continuous Fourier transform coefficients. The system is solved, the new coefficients are apply in the Fourier domain and a new image of higher resolution is retrieved by performing the inverse transformation. More recently, extensions that include denoising stages [CB05] or error handling for registration and blur identification [JF09] has been proposed.

Interpolation-based approaches used the fact that if we have several low-resolution images of a scene, each of them provides an amount of additional information about the scene. This technique projects all the low-resolution images into the hight resolution space and fuses the data maximizing the quality. Bose *et al*. [BA06] proposed a fusing technique using the moving least square (MLS) method, the authors estimated the pixel value as a polynomial approximation of the pixel neighbourhood with adaptive polynomial order for each pixel. Takeda *et al*. [TFM07] proposed a method to generalize this and other problems such as denoising or interpolation into a generalized kernel regression problem.

Regularization-based methods tackle the problem from a probabilistic viewpoint. An estimation using the maximum likelihood (ML) of the high-resolution image $\mathbf{X}^{ML}$ was proposed by Tom *et al*. [TK95], such that

$$\mathbf{X}^{ML} = \arg\max_{\mathbf{X}} P(\mathbf{Y}|\mathbf{X}), \tag{2.1}$$

where $\mathbf{Y}$ the unknown high-resolution image and $\mathbf{X}$ is the low resolution image. Extensions using maximum a posterior (MAP) with Markov random fields [HMD06] and Markov Chain Monte Carlo methods [TM10] have been researched.

In the *learning*-based area, Wie *et al*. [WL00] proposed a technique to synthesized new textures based on a sample image and a random noise. The main idea of the authors work was to maximize the local similarity when choosing a new pixel in the synthesized texture image, based on the already built neighbourhood in the new image. Ashikhmin *et al*. [Ash01] presented a method to generate new textures using a goal image by greedily extending existing patches whenever possible. Hertzmann *et al*. [HJO*01] combined and extended Ashikhmin *et al*. [Ash01] and Wie *et al*. [WL00] methods by adding a second example image and using more complex distance metric to choose the next synthesized pixel. Graham *et al*. [GTB*13] applied Hertzmann *et al*. [HJO*01] example-based filter to generate bump maps with increased quality for skin rendering. An alternative approach using a dictionary of samples was presented by Jianchao *et al*. [YWHM10]. This method is restricted to generating super-resolution images, however, the previous methods support a wide variety of filter effects. For an in depth analysis of super-resolution techniques, we refer the readers to Tian *et al*. [TM11] survey.

Chapter 3

Methodology

This section presents the methodology behind our proposed facial performance capture system and discusses some of the technical challenges we faced in its implementation.

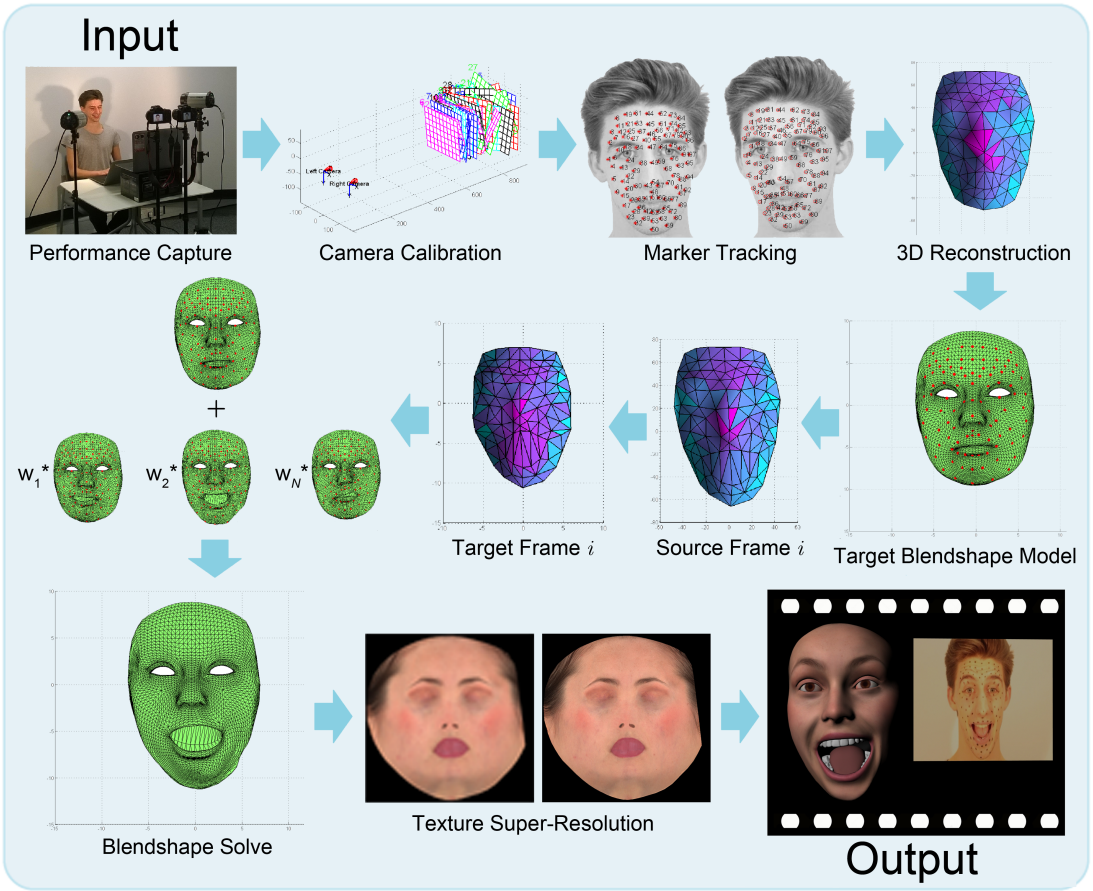


Figure 3-1: A schematic of the proposed pipeline for the performance capture system.

3.1 Data Capture

The first task was to gather some initial input data for the system by recording the facial performance of an actor in the form of a video sequence. Two DSLR cameras were positioned close to the actor's face in a stereo configuration in order to capture the actor's facial performance, as shown in Figure 3-2. A short sequence, consisting of the actor pulling a number of facial expressions and then talking, was recorded on video with a resolution of 640×480 at 60fps. The video streams were roughly synchronised afterwards during a post-processing stage in the computer by aligning the two separate audio signals. The video was also converted into an image sequence of jpeg images. A relatively low image resolution was chosen in order to try to limit the amount of video data we had to deal with, especially when recording multiple sequences at high frame-rates. The cameras were fixed on stands to prevent any camera-shake during the performance capture session and calibration process - since it is important that the camera positions remain static for accurate 3D reconstruction. Positioning the cameras close to one another, at approximately 10 centimetres horizontal separation and at a distance of about 1 metre from the subject, meant that the images from both views would be quite similar - allowing for easier feature matching across the two images. However, the camera spacing was sufficiently large to enable both sides of the face to be in view, and to allow for 3D triangulation of points on the actor's face; a good rule of thumb is for the cameras and subject to subtend an angle of at least 5 degrees. A limitation with the capture system was observed that if the actor rotated his/her head significantly, then one side of the face would be occluded, making detection and reconstruction of some marker points on the face a problem. In a future implementation it would be better to use three cameras - one straight on and the other two slightly to the sides, so that all points on the face can always be seen by at least two cameras.

Figure 3-2: The stereo camera setup used to capture the actor's facial performance.

Markers used to track the facial performance were placed on the actor's face in the configuration shown in Figure 3-3. 97 markers were used in total, and were drawn onto the actor's face with a make-up pencil so as to be easily removed after the capture session. The positions of the markers were roughly based on those used in the Surrey Audio-Visual Expressed Emotion (SAVEE) Database [Sur], as shown in Figure 3-4, although they use fewer markers with just 60 in total. In hindsight, using slightly fewer number of markers in our system might have been preferable, as consistently detecting and tracking a large number of markers (and preserving the labelling order of the markers) became quite a time-consuming and labour intensive process, prone to error. Furthermore, rather than simply using a black pencil, it might have advantageous to use markers of a significantly different colour to the tones of the human face in order to aid marker detection and tracking (by colour segmentation), or perhaps even using retro-reflective markers as in many commercial motion capture systems [Vic].

Although the markers were placed all over the face to capture full facial performance, we tried to ensure that specific markers were placed in positions on the face which exhibit large non-rigid deformation, such as around the eyes, around the mouth and along the eye-brows, in order capture the full expressiveness of the face. In a future implementation, it would also be useful to track the rigid head motion by tracking the positions of a number points on the head of the actor, for example by having the actor wear a skull cap with markers. None of the marker positions we chose on the face are truly rigid - so this made removing rigid head movement more of a challenge later, see Section 3.5.1. Another useful addition would be to somehow track the eyelids so we can capture subtle eye movements and blinks. Also, since we only track points on the outside of the mouth, we do not gather information about the lip movement, such as in expressions involving pursing or funnelling of the lips, etc.

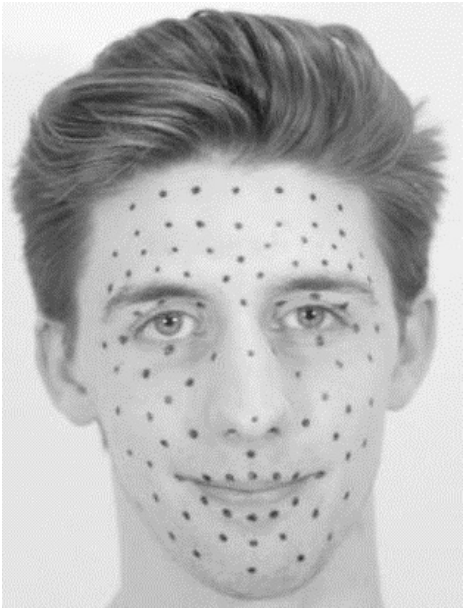


Figure 3-3: The configuration of markers used for tracking the facial performance of the actor.

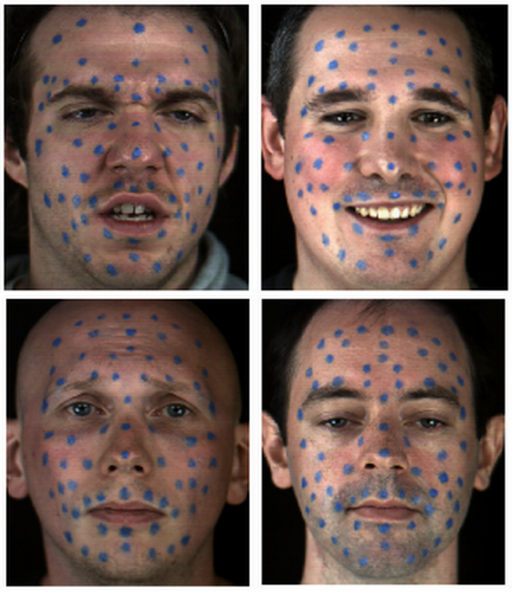


Figure 3-4: Images from the Surrey Audio-Visual Expressed Emotion (SAVEE) Database [Sur].

3.2 Camera Calibration

Once we had obtained a video sequence of the actor's facial performance, the next stage was to calibrate the stereo camera system in order to obtain accurate 3D coordinates of the markers on the face.

Camera calibration is name given to the process of recovering the camera projection matrix **P** from images of a controlled scene. We assume the pinhole camera model and under perspective projection the map between the three-dimensional world coordinates of a point $(X, Y, Z)^T$ and its two-dimensional image pixel coordinates $(u, v)^T$ is a linear mapping in homogeneous coordinates represented by the 3×4 projection matrix [CG00]:

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{3.1}$$

To obtain an estimate of the camera projection matrices for both cameras, we use a checkerboard pattern of known dimensions held in a number of different positions and orientations throughout the scene, in view of both cameras simultaneously, as shown in Figure 3-5. To perform the calibration, we made use of the Camera Calibration Toolbox for Matlab® [Cal] provided by Jean-Yves Bouguet, which automatically detects the grid corners in each image and solves for the intrinsic camera parameters (focal length, principle point, and distortion coefficients) and the extrinsic parameters (rigid rotation and translation). It was important to ensure that the checkboard remained within the frame of both cameras during the entire calibration process, and that the camera positions did not move between the performance capture session and the calibration procedure.

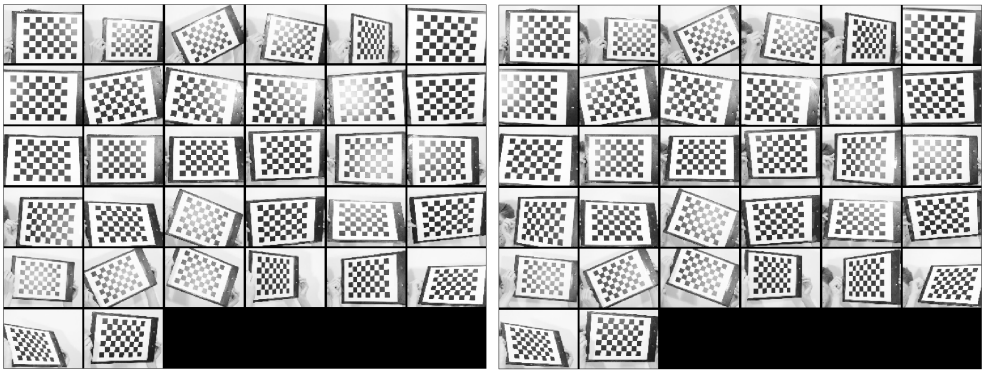


Figure 3-5: A checkerboard pattern was used to calibrate the pair of stereo cameras. For stereo calibration the checkerboard must be visible from both camera viewpoints simultaneously.

As a result of the stereo calibration, we obtain an estimate of the intrinsic matrices **K**, **K′** and external parameters **R**, **t** of the two cameras, from which we can compute the camera

projection matrices $\mathbf{P}$ and $\mathbf{P}'$ for the left and right cameras respectively using the equation for perspective projection:

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \mathbf{K}[\mathbf{R}|\mathbf{t}] \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.2}$$

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \begin{bmatrix} fk_u & 0 & u_0 \\ 0 & fk_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.3}$$

where the left camera is assumed to be centred at the origin and aligned with the world co-ordinate axes, i.e. its projection matrix is $\mathbf{P} = \mathbf{K}[\mathbf{I}|\mathbf{0}]$, and the right camera projection matrix is expressed as $\mathbf{P}' = \mathbf{K}'[\mathbf{R}|\mathbf{t}]$.

The resulting extrinsic parameters describing the rotation $\mathbf{R}$ and translation $\mathbf{t}$ between the two camera views are visualised in Figure 3-6, where the scale in mm. The positions of the checkerboard used for calibration are also displayed. We can see that the cameras are roughly 10 centimetres apart and about 1 metre from the checkerboard pattern as expected.

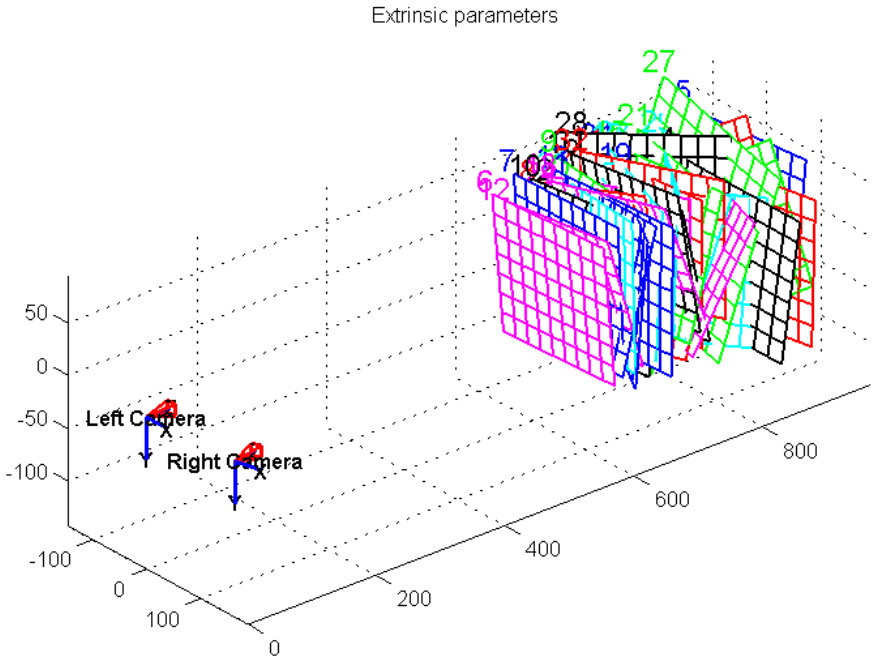


Figure 3-6: The resulting extrinsic parameters of the calibrated stereo camera setup.

From the estimated projection matrices, we can compute an estimate of the fundamental matrix $\mathbf{F}$ relating the two camera views. The fundamental matrix encapsulates the intrinsic

epipolar geometry algebraically and is defined by the following equation [LF96]:

$$\mathbf{u'}^T\mathbf{F}\mathbf{u} = 0 \tag{3.4}$$

$$(u'v'1)\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = 0 \tag{3.5}$$

where $\mathbf{u} = (u, v, 1)^T$ and $\mathbf{u'} = (u', v', 1)^T$ are the 2D pixel coordinates of the same 3D world point imaged in the left and right views respectively. The fundamental matrix $\mathbf{F}$ is a 3×3 matrix, so has 9 elements but only has 7 DoF. This is because it has arbitrary scale, leaving 8 DoF, and is of rank 2 and singular, i.e. $\det[\mathbf{F}] = 0$, leaving 7 DoF. Using the fundamental matrix relation, the equation of the epipolar line in the right image $\mathbf{l'}$ corresponding to the point $\mathbf{u}$ in the left image, on which the feature point must lie, is therefore given by the following expression

$$\mathbf{l'} = \mathbf{F}\mathbf{u} \tag{3.6}$$

Similarly, the epipolar line $\mathbf{l}$ in the left image is given by

$$\mathbf{l} = \mathbf{F}^T\mathbf{u'} \tag{3.7}$$

The left and right epipoles $\mathbf{e}$ and $\mathbf{e'}$ are defined as the point in each image which is common to all the epipolar lines and are given by the null spaces of $\mathbf{F}$ and $\mathbf{F}^T$ respectively

$$\mathbf{F}\mathbf{e} = \mathbf{0} \qquad \mathbf{F}^T\mathbf{e'} = \mathbf{0} \tag{3.8}$$

With the epipolar geometry fully defined, we can then rectify the stereo images for every frame of the captured image sequence so that the eipolar lines are horizontal. This reduces the stereo matching problem to a 1D search problem, as the same 3D world point imaged in both cameras is constrained to lie on the same pixel row in each pair of images. To rectify each pair of stereo images we implemented a function in Matlab®called `StereoRectify.m`, based on the function `rectify_stereo_pair.m` provided in the Camera Calibration Toolbox for Matlab® [Cal] which rectifies each pair of calibration images. This works as follows: we first bring the two cameras into the same orientation by rotating them 'minimally', so as to bring the translation vector $\mathbf{t}$ in alignment with the positive x-axis $(1, 0, 0)^T$. Global rotations are applied to both cameras so that the resulting rigid motion between the cameras is just a translation and the rotation matrix becomes $\mathbf{R} = \mathbf{I}$. The vertical component of the estimated focal length f_u must be the same in both images and we set the horizontal focal length f_v to the same as the vertical, resulting in square pixels, and new principle points are chosen to be the average of the two principle points. New camera projection matrices are computed accordingly

and we warp each image so that the epipolar lines are horizontal. A pair of rectified images for a frame of the captured image sequence is shown in Figure 3-7. Matching image points and their corresponding epipolar lines are plotted. We can see that all the epipolar lines are now horizontal and corresponding points lie on the same pixel row. Therefore, to find matching points in the images we can simply search along the row until a match is found. We apply this warping procedure to all image frames in the captured sequence, resulting in a new stereo-rectified image sequence.

Figure 3-7: Using the stereo calibration results, each pair of images in a captured image sequence is rectified so that corresponding epipolar lines lie on the same pixel row. This reduces the stereo correspondence problem to a 1D search along the epipolar lines.

3.3 Initial Marker Detection

With the stereo camera system fully calibrated, the next task is to compute the 2D image positions of all the facial markers in each left and right stereo pair for all frames in the captured sequence. Before the markers can be tracked through the sequence, their positions need to be initialised in the first frame. We do this semi-automatically, by detecting the facial markers in the left image of the first frame, and then computing the corresponding marker positions in the right image. The initial marker detection is achieved by detecting SIFT features in the left image. We employ the open source Matlab®implementation VLFeat [VF08] to do this.

The marker detection process for the first frame of a given stereo image sequence is carried out as follows. First the user defines a region of interest in the left image around the face of the actor to be tracked, by drawing a contour in the image. We make use of the Matlab®function `roipoly` for this, as shown in left image in Figure 3-8. Then, SIFT features are detected within the selected image region, choosing the appropriate SIFT parameters to pick out features of the correct scale; around 5 pixels in diameter - this may take a few tries to get right. We found that it was quite difficult to pick out just the marker features because the size of the markers varied significantly, and so to detect the majority of the markers we had to search for a wide range of scales of SIFT features. This can be seen in the middle image of Figure 3-8, where running the

SIFT feature detection has picked out a lot of other unwanted points, particularly around the eyes and mouth. However, we found that it was still faster and often more accurate to detect points this way using SIFT and then refine the detection after, rather than manually select all 97 points by hand. Once reasonable SIFT features have been detected, additional feature points can then be added manually, and also wrongly detected points can be removed, by interactively selecting certain points the user wishes to change. We implemented a piece of code to allow the user to simply click on marker points they want to remove or add. This gives the resulting initial marker positions in the left image as shown on the right in Figure 3-8.

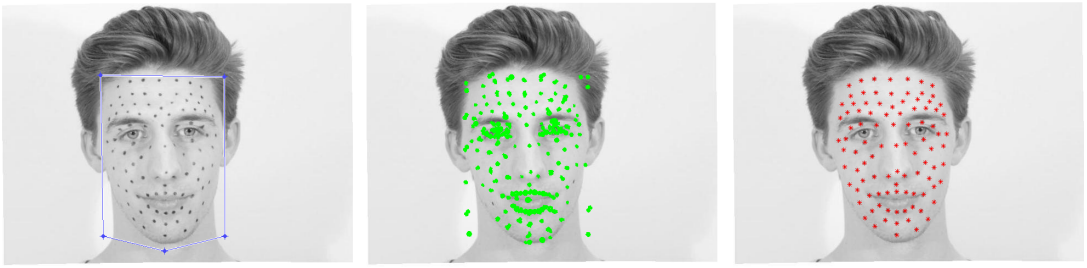


Figure 3-8: Facial markers are detected by first selecting a region of interest, detecting SIFT features, and then removing unwanted points or adding additional points. The final points are plotted in red on the right.

Once the user has successfully detected all 97 facial markers in the left image, the corresponding marker positions are found in the right image by searching along the corresponding epipolar lines; simply the same row of pixels in the right image since all the images have been stereo rectified beforehand. The matching marker positions in the right image are found by computing the normalised cross-correlation (NCC) between image patches, as given in Equation 3.9. So that for each marker point $\mathbf{u}$ in the left image we take the surrounding square patch of N pixels and compute the NCC with each N pixel patch in the right image, for all pixels in the same row. I_i^L is the intensity of pixel i in an image patch in the left image, and similarly I_i^R in the right image. μ^L, μ^R, σ^L, and σ^R are the means and standard deviations of pixel intensities in image patches in the left and right images respectively. The pixel location in the right image that produces the maximum NCC response is chosen as the matching marker position to the marker in the left image. Since the two views are similar, we also apply the constraints that corresponding marker positions must lie close to each other in the two images (within some threshold distance) and that the order of the markers on the plane of the face must be maintained, i.e. if a marker is to the left of another in one image, it must also be to the left of the corresponding marker in the other image.

$$C_{NCC} = \sum_{i=1}^{N} \frac{(I_i^L - \mu^L)(I_i^R - \mu^R)}{\sigma^L \sigma^R} \tag{3.9}$$

This produces the result shown in Figure 3-9, where all markers in the left image are matched to corresponding markers in the right image. The markers are numbered from 1 to 97 to illustrate this. The ordering of the marker indices in the left image is arbitrary (simply

numbered from left to right as they appear in the image), but the markers in the right image must adhere to this ordering and maintain the same order as they are tracked throughout the entire image sequence.

Figure 3-9: Detected facial markers in the left image are matched to markers in the right image pair by computing the normalised cross-correlation between image patches around the points.

3.4 Marker Tracking

With all 97 facial markers initialised in both stereo images for the first frame, the next procedure was to track the positions of the markers as they move throughout the image sequence.

3.4.1 Optical Flow

The first method we explored for tracking was to simply compute the global optical flow for the image sequence, from one frame to the next, and to move the positions of all the markers along the direction of the flow. Optical flow algorithms are commonly based on the assumption that the pixel intensity of the same imaged 3D point will be similar from one image frame to the next, i.e.

$$I(\mathbf{u}, t) = I(\mathbf{u} + \mathbf{v}, t + 1) \tag{3.10}$$

where $I(\mathbf{u}, t)$ is the intensity at pixel $\mathbf{u}$ in the image at time t, and $\mathbf{v}$ is the 2D flow or displacement vector, however this assumption rarely holds exactly. We made use of two high-performing optical flow Matlab®implementations to do this, namely C. Lui's optical flow code [Lui] and M. Black's implementation [Bla]. However, we found that simply tracking the points in this way was not robust enough to coherently track all the markers on the face, particularly when the actor made fast facial movements such as opening and closing the mouth during speech. Even when recording at 60fps, these fast facial movements resulted in slightly blurred

image frames causing the point tracking to go awry, even after just a small number of frames. Furthermore these algorithms, although very sophisticated optical flow implementations, took far too long to compute for lengthy image sequences, so an alternative approach was required.

3.4.2 KLT Tracker

A more sophisticated method of tracking was then explored - the Kanade-Lucas-Tomasi (KLT) feature tracking algorithm [ST94], which is implemented in Matlab® [KLT]. The KLT tracker solves for the flow iteratively and employs a pyramidal image scale implementation to deal with a range of pixel displacements. We can define a residual function $\epsilon(\mathbf{v})$ to be:

$$\epsilon(\mathbf{v}) = \epsilon(v_x, v_y) = \sum_{x=u_x-w}^{u_x+w} \sum_{y=u_y-w}^{u_y+w} (I_t(x, y) - I_{t+1}(x + v_x, y + v_y))^2 \tag{3.11}$$

where $\mathbf{v} = (v_x, v_y)$ is the optical flow at (x, y) in image frame I at time t to time $t + 1$, and w is half the width of the image window around pixel $\mathbf{u} = (u_x, u_y)$. We want to find the flow $\mathbf{v}$ that minimises the residual ϵ. The algorithm generates an image pyramid where the image resolution is halved on each pyramid level, such that I^0 is the original 0th level image and $I^{1\ldots L}$ are the lower resolution pyramid images. If we consider a pixel $\mathbf{u}^0$ in the original image I^0, its corresponding position in the nth pyramidal image is $\mathbf{u}^n = \mathbf{u}^0 \times 2^{-n}$. The pyramid tracking works as follows: $\mathbf{v}^L$ is computed at the lowest pyramid level L and is used as the initial guess to the next level above $\mathbf{v}^{L-1}$, continuing up to the original image $\mathbf{v}^0$. The flow at each level $\mathbf{v}^n$ is small and so is easy to compute using standard KLT algorithm - by minimising the residual ϵ^n at that level. The final optical flow solution is the sum of the flows at all levels, i.e. $\sum_{n=0}^{L} 2^n \mathbf{v}^n$ [Kan].

We track the feature points from one frame to the next independently in both camera views using the KLT tracker. However, since we have previously stereo rectified all frames in the image sequence, and due to the fact that the same imaged 3D point should lie on the same pixel row (epipolar line), then we essentially have two estimates of the y-coordinate for each feature point; one prediction in each camera. We can exploit this by simply setting the y-coordinate of each pair of corresponding feature points to the average of their y-values, for every frame. This means that if in one camera view the feature point starts to drift from the correct position, and in the other view the point is being tracked correctly, the correct point helps compensate for the error in one view and prevents the tracking from deviating further from the correct position. Some of the results from tracking the points through the sequence are shown in Figure 3-10.

We implemented a procedure in which if the KLT tracker starts to lose the tracking of a marker, the user can manually re-estimate the position of the marker in the image at that particular frame. The tracking then continues from that re-estimated point. The tracking method we use is still not very robust, and it can be quite difficult to obtain a good track for all 97 facial points over a large number of frames. One reason for this is because we are simply performing a global track of all points and not taking into account the local object information. For

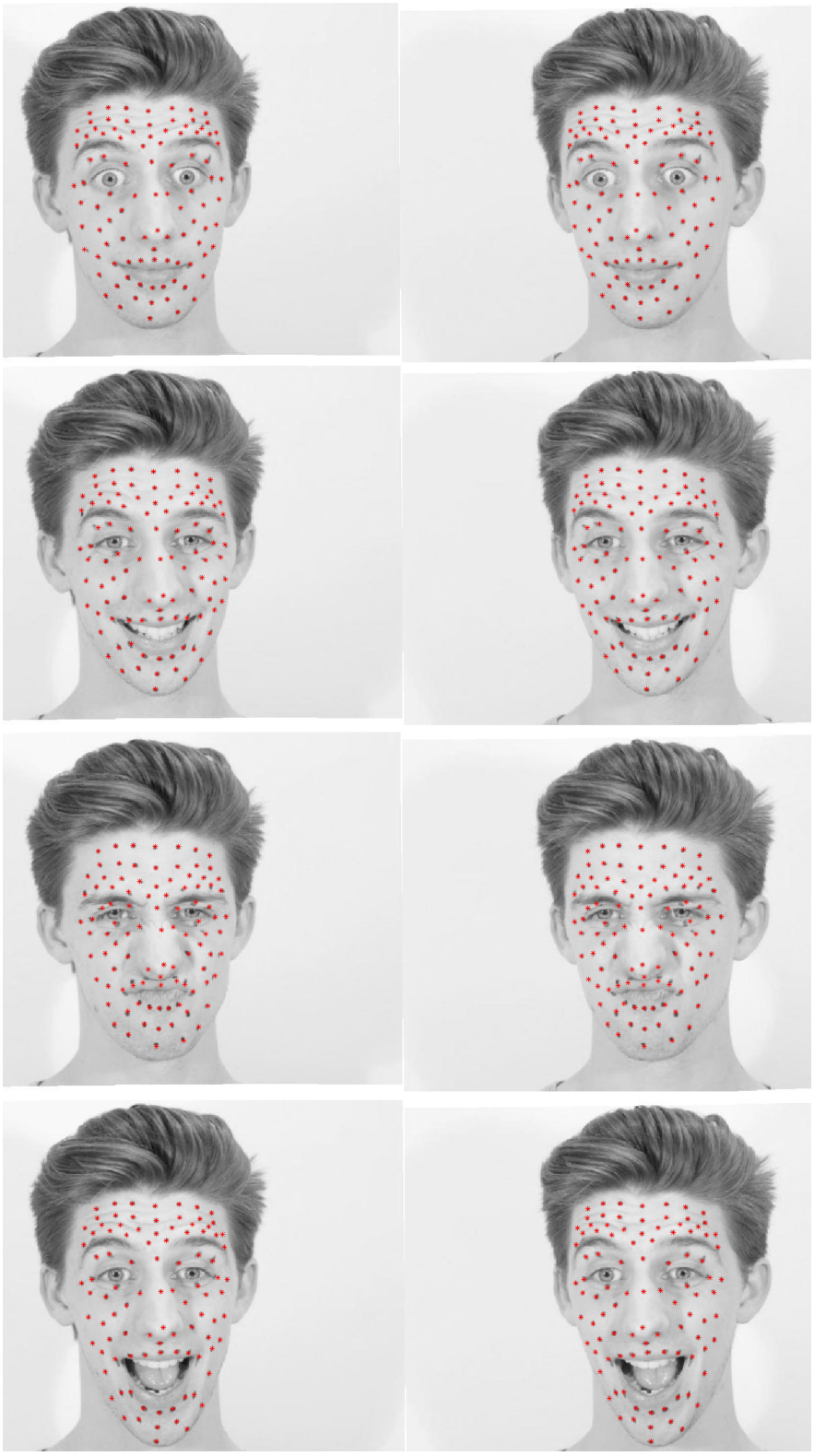


Figure 3-10: A number of frames from the image sequence with facial markers tracked using the KLT algorithm.

example, if we kept a running probabilistic model of the face points, the system would be able to handle occlusions and would be more robust to non-rigid movements and other sources of tracking error. One interesting approach is implemented by Li *et al*. [LCBT13] that uses a mesh based object tracking method which would likely perform much better in this situation. They propose an underlying mesh which penalizes local movements and preserves smooth global ones. Constraints on the local deformations expressed in Laplacian coordinates encourage local regularity of the mesh whilst allowing global non-rigidity.

3.5 Sparse Facial Reconstruction

Once the tracking stage is complete and we have obtained the 2D image coordinates of the facial markers for every frame in a stereo image sequence, we then compute a sparse 3D reconstruction of the face over the sequence. In one frame, given a pair of image correspondences $\mathbf{u} = (u, v, 1)^T$ and $\mathbf{u'} = (u', v', 1)^T$, in the left and right views respectively, of the same 3D world point $\mathbf{X} = (X, Y, Z, 1)^T$, we can write two equations using the projection matrices $\mathbf{P}$ and $\mathbf{P'}$:

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \mathbf{P} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \qquad \begin{pmatrix} su' \\ sv' \\ s \end{pmatrix} = \mathbf{P'} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{3.12}$$

For each image point in a pair stereo images we can write two equations:

$$u = \frac{p_{11}X + p_{12}Y + p_{13}Z + p_{14}}{p_{31}X + p_{32}Y + p_{33}Z + p_{34}} \quad u' = \frac{p'_{11}X + p'_{12}Y + p'_{13}Z + p'_{14}}{p'_{31}X + p'_{32}Y + p'_{33}Z + p'_{34}} \tag{3.13}$$

$$v = \frac{p_{21}X + p_{22}Y + p_{23}Z + p_{24}}{p_{31}X + p_{32}Y + p_{33}Z + p_{34}} \quad v' = \frac{p'_{21}X + p'_{22}Y + p'_{23}Z + p'_{24}}{p'_{31}X + p'_{32}Y + p'_{33}Z + p'_{34}} \tag{3.14}$$

Rearranging, these are expressed in the form $\mathbf{AX} = 0$ as follows, where $\mathbf{p}_i$ denotes the ith row of the projection matrix $\mathbf{P}$,

$$\begin{bmatrix} u\mathbf{p}_3^T - \mathbf{p}_1 \\ v\mathbf{p}_3^T - \mathbf{p}_2 \\ u'\mathbf{p'}_3^T - \mathbf{p'}_1 \\ v'\mathbf{p'}_3^T - \mathbf{p'}_2 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = 0. \tag{3.15}$$

Since two equations are acquired from each image point in the two views, this gives a total of four equations with three unknowns - the problem is over-constrained. A linear least-squares solution can be found by performing Singular Value Decomposition on the matrix $\mathbf{A}$.

The 3D coordinates of the point $\mathbf{X}$ are the singular vector corresponding to the smallest singular value of $\mathbf{A}$, i.e. the last column of the matrix $\mathbf{V}$, where $\mathbf{A} = \mathbf{UDV}^T$. This is implemented in Matlab®in the function `Reconstruct.m`, which takes in the two camera projection matrices for the pair of stereo cameras obtained during the calibration procedure, and the matching 2D image points, returning an array of corresponding 3D points. Since the 2D marker positions have been forced to lie on corresponding epipolar lines, the epipolar cosntraint is satisfied and the camera rays that pass though each image point are constrained to intersect at a point in 3D space.

The resulting 3D reconstruction of the neutral facial expression is plotted in Figure 3-11. The final face mesh is computed by performing a Delaunay triangulation [Ede00] over the estimated vertices. This 3D reconstruction process is repeated for all frames in the recorded image sequence to produce a sparse 3D sequence of the actor's facial movements. A number of frames from the sequence are shown in Figure 3-12, along with their corresponding 3D reconstructions.

3.5.1 Stabilising Head Movement

It is important for the actor's head to remain still throughout the sequence so that when solving for blendshapes (Section 3.6.3), the reconstructed facial expressions are expressed as true non-rigid deviations from the neutral facial expression. Proper stabilisation of the head is required to avoid artefacts in the resulting blendshape facial animation and retargeting process [BB14]. If the rigid head movement is not removed, then the expression shapes will contain this 'baked-in' rigid motion, and any facial animation constructed from the expressions will also contain this unwanted rigid head motion. In industry, face-stabilisation is still usually performed manually; Weise *et al*. [WLVGP09] aim to remove the rigid motion component of multiple expressions using ICP registration, Vlasic *et al*. [VBPP05] use Procrustes alignment, and Beeler [BB14] presents a particularly interesting method of aligning expressions to a model of the underlying skull.

In our implementation, we attempt to remove the majority of the actor's rigid head motion by performing a Procrustes analysis [Gow75] using a number of approximately rigid points. At least four points are required for the Procrustes alignment. The rigid points were chosen to be the tops of the ears, the tip and bridge of the nose, and the centre point on the top of the forehead, as shown by the dots marked in Figure 3-13. However, note that none of these points are actually truly rigid, so this is just an approximation to the true stabilisation. In a future implementation, it would be better to actually track a few points known to be practically rigid, for example by placing markers on the top of the actor's head.

The Procrustes analysis computes a least-squares estimate of the transformation that aligns the vertices of the facial expression in a certain frame to the neutral face, given the rigid vertex correspondences. In Figure 3-13, the rigid correspondences are plotted in red for the neutral face, and blue for a particular frame i. The right image in Figure 3-13 shows the result of the alignment, and we can see that the two meshes are approximately aligned, but it is not perfect - there will always be some error. We do this for the entire image sequence so that in every frame

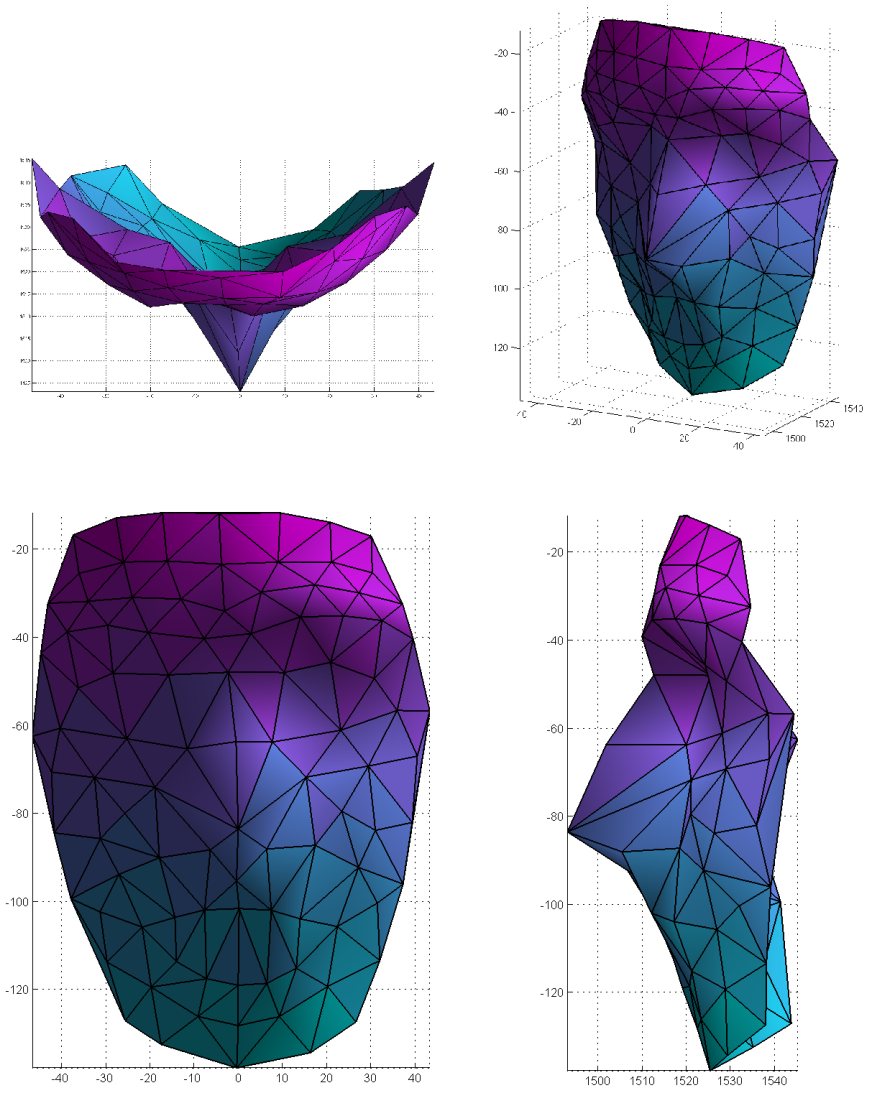


Figure 3-11: A 3D reconstruction of the neutral facial expression.

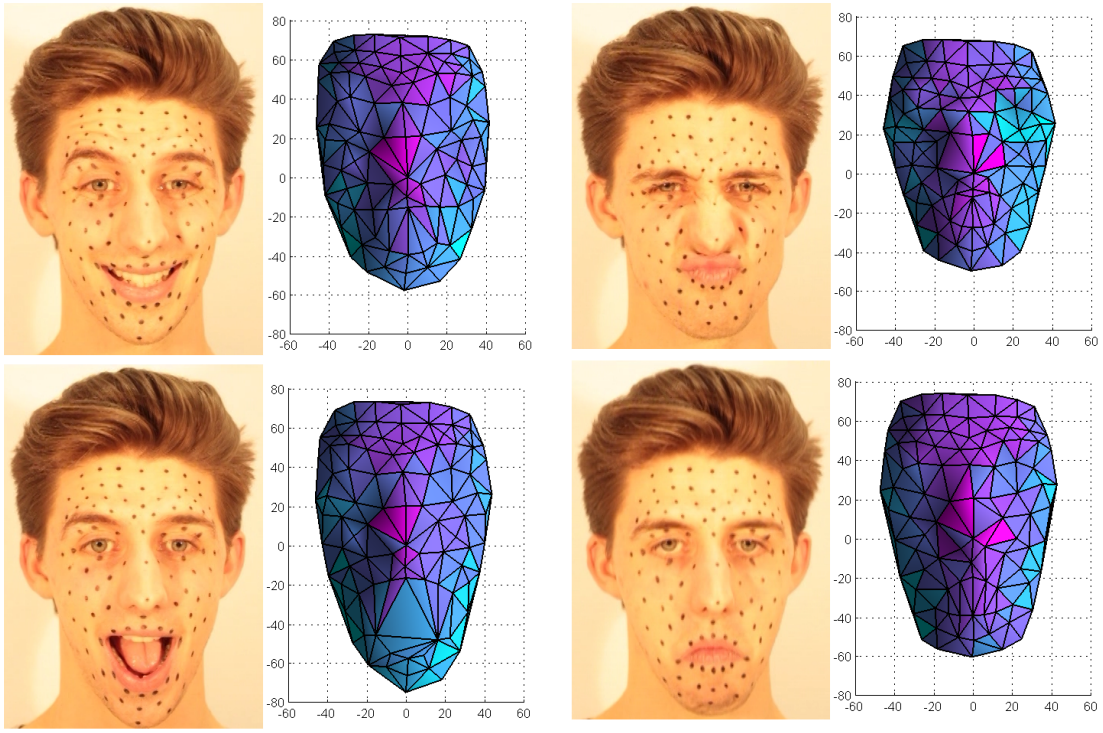


Figure 3-12: A sparse 3D reconstruction is computed for every frame of an image sequence using the detected marker positions and the calibrated camera projection matrices. A sample of image frames and the corresponding 3D reconstructions are shown above.

the head pose is aligned to the neutral position. The Procrustes analysis returns an estimate of the scale, rotation and translation at each frame and we store these estimates so that we can later apply the inverse transformation to put the rigid head movement back in as a last step in the final facial animation.

We have found that the average error over an image sequence is usually around the order of 10^{-3}. However, it is not simply the case that the smaller the error means the better the alignment of the expression to the neutral face, because the head can be orientated in such a way that can make the error in the rigid vertex correspondences small, but can result in an unnatural head pose. Through experimentation, we have found that choosing points that span the entire face and are placed towards the extremities of the face, such as the ears, generally results in a better alignment than rather selecting a few points close together around the centre of the face, such as around the mostly rigid nose region. This makes sense because a small deviation in points around the centre of mass will lead to larger displacements in the other points, whereas small deviations in points at the edges will have much less of an impact on points in the centre of the face.

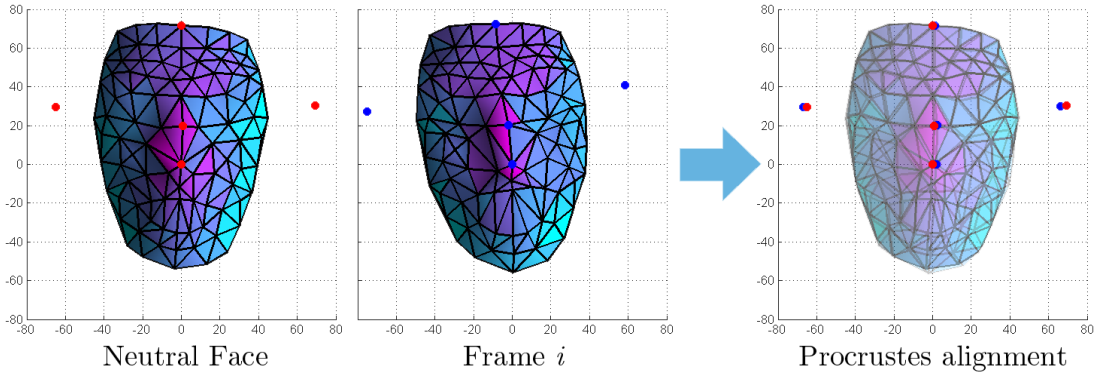


Figure 3-13: The head pose in each frame of an image sequence is aligned to the neutral face position (left) using Procrustes analysis. The rigid vertex correspondences are shown in red for the neutral pose and blue for a given frame.

3.5.2 Sources of Error

There are a number of possible sources of error that can accumulate during the reconstruction process and can cause the resulting 3D reconstruction of the facial performance to not be entirely accurate. It is important to identify and minimise these errors since they will propagate throughout the remaining processes in the pipeline, culminating in a final blendshape facial animation that looks and behaves unnaturally.

Firstly, there may be errors induced if the video sequences from both cameras are not properly synchronised; we only roughly synchronised them by aligning the audio, it would be better to use a properly synchronised multi-camera system. Furthermore, we record at 60fps but, according to the Nyquist criterion, we must sample at atleast twice the frame-rate of the

frequencies we want to capture [Nyq02]. We found that for some fast facial motions, 60fps was not fast enough and the frames would be blurred, causing erroneous detection of the markers. Secondly, errors can be caused during the camera calibration procedure, resulting in inaccurate projection matrices and errors in the stereo rectification, i.e. the epipolar lines will not be perfectly horizontal. Thirdly, errors in the tracking will mean that the reconstructed 3D points do not correspond exactly to the marker positions on the face. A better, more probabilistic tracking method is required that is robust to non-rigid displacements and occlusions. Finally, we reconstruct points using a simple linear least squares approach, but it would be good to perform a final optimisation procedure that minimises the reprojection errors, for example by applying a bundle adjustment optimisation for the non-rigid case.

3.6 Facial Animation

Given the motion capture data, our goal is to produce an animation of a 3D face model that resembles the captured motion. We make use of the Digital Emily blendshape rig designed by Alexander *et al*. [ARL*09], as shown in Figure 3-14. To achieve this we need to transfer the motion captured in the source domain (the actor) to the target domain (the Emily rig), and so in this section we introduce the techniques used to match three-dimensional surfaces and point clouds, and then the optimisation methods used to estimate the parameters that drive the motion of the digital model.

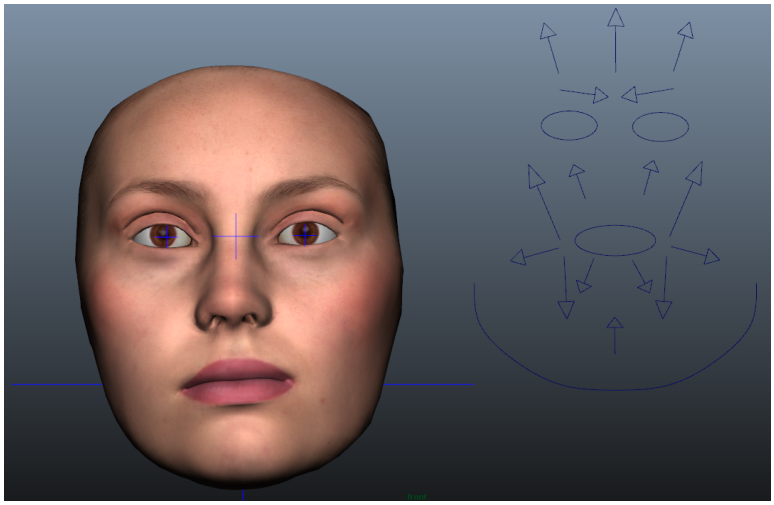


Figure 3-14: The Emily blendshape rig used for producing the facial animation.

3.6.1 Thin Plate Splines

Thin plate theory addresses problems that commonly arise in areas of natural sciences and engineering when trying to model the behaviour of a thin sheet of some material. The possible processes include but are not limited to stretching, bending, crumpling, buckling, shrinking,

straining and tearing. The corresponding mathematical model is based on ideas from differential geometry as well as continuum mechanics, and the set of equations describing the aforementioned phenomena are often notoriously difficult to solve. Therefore, in Computer Graphics, as well as other fields, a number of simplifying assumptions are made when constructing a thin plate model.

A thin plate is considered to be a two dimensional object, i.e. it is assumed that the thickness is infinitesimal. The geometry of the object is often simplified to reduce the computational cost. Thin plate splines (TPS) are a two-dimensional counterpart of the cubic spline [SS91]. TPS are a deformation method based on the assumption that a thin surface deforms in a way that minimises the surface bending energy. The bending energy is proportional to the change in the second fundamental form. Specifically, given two corresponding sets of point $\{(x_i, y_i)\}_i^N$ and $\{v_i\}_i^N$ there exists a height field mapping between the two, $f : \mathbb{R}^2 \to \mathbb{R}$. The bending energy corresponding to this mapping is proportional to the second order derivatives of the mapping:

$$E_{bend}(f) = \lambda \iint \left(\left(\frac{\delta^2 f}{\delta x^2} \right)^2 + 2 \left(\frac{\delta^2 f}{\delta xy} \right)^2 + \left(\frac{\delta^2 f}{\delta y^2} \right)^2 \right) \mathrm{d}x \, \mathrm{d}y, \tag{3.16}$$

where λ is a smoothing parameter which balances the quality of fit and the amount of bending, i.e. the 'wiggliness' of the function. TPS finds the transformation that fits the data while minimising the bending energy. Note that TPS may also be defined in terms of the radial basis functions that are used for smooth scattered data fitting. The RBF solution for thin plate splines is:

$$f(x, y) = \sum_{i=1}^{N} \omega_i \, \phi(\|(x, y) - (x_i, y_i)\|), \quad \text{where } \phi(r) = r^2 \log(r). \tag{3.17}$$

To ensure that the function f has square-integrable second derivatives, the following conditions are imposed:

$$\sum_{i=1}^{N} \omega_i = \sum_{i=1}^{N} \omega_i x_i = \sum_{i=1}^{N} \omega_i y_i = 0. \tag{3.18}$$

These conditions and the data fitting requirement may be combined into a linear system of equations:

$$\begin{bmatrix} K & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} \omega \\ o \end{bmatrix} = \begin{bmatrix} v \\ o \end{bmatrix}, \tag{3.19}$$

where K encodes the relation between the data points, i.e. $K_{ij} = \phi(\|(x_i, y_i) - (x_j, y_j)\|)$, $P_{i\cdot} = (1, x_i, y_i)$ contains the variables from Eq. 3.18, ω contains the values of ω_i, and v contains the target function values v_i. The variables 0 and o denote the zero matrix and the zero column vector respectively. Solving this linear system of equations gives the desired transformation in two dimensions. The method extends naturally to three-dimensional problems by including a dependence on an additional variable in the calculations described above. Radial basis functions are commonly used in the field of performance-driven animation [JTDP03].

3.6.2 Non-Rigid ICP

Iterative Closest Point (ICP) is an algorithm used to align two partially overlapping point clouds by minimising the square error between corresponding points. The quality of the results produced by this algorithm is sensitive to the initial guess at a solution, i.e. ICP only refines the initial estimation. See Alg. 1 for the outline of the algorithm. Mean square error algorithm is used to calculate the average of the squared errors between the target and the transformed source points. Thus the objective function is a function of rotations and translations. The output of the algorithm is a transformation matrix that provides the optimal mapping between the two point clouds within a given threshold. The transformation matrix may be split into rotation and translation. The algorithm solves a linear system of equations where the unknowns are the coefficients in the rotation matrix and the translation vector and the known point cloud values are used as coefficients. The method is often used to find a transformation between a point cloud and a surface; in that case the surface normals are used as additional input.

Algorithm 1: Iterative Closest Point.

```
Data: source point cloud
      target point cloud;
      initial guess;
      error threshold;
while error > threshold do
    for point in source do
        find closest point in target;
    end
    use mean squared error to find best transformation that aligns source to target ;
    apply transformations;
end
```

The original ICP algorithm is limited to rigid transformations, i.e. rotation and translation, which have only 6 degrees of freedom. However, in order to capture scaling, shearing and other more complicated transformation, an affine mapping should be used. Thus an extension of the ICP algorithm, called non-rigid ICP is used when additional degrees of freedom are present. Allen *et al*. proposed a non-rigid registration method that uses a numerically non-linear solver to find a smooth affine mapping [ACP03]. Additional constrained are imposed by manually matching some known marker locations in the two datasets. The method exhibits slow convergence and often leads to local minima; consequently, the authors use a multi-resolution approach where the optimisation is first performed on a low resolution mesh, and then optimised on the high resolution mesh. Amberg *et al*. combined the ICP algorithm with the method of Allen *et al*. to overcome the issues with convergence [ARV07].

3.6.3 Animation

The aim of this project is to used captured data of the actor's face to animate a given 3D model. We use the digital Emily model that comes with 68 controllable blendshapes ranging from simple eyebrow movements to complicated lip corner pulls [ARL*09], [Facb]. Each blendshape has a weight w_i associated with it, and $w_i \in [0, 1]$. The model also includes eyes and teeth. The challenge is to find a combination of these blendshapes that produce a specific facial expression. In particular, we are interested in reproducing the captured sequence of the actor's expressions so that the resulting facial animation appears realistic and life-like.

Initial Approach

Our initial approach was to first compute an estimate of the sparse sequence of expressions in the Emily (target) domain, and then use this to compute a dense facial animation for the Emily model.

First, we manually choose a sparse set of points on the Emily target mesh that correspond to the 97 marker points on sparse source mesh, as shown in Figure 3-15. Then the neutral expression of the subject is matched with the neutral expression of Emily by transforming the sparse source mesh to the sparse target mesh using TPS. The TPS transformation is stored, and it is used to map all of the frames in the captured sequence to the target domain. This produces an estimate of the sparse Emily sequence; a number of frames of which are shown in Figure 3-16, next to the corresponding sparse source meshes in each frame.

Using TPS again we compute the transformation in each frame that transforms the neutral Emily points to the positions of the estimated points in a particular frame. We then use the computed transformations to warp the all the vertices of the dense Emily mesh according to the positions of the sparse points, obtaining a dense Emily sequence. The resulting meshes for a number of frames displayed in Figure 3-17. The quality of the resulting animation is poor for a number of reasons. Firstly, the sparse points are not able to constrain the dense mesh, especially since parts of the face, for example the eyelids and the lips are not tracked in the sparse mesh. Consequently, numerous artefacts appear on the dense mesh, and they are particularly noticeable around the lips as no boundary conditions are imposed. Secondly, though there is a clear correspondence between the generated Emily sequence and the source sequence, the generated expressions look unnatural. This may be explained by the differences in the geometry and anatomy of the two faces; for instance, the source may be able to pull expressions that cannot be mimicked by the model. In addition, the errors in tracking, reconstruction, and manual selection of sparse points on Emily further reduce the quality of the animation.

In our initial attempt, we found that the mouth of the dense Emily mesh was unable to open when we applied the TPS transformation to each frame. Rather the mesh would simply be stretched when the control points around the mouth moved down. This is because the mouth of the Emily model is closed in its neutral expression, and TPS will not generate a gap for the mouth - it simply deforms the existing surface in a way that minimises the surface bending

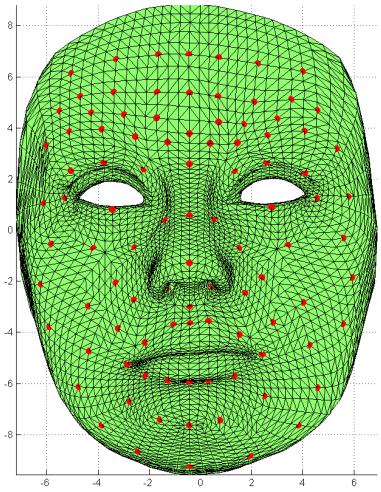


Figure 3-15: 97 points are manually chosen on the Emily mesh that correspond to the actor's mesh. The order of the points must be preserved between the two meshes.

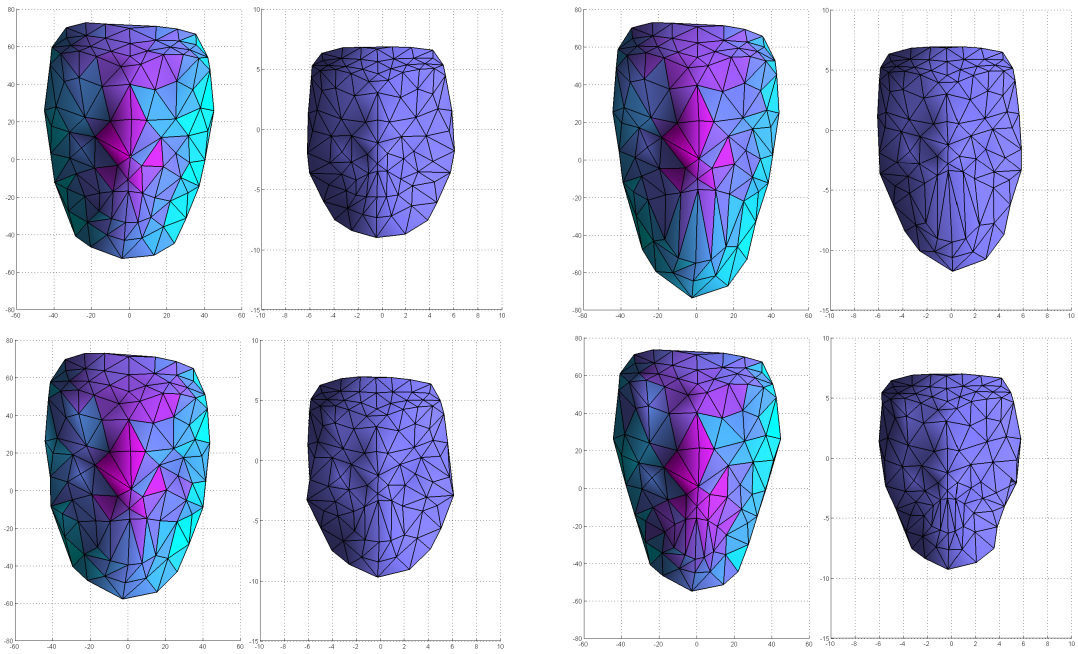


Figure 3-16: A sample of frames from the estimated Emily sequence. For each frame, the original motion capture data is on the left, and the corresponding Emily mesh computed using a thin plate spline transformation is on the right.

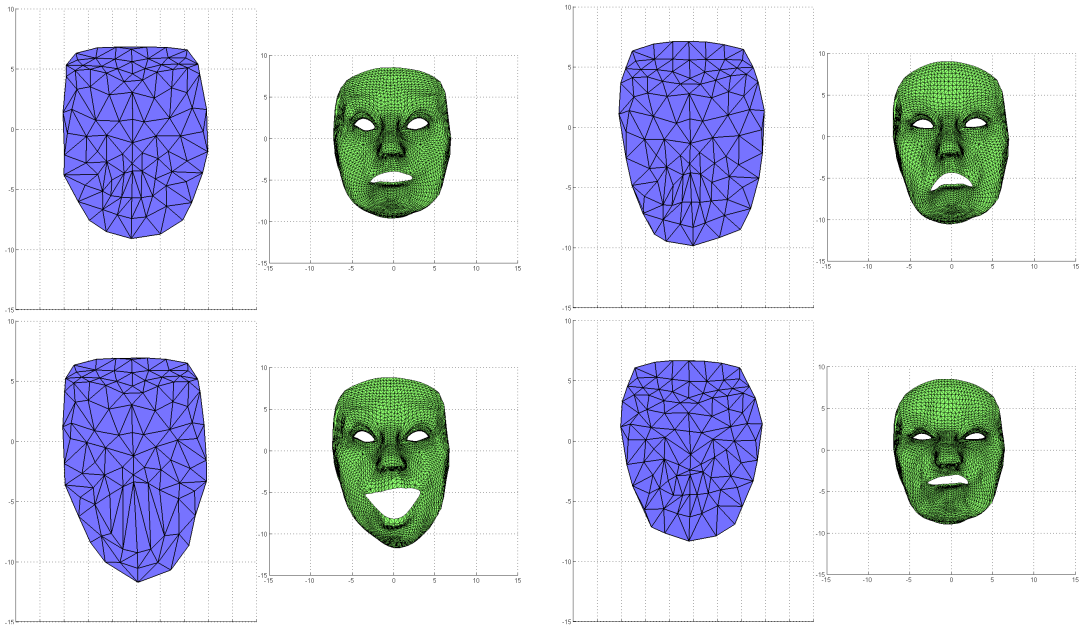


Figure 3-17: The shape of the dense Emily mesh is computed by applying a TPS transform to the neutral face mesh. A sample of frames from the resulting dense sequence are shown here. The sparse sequence points are shown on the left, and the corresponding dense meshes are shown on the right.

energy. To tackle this problem, and improve the resulting mesh animation, we had to simplify the dense Emily model by manually removing the part of the mesh that corresponds to the lips. Though the simpler mesh exhibited slightly better behaviour, i.e. there were fewer visible artefacts, but the sparse point cloud was still unable to constrain the dense mesh well enough to produce realistic results. Furthermore, the lack of lips in the mesh makes the resulting rendered facial animation appear very unnatural, see Figure 3-18.

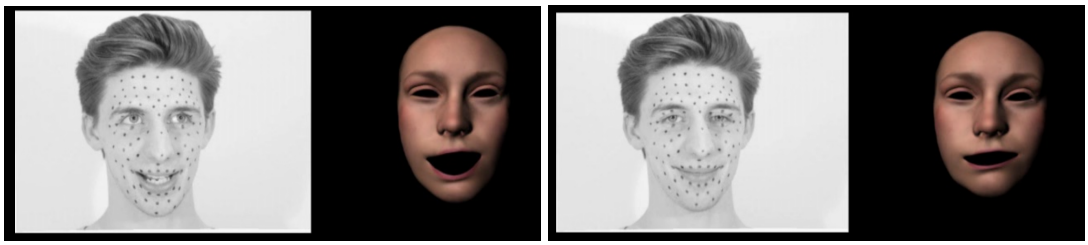


Figure 3-18: Our first animation produced by warping the actor's expressions at each frame of the sequence using TPS.

Introduction of Blendshapes

In an attempt to improve the visual quality of the generated animation, we decided to use the set of Emily key-shapes, or key-shapes, that were readily available to us. The main argument for using these blendshapes is that in most situations a simple linear relation exists between an arbitrary facial expression and the set of key-shapes. The shape may encode the entire geometry of the face, the displacement from the neutral face, or the local features of a face. If no such shapes are available, they may be produced using principal component analysis (PCA). Such a set of blendshapes is orthogonal, thus the features encoded by each shape are controlled independently; however, such shapes do not allow for intuitive facial transformations making it difficult for a VFX artist to perform manual post-process editing.

Let us define a set of blendshapes $\mathbf{B} = [\mathbf{b}_0, \ldots, \mathbf{b}_n]$ where $\mathbf{b}_0$ is the mesh of a neutral face, and each of the $\mathbf{b}_i$ correspond to a basic facial expression, for example the raise of an eyebrow. Unless stated otherwise, we shall use 68 shapes provided with the digital Emily model. Then a new facial expression $\mathbf{F}(\mathbf{w})$ may be constructed by finding an appropriate linear combination of the offsets of the basic shapes:

$$\mathbf{F}(\mathbf{w}) = \mathbf{b}_0 + \sum_{i=1}^{N} w_i \, |\mathbf{b}_i - \mathbf{b}_0|, \tag{3.20}$$

where $\mathbf{w} = [w_1, \ldots, w_n]$ is a set of weights that describe how much each of the basic shapes affect the new expression $\mathbf{F}(\mathbf{w})$. Typically, a convexity constraint is imposed on the weights; in our case the choice of constraints is influenced by the constraints present in our blendshape model, i.e. $w_i \in [0, 1]$. Eq. 3.20 may be written as a linear system of equations as follows:

$$\mathbf{F}(\mathbf{w}) - \mathbf{b}_0 = \hat{\mathbf{B}}\mathbf{w}, \tag{3.21}$$

where $\hat{\mathbf{B}}$ is the original set of blendshapes without the neutral expression.

A number of different numerical optimisation methods where tested when solving for the blendshape weights. An unconstrained solution may be calculated by inverting matrix $\hat{\mathbf{B}}$, and multiplying it from the left with the left hand side of Eq. 3.21. However, we found that this produced both positive and negative weights. Instead we use the Matlab®constrained non-negative least-squares solver (`lsqnonneg`) [lsqb] and constrained linear least-squares solver (`lsqlin`) [lsqa]. For the non-negative least-squares solver we formulate our problem as follows:

$$\min_{\mathbf{w}} ||\hat{\mathbf{B}}\mathbf{w} - (\mathbf{F}(\mathbf{w}) - \mathbf{b}_0)||_2^2, \; \mathbf{w}_i \geq 0, \; i = 1, \ldots, n. \tag{3.22}$$

The solver uses an active set method, and iteratively improves the active set of basis vectors; it converges in finite time. The same formulation is used for the linear least squares solver though it includes an additional upper bound on the weights in the function argument. This solver uses a reflexive Newton method which is able to accurately locate the local minimisers of large systems, and exhibits global convergence. It is also able to maintain sparsity of matrices but is generally slower.

Two-dimensional Sequence

Our initial tests of the different optimisation algorithms were carried out on a simpler single-view sequence of facial motion, as shown in Figure 3-19. Approximately 80 facial markers were used and their 2D positions were tracked through an image sequence to produce a sparse 2D motion sequence of expressions. Then a set of key frames were hand-picked to represent the extreme cases of the facial motion, for example in Figure Figure 3-19 we can see the neutral face, one eye-brow raise, then the other, etc. The sparse points from these frames are then used as blendshapes. The two solvers were tested on the entire motion sequence. We shall compare the results by computing the average error between the tracked markers in the original sequence, and estimated marker positions in the sequence from the computed blendshape weights.

Figures 3-20 and 3-21 illustrate the concept of blendshapes where we solve for the weights, using both aforementioned least square solvers, that best reproduce the expression shown in a frame from the image sequence. In Figure 3-20 we can see that the expression in the frame in question (frame 10) is very close to one of the blendshapes (frame 9), and therefore the weight for this blendshape is much higher than the others; the remaining blendshapes have weights close to zero. Both optimisation methods are able to represent this expression well; the average error associated with the constrained non-negative least-squares solver is less than 0.29, which corresponds to approximately 0.1 centimetres, while the average error for the second solver is less than 0.21 or approximately 0.09 centimetres. Note that although both methods use frame 9 with a large weight as expected, the other blendshapes are different for the two methods. This suggests that none of the blendshapes are really able to capture the subtle change in expression between frames 9 and 10. In comparison, Figure 3-21 shows the solve for the expression at frame 400. Frame 400 is twelve frames away from a frame that was chosen as a blendshape, resulting in an increase in average error for the two methods (approximately 0.49 in both cases). As expected, both methods use the two blendshapes that are the nearest to frame 400 in terms of timing. The weights used for these two blendshapes are the same to four decimal places, and the extra shape used by the first method has a low weight assigned to it. Though the second method is able to produce slightly more accurate results, at least for this data set the difference does not seem significant.

Principal Component Analysis

An alternative set of key shapes may be constructed by applying Principal Component Analysis (PCA) to the observed high-dimensional data. Generally, PCA is used to transform a set of correlated observations into a set of values of linearly uncorrelated principal components. If the number of principal components is smaller than the number of original variables, then the method may be used for dimensionality reduction. PCA tries to find a set of orthonormal axes which under projection preserve the highest variance. When formulated as an eigenvalue problem, it finds the eigenvectors of the covariance matrix with the largest eigenvalues.

PCA has been used in facial animation as an automatic way of constructing a set of blend-

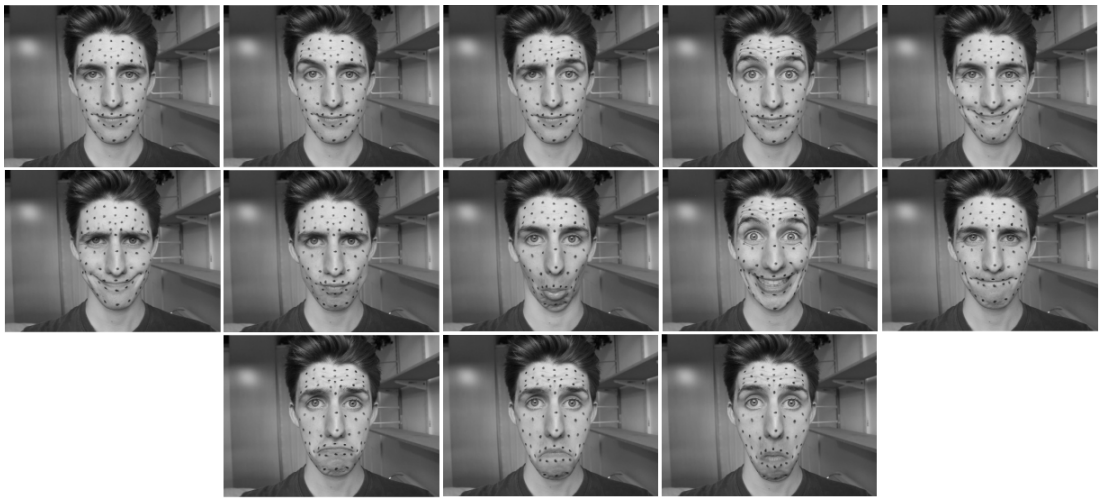


Figure 3-19: Set of facial expressions used as blendshapes. The first frame is used as a neutral expression.

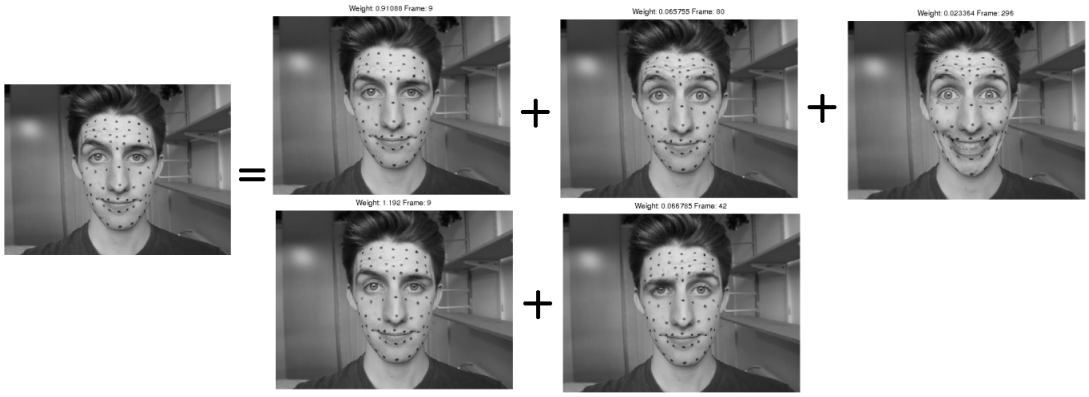


Figure 3-20: Frame 10 on the left is represented as a linear combination of the blendshapes, generated from key frames in the sequence. The top solution was obtained using the `lsqnonneg` solver while the bottom one was obtained using `lsqlin`.

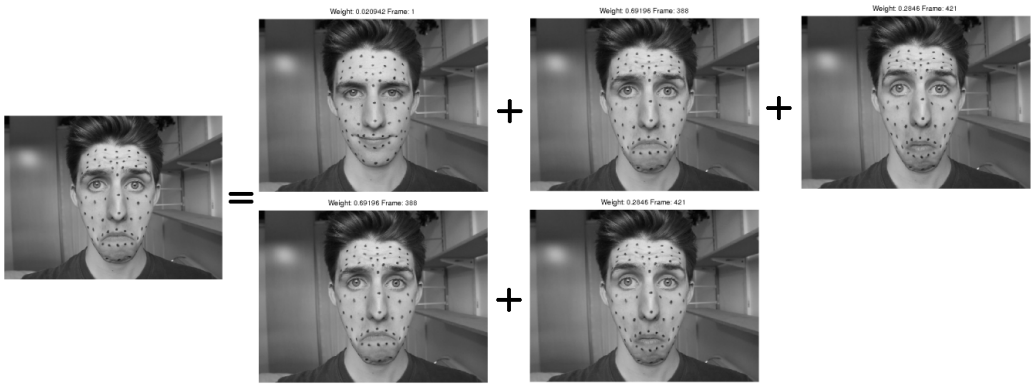


Figure 3-21: Frame 400 on the left is represented as a linear combination of the blendshapes. The top solution was obtained using the `lsqnonneg` solver while the bottom one was obtained using `lsqlin`.

shapes, see Section 2.1. The main advantages of using such blendshapes is that the construction does not require artistic skills, and the shapes are guaranteed to be orthogonal to each other. The second argument is particularly important in compression algorithms, i.e. when the aim is to find the smallest possible set of blendshapes that is able to represent the data while minimising the squared reconstruction error. We use the Matlab®implementation of PCA to find a set of PCA eigenvectors that represent the directions with highest variance in our data. Given this new set of shapes, we use the same optimisation algorithm as previously described, replacing the neutral expression with the mean of the data.

To compare the results with a different number of blendshapes, we calculate the average reconstruction error for a 500 frame sequence. When the first 3 shapes are used, the error is approximately 0.16 centimetres and 0.25 centimetres for the two optimisation methods respectively. Introduction of five additional shapes reduces the error by approximately 0.02 centimetres. Thus PCA does not outperform the model with manually designed set of key shapes. An additional drawback of the PCA basis is that it is not intuitive, i.e. the key shapes do not correspond to recognisable facial expressions. As a consequence, manual editing of the resulting animation is very complicated. Throughout the rest of the project we use the standard set of 68 blendshapes from the Digital Emily project.

Animation of a 3D Model

Next, we shall consider the animation of the previously introduced 3D Emily blendshape model using the reconstructed sparse 3D sequence of the actor. The captured sequence contains 731 frames and shows the actor talking for approximately 12 seconds. The corresponding sparse sequence in the Emily domain is obtained using a linear elasticity method, see Section 3.6.1. As in the two-dimensional case, at each frame of the sequence, the chosen optimisation method calculates the best possible combination of blendshape weights given the sparse Emily sequence. In particular, the input sequence is a sparse sequence of the motion that was trans-

formed from the capture space to the digital model space.

We start by manually marking the three-dimensional face of Emily; we try to replicate the pattern we used when capturing the actor's face, see Figure 3-3. The markers are placed at the vertices of the mesh, and the indices of these vertices are used to find the location of the corresponding markers on the meshes of the blendshapes, see Figure 3-22. Then the optimisation methods are used to extract the best linear combination of shapes together with the corresponding weights. The average reconstruction error for the non-negative least-squares solver is approximately 0.26 centimetres, compared to 0.23 centimetres for the constrained linear least-squares solver, see Figure 3-23. A number of factors may increase the error. Firstly, the expressions captured in Emily blendshapes are fairly mild, i.e. the shapes do not capture the most extreme motion of the face. Consequently, the estimated facial movements may not be expressed as a linear combination of the blendshapes. Secondly, as the blendshapes are not orthogonal, they may combine in unexpected ways, for example cancel each other out. This behaviour may be avoided if a set of PCA-produced shapes is used. Also, note that the mesh of Emily is not symmetrical, i.e. the left side of the face does not match the right side. This may lead to a sideways motion that cannot be captured by the blendshapes.

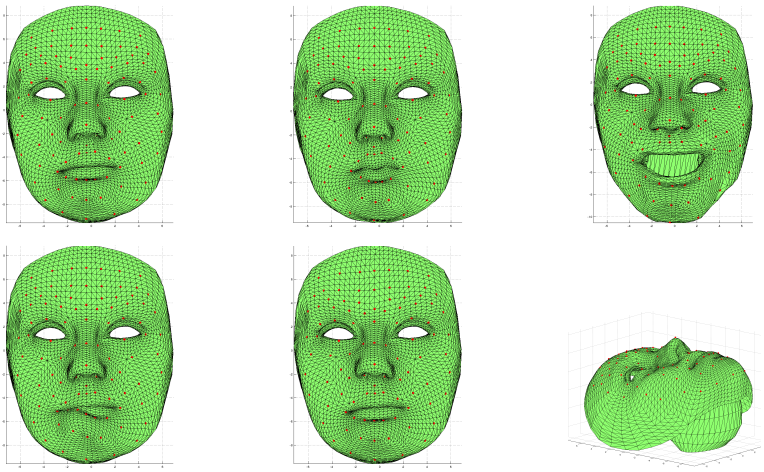


Figure 3-22: Examples of marked blendshapes.

The use of blendshapes greatly improves the visual quality of the resulting animation. However, the animation still lacks realism and expressiveness, thus we implement a number of changes to the original model. Firstly, we aim to find a sparse solution, i.e. minimise the number of blendshapes used at any given time. This is done by adding a penalty term on the $L1$-norm of the weights to the original minimisation problem. Then an iterative method is used to find the optimal set of weights; we use a Matlab®implementation of the LASSO algorithm [Sch05]. After careful adjustment of the scaling parameter, the least squares solver is able to reduce the mean reconstruction error by approximately 0.03 centimetres. However, the best sparse results were achieved without imposing the upper and lower bound on the weights. We found that in our case it is more important to impose these bounds than to favour sparse solutions. Moreover, the solution produced by the constrained non-negative least squares solver

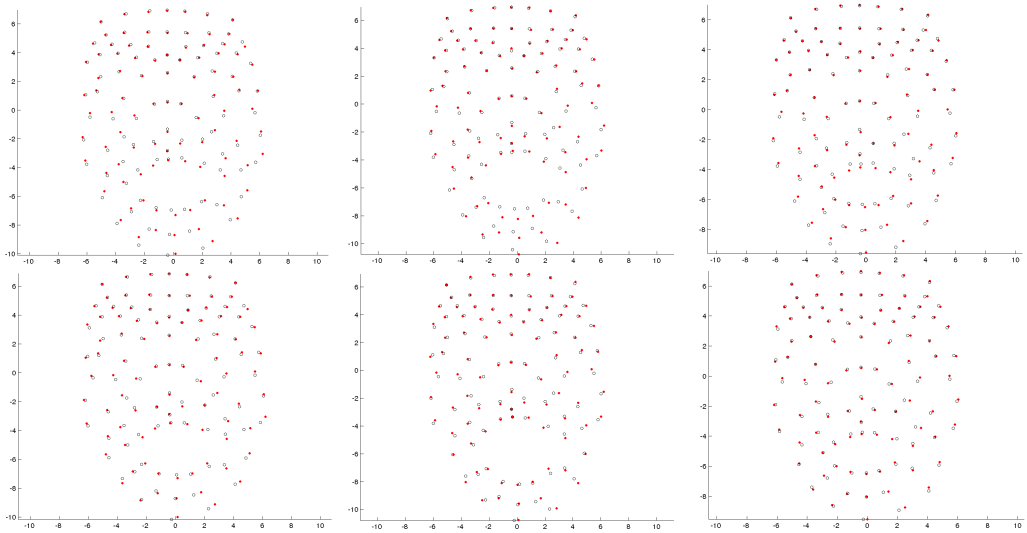


Figure 3-23: The red dots mark the position of the markers in the motion sequence while the black cirles correspond to the reconstructed position of the markers. The top results were produced using the non-negative least-squares solver while the bottom results were produced by the linear least-squares solver.

uses on average 21 blendshapes per frame, which does not seem unreasonable given the complexity of human facial motion.

As noted by Bouaziz *et al.*, a motion capture sequence also has temporal constraints, i.e. each frame is closely related to the previous frames [BWP13]. This constraint may be included in the minimisation problem by adding a smoothness term, $\|\mathbf{w}^{t-2} - 2\mathbf{w}^{t-1} + \mathbf{w}^{t}\|_2^2$, where $\mathbf{w}$ is the vector of weights, and t is the frame number. The resulting problem is generally hard to solve, but there exist a number of algorithms that minimise the sum of Euclidean norms. Beside the one mentioned in the paper by Bouaziz *et al.*, we have considered the algorithms proposed by Andersen *et al.* [ACCO00], and Xue and Ye [XY97]. However, after examining the solution produced by our current methods, we noticed that it automatically obeys this smoothness constraint; this may be explained by the fact that the input sequence was filmed at a relatively high frame rate (60 frames per second). Specifically, we compared the change in blendshape weights from frame to frame; the average difference is 0.007, which, when reconstructed, is hardy noticeable. The highest differences (of around 0.3) appear during the opening and closing of the mouth; these transitions look natural at the original frame rate. We note that if the motion capture data was recorded at a low frame rate, then the implementation of the temporal constraints might improve the quality of the resulting animation.

Next, we examined the effect of using a reduced set of key expressions in the solver; this leads to a smaller system of equations to be optimised. The reduced set contains 19 shapes included in the visual set of controls in the original blendshape model. The smaller set of shapes leads to a slight increase (of approximately 0.03 centimetres) in reconstruction error. There is no noticeable effect on the visual quality of the resulting animation, see Figure 3-24. Thus the only notable advantage of using fewer blendshapes is the acceleration of the

optimisation step. However, if the performance of the algorithm is the main objective, then a PCA basis is a better choice than a reduced non-orthogonal set of shapes.

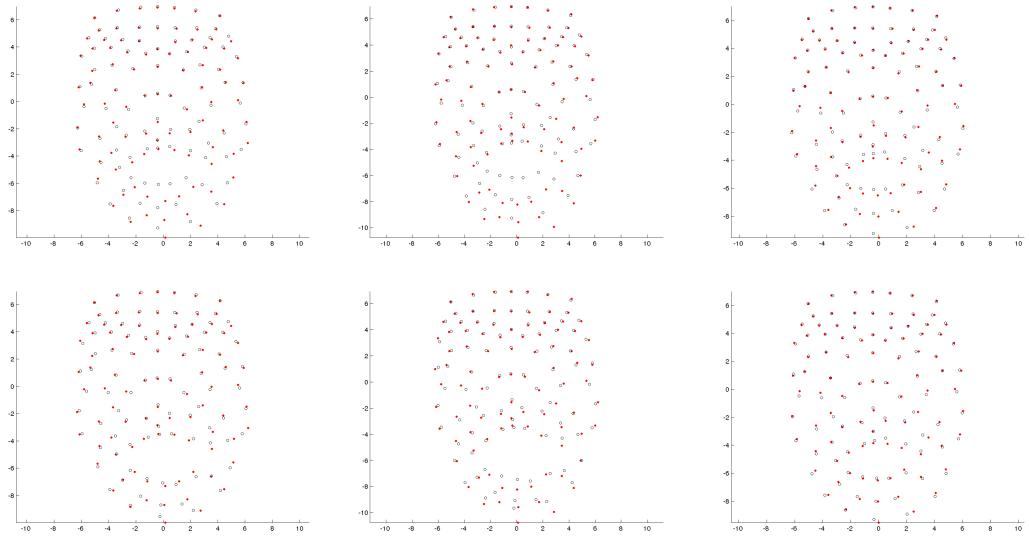


Figure 3-24: The reconstruction error for a model with 19 blendshapes. The top results were produced using the non-negative least-squares solver while the bottom results were produced by the linear least-squares solver. Colours as in Figure 3-23.

For comparison, we attempt the opposite approach; we include extra blendshapes to account for the features that are not present in the original set. This approach was inspired by the research of Li *et al.*, who introduced corrective shapes to systematically extend the initial set of shapes [LYYB13]. The authors start by using linear combinations of PCA principal components to construct new facial expressions. They then extract a number of samples that cannot be explained by the original set of principal components. An iterative procedure is used to produce corrective shapes that are orthogonal to the members of the original set, and that improve the fitting accuracy. Unfortunately, such corrective method cannot be applied to a manually constructed set of blendshapes since it relies on vector orthogonality. Instead, we manually construct a set of new shapes that are more extreme than our original blendshapes; we then include these new shapes in the solver. The criteria for choosing the new expressions are, (a) the likelihood of that or similar expressions, and (b) the dissimilarity of the new expressions in comparison to the existing ones. The four extra blendshapes are shown in Figure 3-25; they were constructed by increasing the weights on the original blendshapes beyond the predefined limit. Figure 3-26 shows the activation of the extra blendshapes throughout the motion sequence. Both of the optimisation methods favour the first additional blendshape that corresponds to the smile in Figure 3-25 while the other shapes are only used with low weights. Despite the extension to the set of shapes, the reconstruction error decreased by less then 0.01 centimetres for both methods. This is explained by the fact that the extra blenshapes are closely related to the original ones, as they are in fact a linear combination of the original blendshapes. A more systematic approach needs to be used to produce more diverse shapes. For example, an additional optimisation step may be included that randomly alters the existing blendshapes,

Figure 3-25: Additional blendshapes.

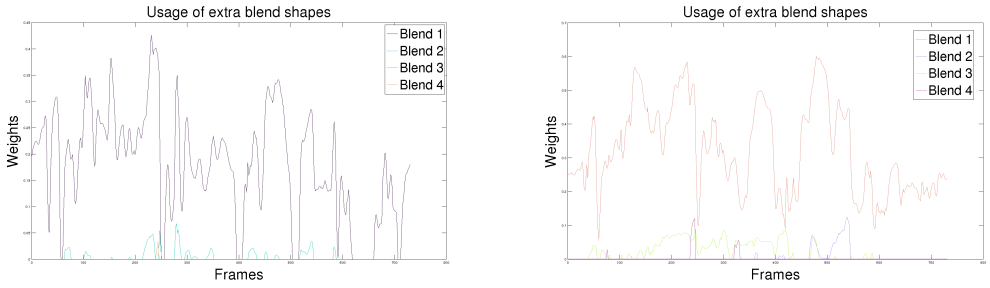


Figure 3-26: The weights associated with the extra blendshapes as a function of frames in the sequence. The left diagram corresponds to the constrained non-negative least-squares solver while the right one gives he results for the linear least-squares solver.

and uses a small subset of the altered shapes to find the extension of the blendshape set that minimises the reconstruction error. In particular, we choose a set of new shapes that are constructed by adding a small random quantity to the average of the existing set. Then these shapes are included in the optimisation algorithm. The new shapes are stored if they reduce the reconstruction error, and new blendshapes are suggested until the error decreases below a chosen threshold. Using this iterative procedure, we are able to reduce the reconstruction error by about 10%; however, this approach is very computationally intensive, and the new shapes cannot be easily included in the existing model.

Estimation of Weights in Actor's Domain

At this stage our best results still lack expressiveness and visual appeal. In the current pipeline, we first transform the motion sequence from the actor to Emily, and then solve for the blendshapes in the Emily domain. This approach relies on the transformation to produce valid motion in the Emily domain; however, the transformation is geometrical, and it does not guarantee to produce a realistic and natural-looking animation. Therefore, we test the opposite approach. We first find the thin plate spline transformation $\mathbf{T}$ from the actor's neutral expression to Emily's neutral expression. Then the inverse transformation $\mathbf{T^{-1}}$ is used to map the Emily blendshapes to the actor's domain. We thus have matching sets of sparse blendshapes in both domains, see Fig 3-27. The weights are calculated at each frame using the original

captured sequence and the newly designed blendshapes in the actor's domain; these weights are then applied directly in the Emily domain.

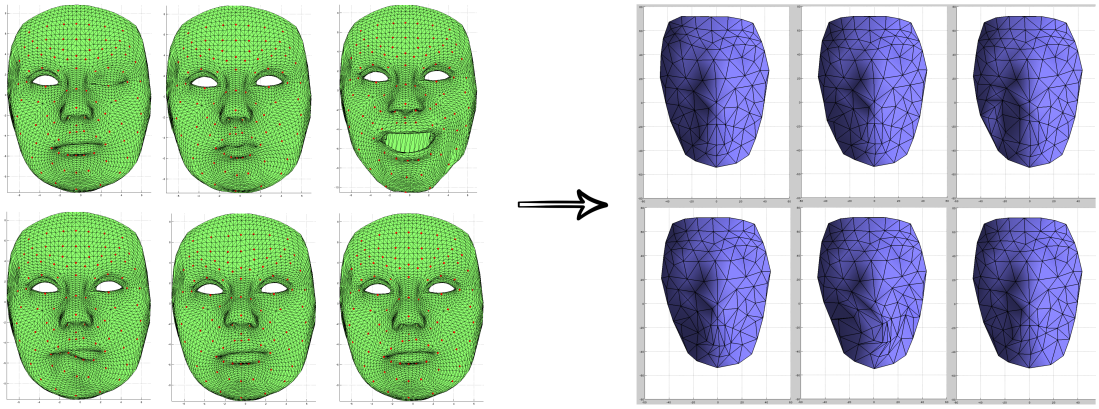


Figure 3-27: We use the inverse thin plate spline transform $\mathbf{T}^{-1}$ to produce a set of blendshapes in the actor's domain. The left images are the sparse Emily blendshapes (red dots) while the right images are the corresponding actor's sparse blendshapes (vertices of the mesh) generated from the existing Emily blendshapes.

Figure 3-28 compares the results produced using the two antipode approaches. The new approach is able to better mimic the motion of the actor, and it improves the appearance of the digital model, i.e. the new animation appears more natural and realistic. Note that the original method tries to reproduce the actor's performance using the transformed geometrical data while the new method relies more on the semantics associated with the blendshapes. In particular, we do not expect the actor and Emily expressions to match identically, for example one of them might be able to raise the eye-brows higher than the other. The new approach is better at respecting and enhancing the different features of the two faces. The mean reconstruction error is smaller than 0.1 centimetres which is an improvement over the previous methods. We test our new approach using another motion capture sequence; in the new sequence the actor pulls the expressions listed in the Facial Action Coding System [FACc]. This sequence is more challenging as the expressions are diverse and intricate as opposed to the rather monotonous motion in the talking sequence.

We note that our final animation was not always able to capture the slight smiles that are noticeable in the recorded sequence. This may be explained by the poor match of neutral expressions. Specifically, all of the methods that we used rely on the transformation between the neutral faces. However, the actor's neutral expression is more positive (i.e. the corners of the lips are raised) than that of Emily, see Figure 3-29. This may be fixed by choosing a better matching neutral face from the actor's sequence, or by mimicking the actor's expression on Emily, and using that as her neutral. In our final results we use the second approach, and apply an additional small positive weight to the blendshapes that correspond to a slight smiling face, see Section 4.2.

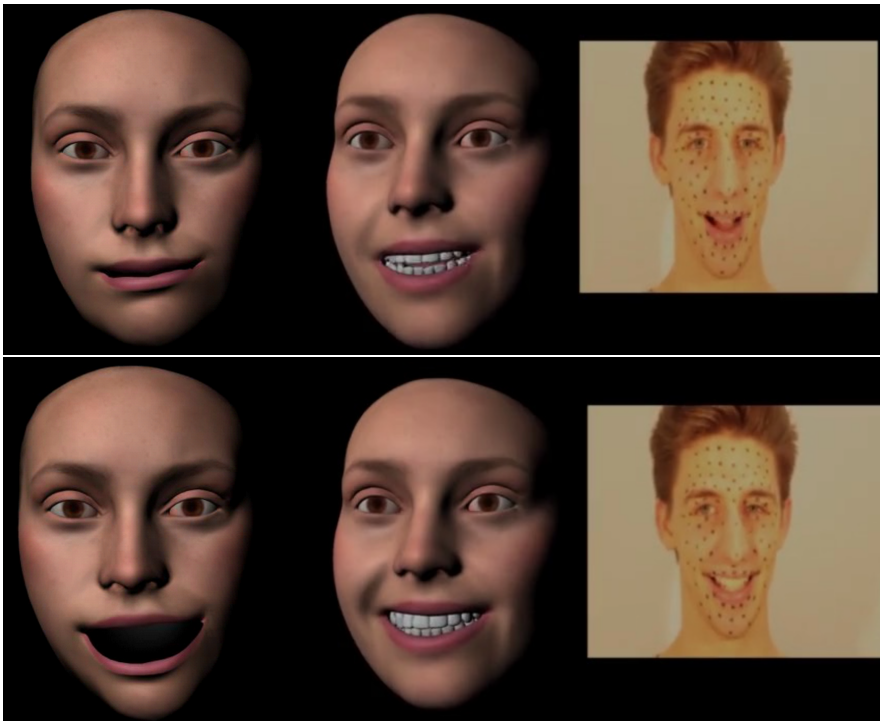


Figure 3-28: The expressions on the left were produced using our original approach while the expressions in the middle use the weights that were calculated in the actor's domain.

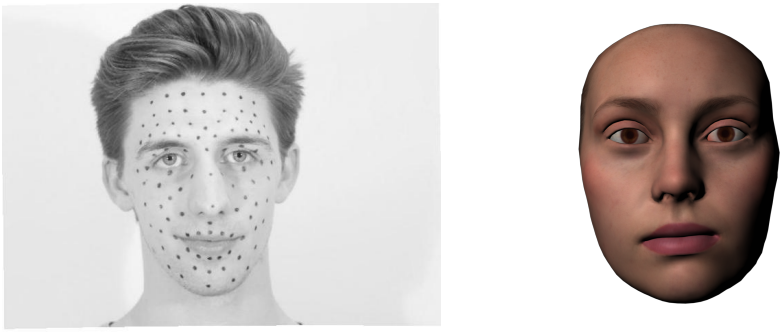


Figure 3-29: A comparison of neutral expressions for the actor and Emily. The two facial expressions don't match perfectly, and so errors due to this mis-match are propagated through the blendshape animation.

3.7 Skin Rendering

Our objective in skin rendering is to generate a face that would be indistinguishable from a real one. In our case, we have have taken a 3D scan of a subject and our aim is to improve the realism when rendering it. For this task we will mainly look at the techniques presented by Hertzmann *et al.* [HJO*01] and Graham *et al.* [GTB*13].

Algorithm 2: Image Analogies

Data: A unfiltered example, A' filtered example, B unfiltered source, L number of levels, k coherence parameter, t neighbourhood size.
Result: B' filtered source image.
Compute Gaussian pyramids for A, A' and B;
Compute features for A, A' and B;
Build k-d tree for $\{A, A'\}$;
for $l = 0$ *to* L **do**
  for *each pixel* $q \in B'_l$ **do**
    p_{app} = ANN search of q neighbourhood from $\{B, B'\}$;
    $r^* = \arg\min_{r \in N(q)} \|F_l(s(r) + q - r) - F_l(q)\|^2$;
    $p_{coh} = s(r^*) + (q - r^*)$;
    $d_{app} = \left\|F_l(p_{app}) - F_l(q)\right\|^2$;
    $d_{coh} = \|F_l(p_{coh}) - F_l(q)\|^2$;
    if $d_{coh} < d_{app}(1 + k2^{l-L})$ **then**
      $p = p_{coh}$
    else
      $p = p_{app}$
    end
    $B'_l(q) = A'_l(p)$;
    $s_l(q) = p$;
  end
end
return B'_L

Before we begin, lets explain the Image Analogies framework [HJO*01] in more detail. Given three images A, A' and B, where A is an unfiltered example, A' is a filtered example, and B is an input image, the algorithm will generate an output image B' such that B' relates in the same way to B, as A' does to A. A k-d tree for an Approximate Nearest-Neighbour Search (ANN) is built using a feature vector from a neighbour pixel p in A and A'. The closest match for a neighbourhood in pixel q in B and B' is located in the tree. A detailed description of the algorithm in pseudo code is shown in Algorithm 2, where F is a computed weighted distance over the feature vectors using a Gaussian kernel and s is a data structure such that $s(q) = p$. Based on Ashikhmin's work, a match that is coherent to what has been already synthesized is computed. These two candidates are weighted and the best one is chosen. The whole process is carried in a multiresolution pyramid, as shown in Figure 3-30, where l indicates the current level, in essence this means that the neighbourhoods also include the previous level

in the search. We found three Image Analogies implementations available [ImAa], [ImAb], and [ImAc]. The first one is a simple single threaded implementation, the second one was done with CUDA, however the author's single threaded code produced overall better results.

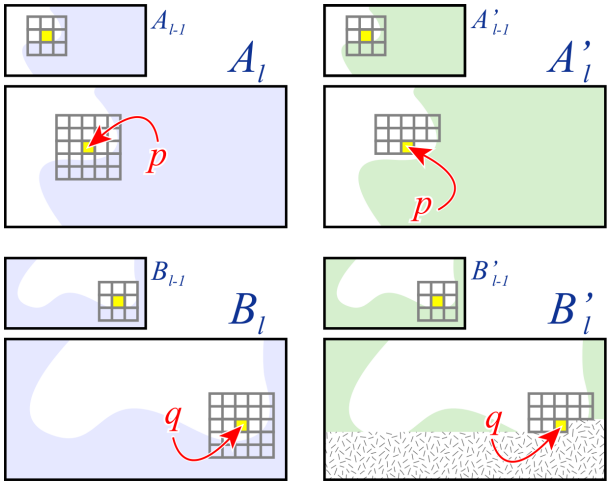


Figure 3-30: Neighbourhood matching for the Image Analogies framework, image taken from [Ash01].

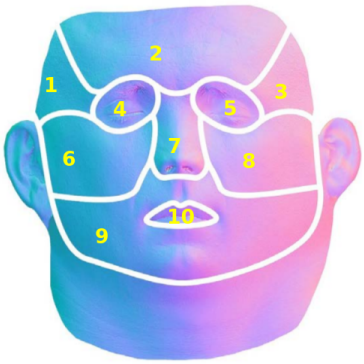


Figure 3-31: Texture segmentation in common coloured areas, image taken from [GTB*13].

As a first step, we tried to reproduce the results for bump mapping quality increase by Graham *et al*. [GTB*13], results are shown in Figure 3-32. The authors add an extra parameter $\alpha \in \{0, \ldots, 1\}$ to control Hertzmann's image synthesis process. To be more precise, a match between A and $\{B, B'\}$ will be weighted by $1 - \alpha$, and a match between A' and $\{B, B'\}$ will be weighted by α. The logic behind this addition is to encourage more details of A' to be included in B'. The modifications required to include this addition begin by building two separated k-d trees for A and A', when choosing the best match both distances will be weighted as explained above, and the pixel with the smaller distance will be chosen. Also in the coherence match two searches will be done and weighted accordingly, and the final pixel will be chosen without further adjustments.

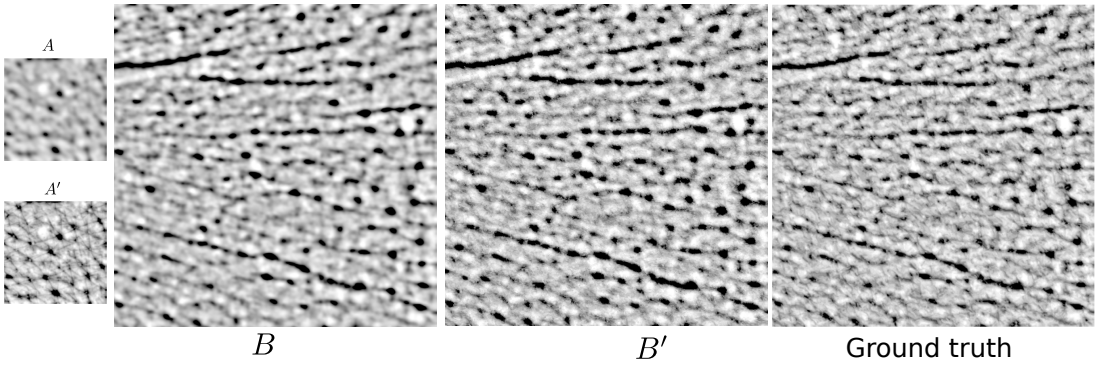


Figure 3-32: Bum map deblurring, A, A', B and ground truth images images taken from [GTB*13].

We also tried applying the Image Analogies filter to generate increased quality textures of skin, which is in essence a deblurring filter. The idea is to improve a low quality texture from a 3D scan using pictures of the texture taken at a closer range. To achieve this we took a close-up high quality photograph of the skin A' using a DSLR fitted with a macro lens. To generate the low quality texture A, the sample A' was blurred using a Gaussian kernel until it looked of similar quality to the 3D scan texture B. With these three inputs we generated an image B' of the skin at higher quality. Since faces have differentiated areas, this process was done separately for each relevant section of the face in the texture space, and the generated patches were stitched together using linear interpolation. The texture segmentation is shown in Figure 3-31 and results are shown in Figure 3-33.

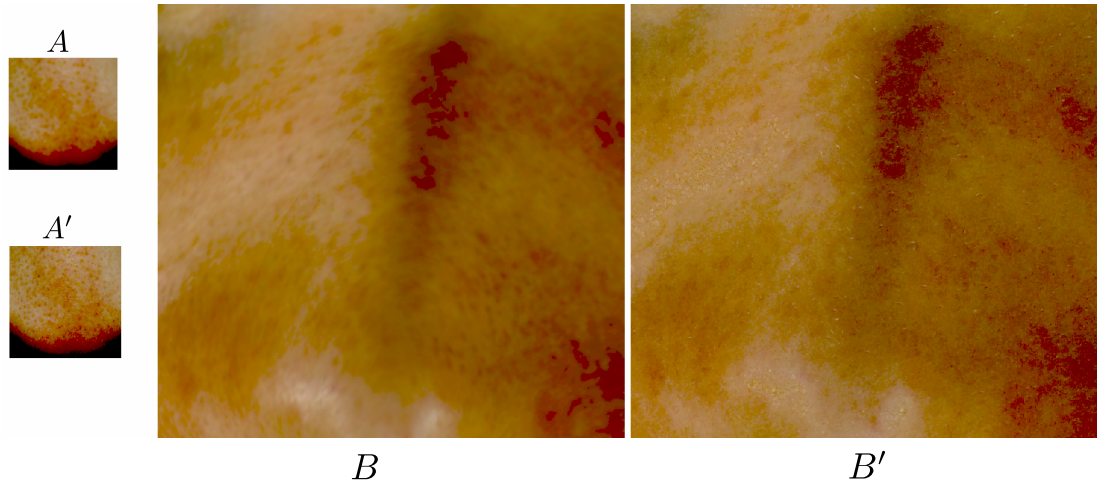


Figure 3-33: A skin texture taken from a segment of the 3D scan of the actor's face is shown in image B. An increased quality image B' is generated using the Image Analogies algorithm. The texture is false-coloured to highlight the differences.

Another option for increasing the quality of the texture is to apply image super-resolution techniques, we used Jianchao *et al.* [YWHM10] work for this purpose. The idea is that by doubling the resolution of the original texture, yet avoiding blur by adding information from a dictionary of high resolution images, the final rendering quality of the face will increase. Results for this approach are shown in Figure 3-34.

Generating normals maps using Image Analogies is another interesting area, as it could provide an alternative to the costly standard capture methods. We synthesised normal maps from albedo images and also from bump maps generated from the previous 3D scan texture, results are shown in Figure 3-36. To create the bump maps, the textures were converted to gray scale and a histogram equalisation was applied. In order to improve the quality of the bump maps, we also applied the Image Analogies filter to them using a known good bump map as a filtered example, results are shown in Figure 3-37.

The advantages of using a bump mapping are shown in Figure 3-38. In this example, a render with the original texture is shown in Figure 3-38a. The original RGB texture was converted to a gray scale image and applied directly as a bump map texture, results are shown

Figure 3-34: Super resolution example. The top left image is the original texture and top right is the 3D face model rendered with the original texture. Bottom left is the super-resolution texture, and the bottom right is the face rendered with the super-resolution texture. Original data from [Faca].

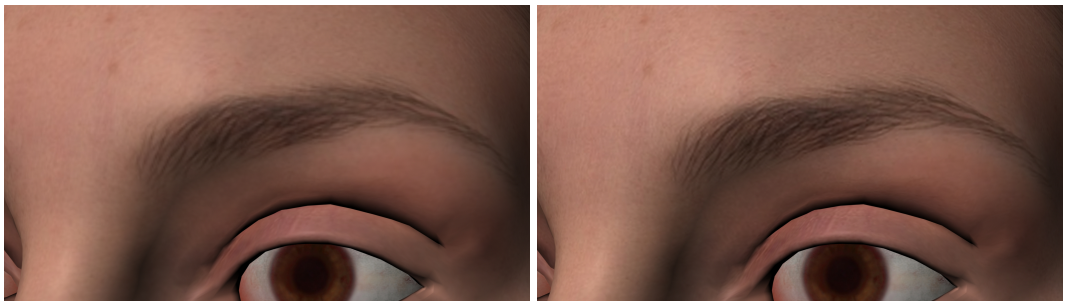


Figure 3-35: Close up of texture super-resolution for Emily, (left) original texture, (right) synthesized texture with higher resolution.

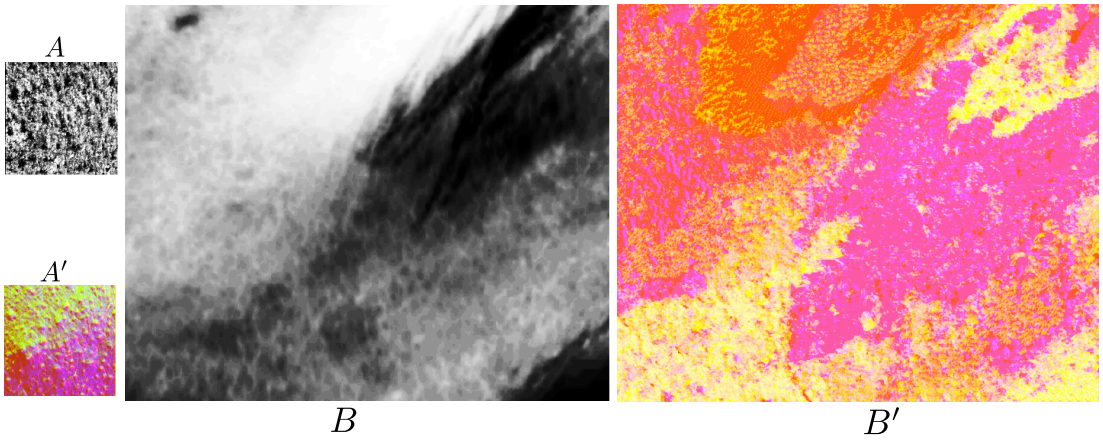


Figure 3-36: Normal synthesis from albedo image.

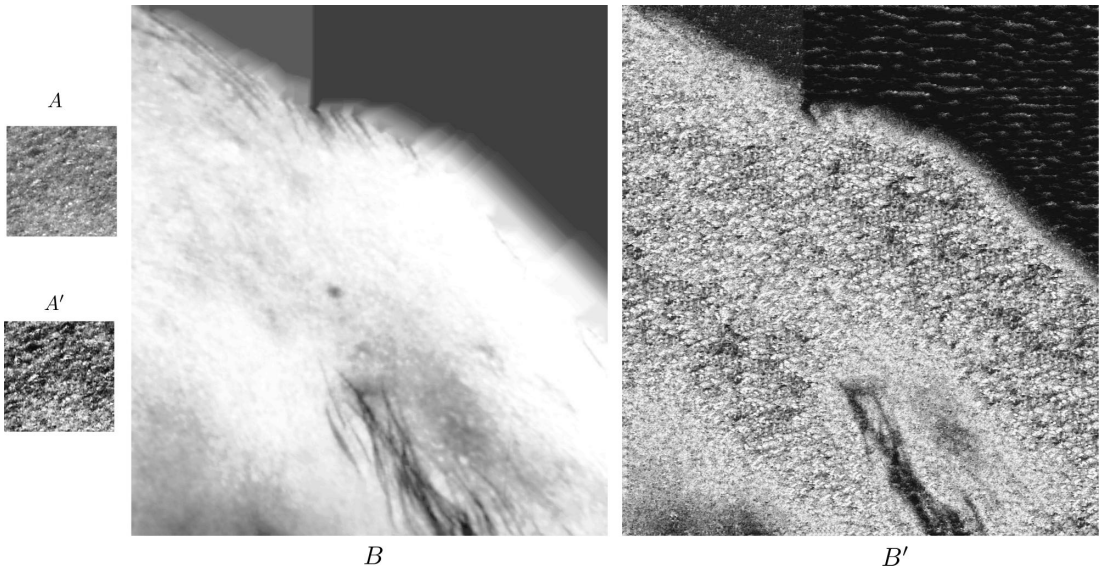
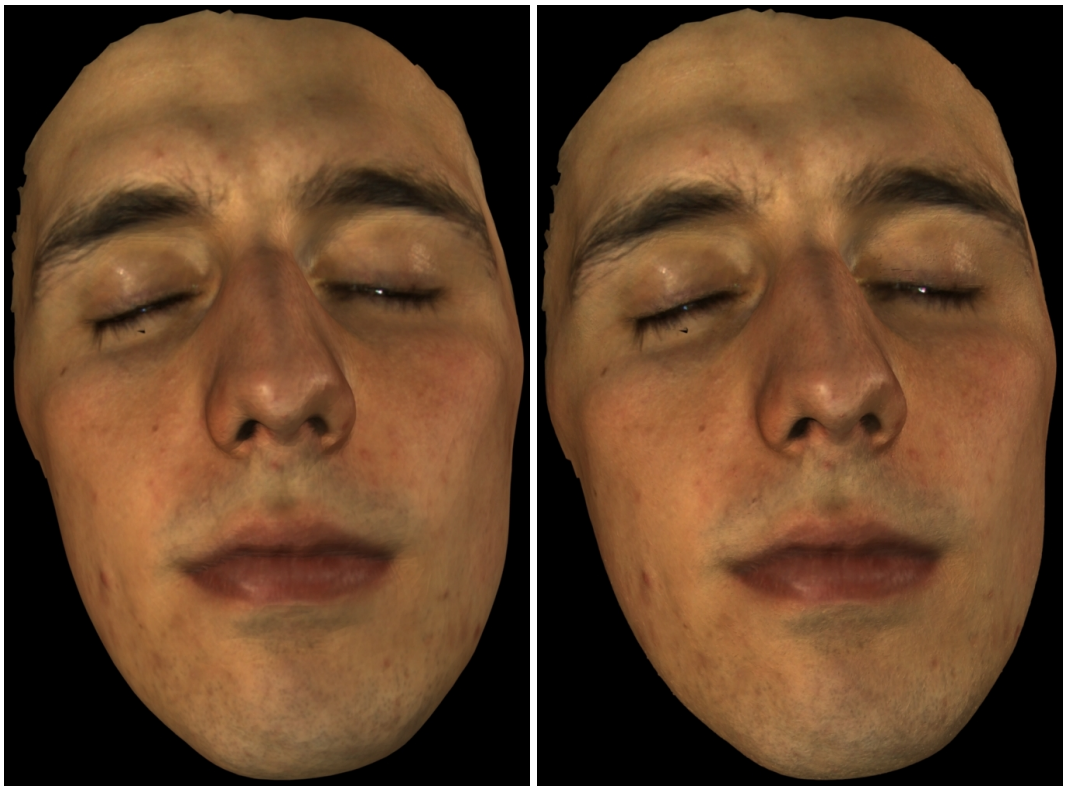


Figure 3-37: Improving the quality of a bump map from a gray scale texture.



(a) Render with the original texture.

(b) Render with the original texture and bump mapping.

Figure 3-38: The advantages of using a bump map.

in Figure 3-38b, note how the addition of high frequency imperfections adds extra realism to the image.

Chapter 4

Implementation details

This chapter aims to provide some insight into the technicalities in our development process, and to highlight the issues that we had to solve in order to get the final results.

4.1 Matlab®Processing

Once we had acquired motion capture data of the actor's performance, we used Matlab®for the majority of the post-processing work before rendering the final output animation in Maya®using our own C++ plugins. All the stereo geometry processing was done in Matlab®, namely calibration of the stereo camera system, estimating the projection matrices and epipolar geometry, rectifying the image sequences, detecting and tracking the face markers, and finally computing the sparse 3D reconstructions of the facial expressions in each frame. Results from the tracking process, such as the tracking of the pupil movements and rigid head motion estimation, was output as a text file and read into C++ so as to be applied to the Emily model using the Maya®pugin. The solve for blendshape weights was also carried out in Matlab®, and the final weights were exported as a text file which could then be read into C++.

4.2 Maya®Plugin

We used blendshape rigs for Maya®that are provided free by [Facb]. The first task was to reset the paths of the texture files for all the materials in the Hypershade window in Maya®. The next step was to write a plugin to load the weights from a text file exported from Matlab®, and set keyframes for each of the blendshapes - to do this our plugin created a new Maya®command that could be easily executed. The work flow for writing the code was to first perform the action in Maya®manually if possible, see which MEL commands were executed, then implement them in the C++ plugin, and if possible rewrite them as pure C++ code instead of MEL commands. Our code complies with Maya®'s undo command policy, which means using the

DagModifier class or including extra code to undo the actions. Nevertheless, in the final stages of the implementation, it was faster to reload the scene than to undo the command.

The text file format we use for the weights is one blendshape weight value per line in the order of the sequence of frames, as shown in Figure 4-1, where N is the number of weights and F is the number of frames.

$$\begin{bmatrix} w_{00} \\ w_{10} \\ \vdots \\ w_{ij} \\ \vdots \\ w_{NF} \end{bmatrix}$$

Figure 4-1: The text file format used to store the blendshape weights for a given image sequence - used for exporting from Matlab®and importing into Maya®.

After reading in the weights from the text file, the connections between the blendshape object weights and their outputs have to be broken. This is done in order to be able to set the keyframes and update the blendshape values, otherwise the MEL command setKeyframe would not work. An extract of the code is shown in Listing 4.1. A minor point to highlight in this section is that the blendshape weights can be accessed using the following command, *"shapesBS.weight[i]"*, however to break the connections the blendshape name in Maya®is needed. To obtain a list with all the blendshape names the following command can be used, *"listAttr -m shapesBS.w"*. The weights will be listed in alphabetical order, yet the indices do not strictly follow the same order.

Listing 4.1: Breaking the weights connections and setting keyframes.

```
// Break connections.
for (unsigned int i = 0; i < (unsigned int)(numWeights); i++){
                cmd = "disconnectAttr shapesBS_";
                cmd = cmd + names[i];
                cmd = cmd + ".output shapesBS.";
                cmd = cmd + names[i];
                dgMod.commandToExecute(cmd);
}

// Set key frames
for (unsigned int j = 0; j < weights.at(i).size(); j++){
                cmd = "setAttr shapesBS.weight[";
                cmd = cmd + j;
                cmd = cmd + "] ";
                cmd = cmd + weights.at(i).at(j);
                dgMod.commandToExecute(cmd);

                cmd = "setKeyframe { \"shapesBS.w[";
                cmd = cmd + j;
                cmd = cmd + "]\" }";
                dgMod.commandToExecute(cmd);
        }
}
```

Once the blendshape weights for all frames in the sequence were added, we noticed that the teeth and tongue of the Emily rig would not move. Both meshes were actually regulated by the position of a control object in the scene, so the solution was to translate the control object by a factor of the mean of all the blendshape weights involved in the control of the mouth, as shown in Listing 4.2. The lower teeth and tongue would be left hanging in mid air when the mouth opens if the blendshapes that control the mouth opening were activated together; this problem was fixed by lowering the position of both meshes.

Listing 4.2: Teeth and tongue control based on relevant weights.

```
cmd = "setAttr con_jaw_c.translateY ";
cmd += -3.2 * (weights.at(i).at(58) + weights.at(i).at(55)) + 1;
dgMod.commandToExecute(cmd);
cmd = "setKeyframe  \"con_jaw_c.translateY\"";
dgMod.commandToExecute(cmd);
```

In order to make the animation more life-like, we decided to incorporate the removed rigid head rotation and translation back into the final animation. In order to achieve this, we saved the inverse rotation for each frame from the Procustes analysis that was performed in Section 3.5.1 into a file with the same format as shown in Figure 4-1. We then read the data into Maya®using the plugin, and applied the transformation for each frame; for the translation we did an equivalent procedure with a translation file. Note that Matlab®matrices are column-wise while Maya®matrices are row-wise, so we read them into Maya®taking this into account. The matrix multiplication order also plays an important role here, as shown in Equation 4.1, where to achieve the correct resulting head pose, we need to undo the translation first and rotate the result.

$$\mathbf{x}_{new} = \mathbf{R}\mathbf{x} + \mathbf{t} \;\rightarrow \mathbf{R}^{-1}(\mathbf{x}_{new} - \mathbf{t}) = \mathbf{x}, \tag{4.1}$$

where $\mathbf{x}$ is the original 3D point, $\mathbf{x}_{new}$ is the transformed point as a result of the Procrustes alignment, $\mathbf{R}$ is the rotation matrix and $\mathbf{t}$ is a translation vector. We implemented this transformation in Maya®using the "xform -m" command, first applying the translation and then applying the inverse rotation with an extra parameter "-r" so that the matrices get multiplied in the desired order.

Having eye movement is quite important to achieve realism in the facial animation, so for this purpose we tracked the 3D world coordinate positions of the actor's pupils in the input image sequence. These positions were saved into separate files for each eye using the same file format shown in Figure 4-1. The positions were loaded into Maya®and the eye control object was translated using the offsets with respect to the first frame, which was assumed to be looking straight forward. With this first approach, each eye can move independently which can lead to unsatisfactory results when one eye goes slightly off track and the other eye remains looking in the correct direction. An initial solution to this problem was to move both eyes together using the mean offset for each frame, so that their line of slight would always be parallel. This gave much better looking results, but nevertheless, this would still introduce considerable error when one of the eyes winks - since the tracking becomes completely unreliable for the winking

eye which in turn drags the other eye with it. Our solution involves interpolating between the two offsets using variable factors. The idea is to have both eyes moving together, start ignoring the offset for an eye if it goes off, while sustaining smooth movements. The final offset d_f will be computed as $d_f = \alpha d_l + \beta d_r$, where α and β are the interpolation factors, d_l and d_r are the offsets for the left and right eye, respectively. Figure 4-2 shows the α_x values along the x-direction, where t is the change threshold and l is the maximum offset limit. Initially α_x is 0.5, however if the offset goes beyond a threshold t, α will decrease linearly until it reaches the limit l. The same criteria will be applied in the y-direction, and the value for that eye will be $\alpha = \min\{\alpha_x, \alpha_y\}$. The β factor will be computed as $\beta = 1 - \alpha$, however, if the right eye is the one surpassing the threshold t, then the whole process will be reversed. Additionally, if both eyes surpass the threshold t, α and β will be 0.5, this will allow the eyes to move together beyond the threshold if needed.

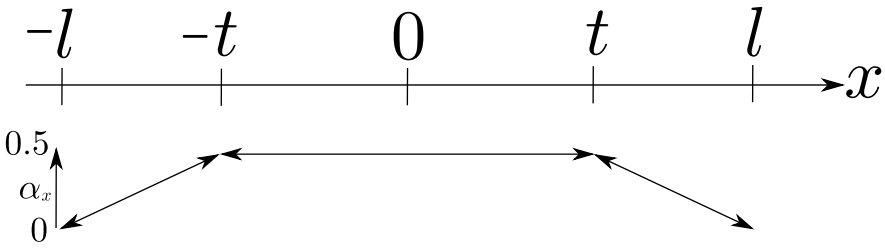


Figure 4-2: The offset interpolation for eye movements in Maya®. Here we are only considering the x-component of the pupil displacement, where 0 means the eye is looking straight forwards, and l and $-l$ are the maximum and minimum pupil displacements respectively.

Incorporating blinks was another important step towards attaining a more realistic animation. Since the eyelids were not tracked in the recording session, the only option was to manually set the blinking times in Maya®. To faithfully reproduce the blinking of the actor in the captured sequence the following information was manually added:

- Frame number marking when the eye:
  - started closing;
  - closed completely;
  - started opening again;
  - was fully open again;
- The blink magnitude (eyes fully closed, squinting, narrowing, etc);
- The blink type (both eyes closed, only one eye).

As mentioned in Section 3.6.3, there is a mismatch between Emily's neutral face and the actor's neutral face. In order to disguise this discrepancy as much as possible, a fixed constant was added to the blenshape weights controlling the smile, and we found that the best value for this constant was 0.35.

4.3 Skin Rendering

Initially for all the Image Analogies work we used an implementation based on [ImAa]. This library was quite slow and it would not give results as good as the ones shown in the original paper. We then moved to a parallel CUDA implementation [ImAb]; with this code the results were inconsistent, dependent on the number of threads, and exhibited erroneous pixel values, as shown in Figure 4-3. The original code for Image Analogies has not been maintained since 2001, a patched version of the library, along with instruction on how to compile and run the code are provided alongside this report.

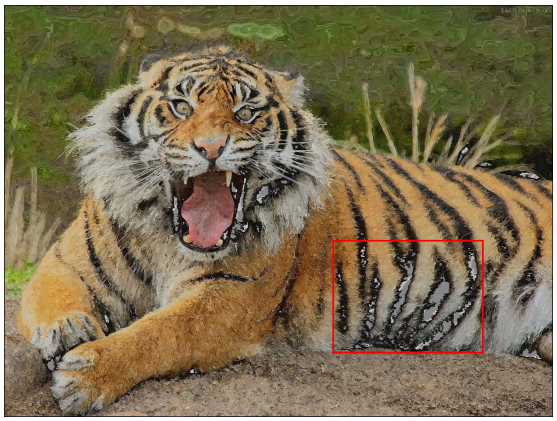


Figure 4-3: Result for the CUDA Image Analogies implementation, erroneous values are highlighted in the red square, image taken from [ImAb].

There are two important problems when using Image Analogies filter for adding detail to textures. The first one arises due to the difference in colour distributions between the inputs. For our data, there is a mismatch between images $\{A, A'\}$ and B, which was caused by divergent lighting conditions in the capture studio. Since we have the prior knowledge that the images should represent the same areas on the face, we applied colour correction techniques to match the images. The second issue derives from the local texture patch growth that the algorithm uses to replicate coherent patches in the input images. If the kernel sizes does not match the features that we want to preserve in the example images, unconnected patches will be produced. Moreover, this problem will be exacerbated with the face segmentation approach that was presented in Section 3.7. In Figure 4-4, the top half of the image shows an example of such kernel size error, and the seam between the two areas is quite evident due to the patch segmentation.

Normal map synthesis also required some pre- and post-processing steps for the pipeline to work. The ground truth normal maps that we use as $\{A, A'\}$ were taken from Graham *et al*. [GTB*13]. This data was encoded in the EXR format [EXR], which was not supported by the code we were using. Each EXR image was converted to two PNG images, one encoding the positive values of the original data and the other encoding the negative ones. The Image Analogies filter was run separately for each of them, and the results were recombined again into an EXR image.

Figure 4-4: Erroneously generated skin texture.

To generate the results with the image super-resolution technique from Jianchao *et al*. [YWHM10], we made use of the image dictionaries provided by the authors. In order to increase the quality and efficiency of the computation, the texture was divided into the segments introduced in Section 3.7. Dividing the face into segments exploits more efficiency due to the nowadays more common multi-core CPU architectures, as we can run the algorithm for each patch independently.

Chapter 5

Results

In this chapter we present our final facial animation result, highlight its strengths and weaknesses, and compare the results we have produced throughout the course of this project.

Overall, our final animation resembles the facial performance of the actor quite well. A selection of image frames from the finished render are shown in Figure 5-1, along side the corresponding poses of the actor. The animation plays at 60 frames-per-second and it is fluid and expressive - see the final video in the supplementary material. It is perhaps surprising how well the resulting facial animation respects the intrinsic differences between the actor's face shape and Emily's face shape, i.e. the animated expressions are not exact replicas of the actor's expressions, but instead they are Emily versions of the actor's expressions. As we can see in the images in Figure 5-1 the blendshape model is able to capture the extreme facial movements of the actor, including opening the mouth wide, funneling of the lips and scrunching of the face. It is clear that by using the 3D positions of markers on the face rather than simply their 2D positions from a single-view sequence, we able to capture the full 3D motion of the actor's face, for example note the slight forward jutting movement of the jaw in the bottom image. This gives the final facial animation much more nuance and character, and ultimately contributes to a much more believable animation.

Our early animations were generated by solely transferring the computed blendshape weights onto the Emily rig in Maya®. However the resulting animations appeared slightly life-less and dull as the animations did not include subtle movements such as blinking of the eyes, the eye motion, and any movement of the head - giving the animation a slightly robotic appearance. The final result was enhanced by manually activating the blendshapes that control blinks and winks throughout the sequence, an example of which is shown in Figure 5-2a. We simply analysed the reference video and noted the frames in which the eyelid start to close and open again. Since we tracked points on the face just around the eyes, when the actor blinks the muscles around the eyes move in and this is captured by the blendshape animation quite well. When combined with the manually added eyelid movement this gives a very impressive and natural looking motion. Moreover, when we reintroduced the rigid-head motion that was removed before solving for blendshape weights (see Section 3.5.1), the animation immediately became

much more dynamic, see Figure 5-3.

However, the final animation does have a number of flaws. First, since only a sparse set of points is used to find the blendshape weights, some motion remains poorly captured. For instance, the expression in Figure 5-3a presents such a situation. Here the sparse points alone are unable to define the downward positioning of the mouth; the sparse markers around the mouth form a ellipse which is then replicated by Emily without taking into consideration the fact that the actor's mouth is shut. This problem could be solved by actually tracking the inner edge of the lips so that we know when the mouth is actually open or closed.

Another issue is visible in Figure 5-3b, where the specific combination of blendshape weights here has caused the spherical structure that represents the inside of the mouth to penetrate through the side of Emily's cheek. Unfortunately, we have no control over the motion of this inner structure, and it cannot be hidden since in the Maya®rig it is connected to the mesh of the face. Our approach to this problem involves limiting the upper bound on the weights to ensure that the combination of blendshapes that cause this problem does not occur; by trial and error we were able to eliminate most of the problematic mixtures of shapes.

The final animation is of a much higher quality than our previous results and the way Emily's face moves is much closer to that of the actor's. This is partly due to the method we use to solve for the blendshape weights. In earlier approaches we would transform the sparse performance of the actor using thin plate splines to get an estimate of Emily's sparse performance, and then solve for expressions made by the actor using the Emily blendshapes. But by using the given Emily blendshapes we found that computing a similar set of blendshapes for the actor and solving for the actor's expressions using the generated sparse blendshapes gave a much more plausible set of facial expressions than solving for the actor's expressions in the Emily domain. Usually in industry when doing performance-driven facial animation, we would have a blendshape model of the actor's face and we would solve for the actor's expressions using that model. The problem then becomes just a retargeting problem to transfer the computed blendshape expressions from one model to another.

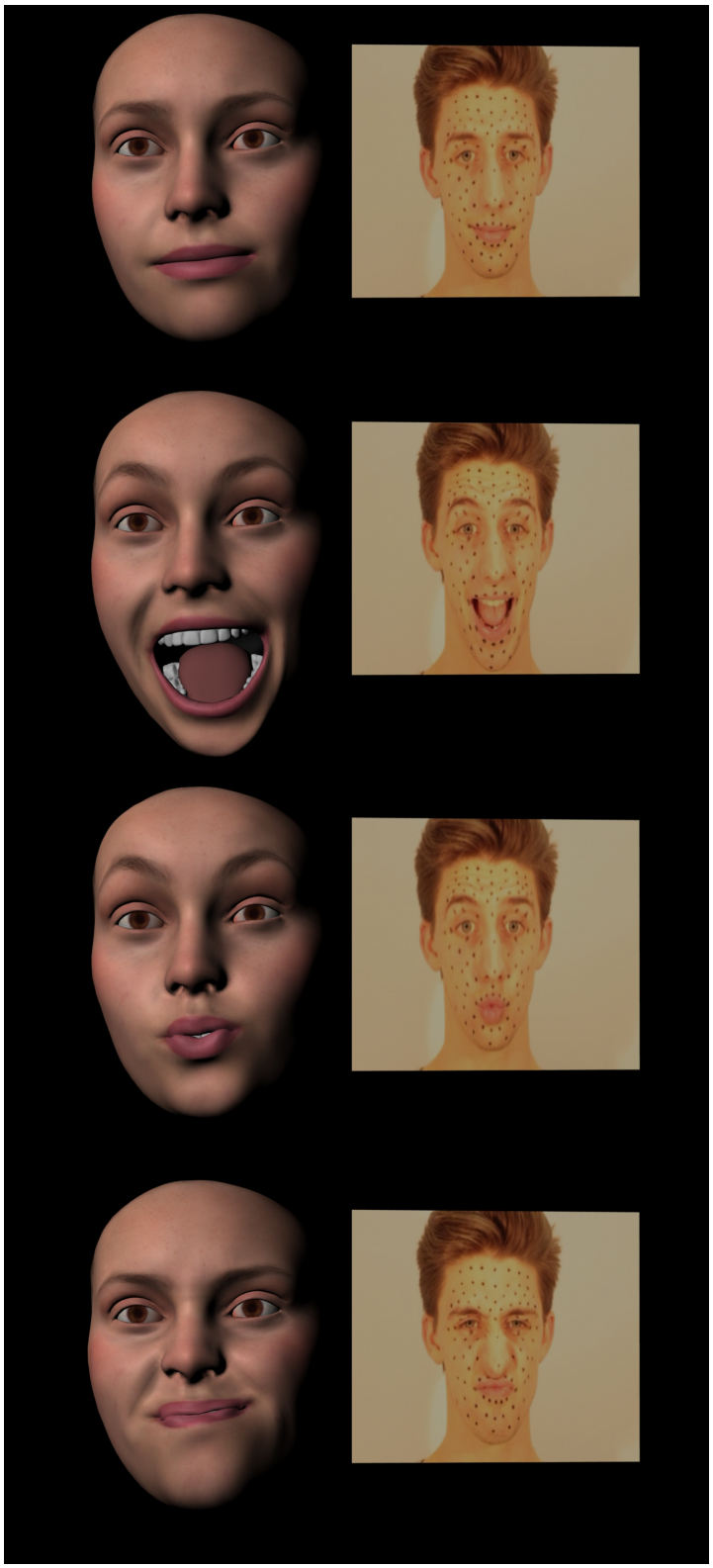
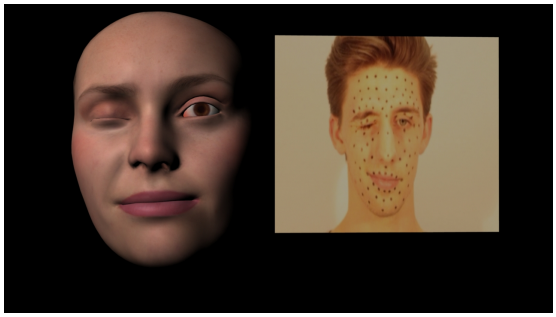
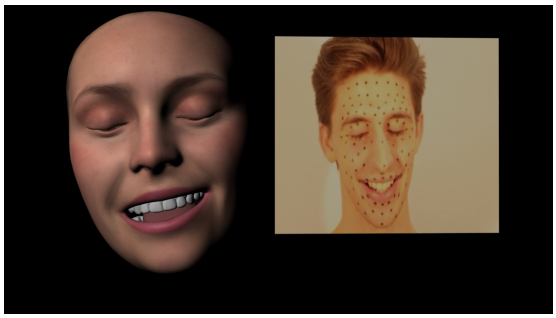


Figure 5-1: The final animation resembles the captured sequence quite well.

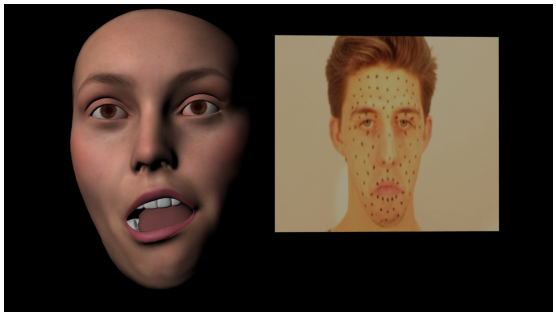


(a) Manually added blinks.

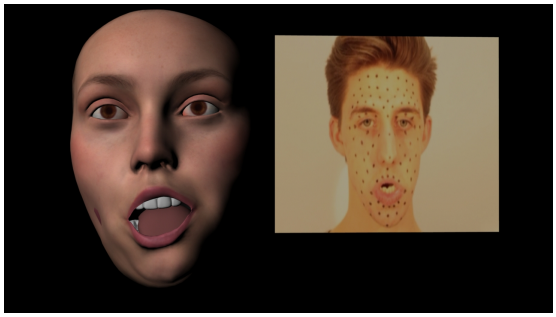


(b) Reintroduced the head rotation.

Figure 5-2: Improvements to the blendshape model.



(a) Sparse data does not describe the motion fully.



(b) The mesh inside the mouth is allowed to penetrate itself.

Figure 5-3: Shortcomings of the current model.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The objective of our project was to produce a prototype VFX pipeline that takes a captured facial performance as input, and creates an animation of a three-dimensional model of a human face.

In this report we have presented the details of our proposed pipeline. First we capture an actor's performance using a pair of stereo cameras. We calibrate the cameras and stereo rectify the image sequences from both cameras. Then we track the motion of a facial performance by detecting and tracking markers drawn onto the actor's face. We made use of SIFT features to detect the facial markers and then the KLT tracking algorithm to track them through the sequence. Once the 2D marker positions are found in every frame over the sequence we are able to obtain a reasonably accurate sparse 3D reconstruction of the actor's performance, given that we can accurately calibrate our stereo camera system and the camera projection matrices are known. The 3D sequence is computed using standard linear triangulation. We then remove the rigid head motion by uses Procrustes analysis to align the 3D facial expression in each frame to the neutral face, given a set of corresponding rigid points. We also track the pupils for adding eye movement to the final animation. Then given the target Emily blendshape rig, we select corresponding sparse points on the neutral Emily mesh and compute a TPS transformation to the points on the actor's neutral face. We apply this transformation to the set of Emily blendshapes to obtain a set of blendshapes for the actor, from which we use to solve for blendshape weights given a novel facial expression from the actor. Finally, the blendshape weights are applied to the Emily rig, and subtle details are added, including the rigid head motion, eye motion, and blinks. This results in the final animation we have presented.

The quality of an animation may be evaluated in a number of different ways. In our final animated sequence Emily is able to closely mimic the motion of the actor while retaining individuality imposed by the structure and features of Emily's face. The realism of our animations improved with time; the first animation has no eyes or mouth, and it exhibits unusual

behaviours and mesh warping artefacts. In contrast, the final animation is a significant improvement; it includes the motion of the eyes and the teeth, as well as more plausible overall performance, and thus it looks significantly more appealing. Though more realistic than a hand-drawn character or an industrial robot, our results lack the naturalness of human motion. We believe that our results belong in the uncanny valley of the familiarity versus human likeness diagram, see Figure 6-1. The theory of the uncanny valley suggests that human-like appearance and behaviour is perceived as repulsive as the resemblance increases [UV].

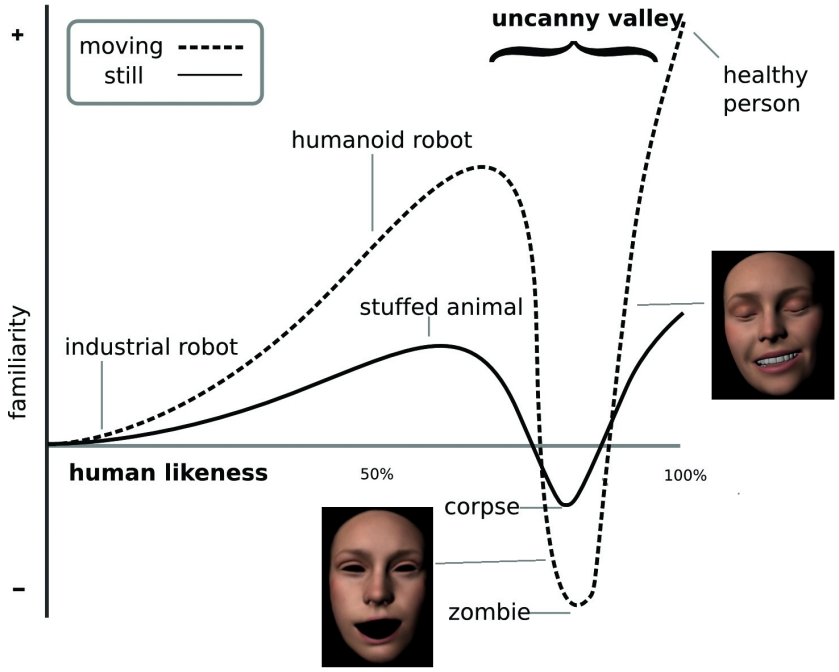


Figure 6-1: Our animations in the context of the uncanny valley.

Despite our animation being somewhere in the uncanny valley, we believe the outcome of this project has actually been quite impressive. Not only is the final animation surprisingly life-like and true to the actor's original performance when just based upon the movements of a number of sparse markers on the face, but we have shown that it is possible to achieve a good quality performance-driven facial animation just by using standard and in-expensive off-the-shelf consumer-level tools. We don't use complicated synchronised cameras and lighting rigs - we use standard consumer level DSLR cameras, we use markers hand-drawn on the face with a make-up pencil, and we make use of a free and readily available 3D blendshape rig. This shows the power of the blendshape animation technique and what can be achieved.

In terms of rendering, doubling the resolution of the texture for the Emily model provided an improvement in the realism in the final animation. This enhancement is mostly noticeable on close up views, where in medium distance or further shots it provides marginal to unnoticeable gains in quality. It should be noted that, even though the texture super-resolution technique is quite computationally expensive to generate, once it is done, it will benefit every frame of the

animation without any further costs.

6.2 Future Work

There are number of ways we could possibly improve the resulting facial animation, starting with the way we obtain the motion capture data. Using properly synchronised cameras at higher frame-rates (around 200fps) would significantly aid with tracking and reduce reconstruction errors. Tracking needs to be improved by incorporating a local probabilistic model, as well as global tracking, to cope with occlusions and be more robust to non-rigid deformation. We were also never able to fully remove the rigid head motion with Procrustes alignment, which then affects the resulting blendshapes since this rigid movement gets incorporated into the displacements from the neutral expression. So tracking rigid markers on the head would be helpful, or exploring alternative methods to remove this such as using non-rigid ICP or fitting some underlying model of the head which is aligned on every frame. Alternatively, we could make use of the Vicon Cara system [Car], where the actor wears head-mounted cameras, although there will always be some minor head movement even with this. Other things we would like to investigate would be ways in which to track the movement of the eye-lids as they open, close, and squint etc. Additionally, tracking points on inside of the lips would help in the accuracy of the blendshape solve of certain expressions involving the mouth, which is one of the areas our current system struggles with.

There are a number of different techniques that could be used to construct blendshapes in the actor's domain. Deng *et al*. proposed designing the blendshapes by hand [DCFN06]. In particular, the authors choose a number of characteristic frames in the motion capture sequence, and manually design a set of shapes that perceptually match the expressions observed in these frames. They then form correspondences between the PCA coefficients of motion capture frames and the weights of the blendshapes. Finally, Radial Basis Functions are used to map the new frames in the sequence onto the three-dimensional model to produce the animation. Another option would be to use a muscle actuation basis, proposed by Choe and Ko [CK05]. The design of a muscle actuation basis is based on the human facial anatomy, and is driven by the actuation of different facial muscles. Our main reservation about these approach is the extensive manual effort required to design the set of initial correspondences between the chosen frames and the three-dimensional mesh.

The accuracy of the model may be improved by taking extra care when matching the sparse points on Emily and the actor. One possible approach involves drawing a number of stable points on Emily and the actor and finding the mapping that transforms sparse Emily points to the actor's domain. Then a large number of additional points are drawn on Emily, and the same transformation is used to transform these points to the actor's domain. These points are then used to identify the ideal location of the markers on the actor's face. Moreover, a 3D printed mask may be used to ensure high accuracy when placing the markers.

The main weakness of the current method is the amount of manual work that needs to be done before the animation takes its final shape.

The realism of the final animation could also be enhanced by applying more complex skin rendering techniques. In this line, we could add layered materials which model subsurface scattering in the skin [WMP*06]. A further extension would be to include haemoglobin related phenomena, as proposed by [DWd*08, JSB*10]. Adding bump maps or normals maps also increases the quality of the final result, however a light stage is needed to acquire such data with enough resolution [GTB*13].

Bibliography

[ACCO00] Andersen K., Christiansen E., Conn A., Overton M.: An efficient primal-dual interior-point method for minimizing a sum of euclidean norms. *SIAM Journal on Scientific Computing 22*, 1 (2000), 243–262.

[ACP03] Allen B., Curless B., Popović Z.: The space of human body shapes: Reconstruction and parameterization from range scans. *ACM Trans. Graph. 22*, 3 (July 2003), 587–594.

[ARL*09] Alexander O., Rogers M., Lambeth W., Chiang M., Debevec P.: The digital emily project: Photoreal facial modeling and animation. In *ACM SIGGRAPH 2009 Courses* (New York, NY, USA, 2009), SIGGRAPH '09, ACM, pp. 12:1–12:15.

[ARV07] Amberg B., Romdhani S., Vetter T.: Optimal Step Nonrigid ICP Algorithms for Surface Registration. *2007 IEEE Conference on Computer Vision and Pattern Recognition* (June 2007), 1–8.

[Ash01] Ashikhmin M.: Synthesizing natural textures. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics* (New York, NY, USA, 2001), I3D '01, ACM, pp. 217–226.

[BA06] Bose N., Ahuja N.: Superresolution and noise filtering using moving least squares. *Image Processing, IEEE Transactions on 15*, 8 (Aug 2006), 2239–2248.

[BB14] Beeler T., Bradley D.: Rigid stabilization of facial expressions. *ACM Trans. Graph. 33*, 4 (July 2014), 44:1–44:9.

[BBA*07] Bickel B., Botsch M., Angst R., Matusik W., Otaduy M., Pfister H., Gross M.: Multi-scale capture of facial geometry and motion. *ACM Trans. Graph. 26*, 3 (July 2007).

[Ben05] Bennett D.: The faces of "the polar express". In *ACM SIGGRAPH 2005 Courses* (New York, NY, USA, 2005), SIGGRAPH '05, ACM.

[Bla] Optical Flow software (C and Matlab). http://cs.brown.edu/~black/code.html. Accessed: 2015-02-29.

[BRM12] Baltrušaitis T., Robinson P., Morency L.-P.: 3d constrained local model for rigid and non-rigid facial tracking. pp. 2610–2617.

[BV99] Blanz V., Vetter T.: A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1999), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., pp. 187–194.

[BWP13] Bouaziz S., Wang Y., Pauly M.: Online modeling for realtime facial animation. *ACM Trans. Graph. 32*, 4 (July 2013), 40:1–40:10.

[Cal] Camera Calibration Toolbox for Matlab. http://www.vision.caltech.edu/bouguetj/calib_doc/. Accessed: 2015-02-10.

[Car] Cara. http://www.vicon.com/products/camera-systems/cara. Accessed: 2015-04-15.

[CB05] Chappalli M., Bose N.: Simultaneous noise filtering and super-resolution with second-generation wavelets. *Signal Processing Letters, IEEE 12*, 11 (Nov 2005), 772–775.

[CG00] Cipolla R., Giblin P.: *Visual Motion of Curves and Surfaces*. Cambridge University Press, New York, NY, USA, 2000.

[CK05] Choe B., Ko H.-S.: Analysis and synthesis of facial expressions with hand-generated muscle actuation basis. In *ACM SIGGRAPH 2005 Courses* (New York, NY, USA, 2005), SIGGRAPH '05, ACM.

[CLK01] Choe B., Lee H., Ko H.-S.: Performance-driven muscle-based facial animation. *Journal of Visualization and Computer Animation 12*, 2 (2001), 67–79.

[DCFN06] Deng Z., Chiang P.-Y., Fox P., Neumann U.: Animating blendshape faces by cross-mapping motion capture data. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2006), I3D '06, ACM, pp. 43–48.

[DWd*08] Donner C., Weyrich T., d'Eon E., Ramamoorthi R., Rusinkiewicz S.: A layered, heterogeneous reflectance model for acquiring and rendering human skin. *ACM Trans. Graph. 27*, 5 (Dec. 2008), 140:1–140:12.

[Ede00] Edelsbrunner H.: Triangulations and meshes in computational geometry. *Acta Numerica 2000* (2000), 1–81.

[EXR] EXR Format. http://www.openexr.com/. Accessed: 2015-05-19.

[Faca] Faceware Tech. http://facewaretech.com/. Accessed: 2015-05-19.

[Facb] Faceware Tech: Free Rigs. http://facewaretech.com/learn/training-assets/. Accessed: 2015-05-19.

[FACc] Facial Action Coding System. http://www.cs.cmu.edu/~face/facs.htm. Accessed: 2015-05-27.

[GGW*98] Guenter B., Grimm C., Wood D., Malvar H., Pighin F.: Making faces. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 55–66.

[GHDS03] Grinspun E., Hirani A. N., Desbrun M., Schröder P.: Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), SCA '03, Eurographics Association, pp. 62–67.

[Gow75] Gower J.: Generalized procrustes analysis. *Psychometrika 40*, 1 (1975), 33–51.

[GTB*13] Graham P., Tunwattanapong B., Busch J., Yu X., Jones A., Debevec P., Ghosh A.: Measurement-based synthesis of facial microgeometry. *Computer Graphics Forum 32*, 2pt3 (2013), 335–344.

[GVWT13] Garrido P., Valgaert L., Wu C., Theobalt C.: Reconstructing detailed dynamic face geometry from monocular video. *ACM Trans. Graph. 32*, 6 (Nov. 2013), 158:1–158:10.

[HJO*01] Hertzmann A., Jacobs C. E., Oliver N., Curless B., Salesin D. H.: Image analogies. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 327–340.

[HMD06] Humblot F., Mohammad-Djafari A.: Super-resolution using hidden markov model and bayesian detection estimation framework. *EURASIP Journal on Applied Signal Processing 10* (2006), ID.

[IGAJG15] Iglesias-Guitian J. A., Aliaga C., Jarabo A., Gutierrez D.: A biophysically-based model of the optical properties of skin aging. *Computer Graphics Forum (EUROGRAPHICS 2015) To Appear* (2015).

[ImAa] Image Analogies: C++ single threaded code. http://vis.berkeley.edu/courses/cs294-69-fa11/wiki/index.php/FP-JeffDonahue. Accessed: 2015-05-19.

[ImAb] Image Analogies: CUDA code. https://sites.google.com/a/college.harvard.edu/yaoyu-imageanalogies/parallel. Accessed: 2015-05-19.

[ImAc] Image Analogies: Hertzmann's code. http://www.mrl.nyu.edu/projects/image-analogies/. Accessed: 2015-05-19.

[JF09] Ji H., Fermuller C.: Robust wavelet-based super-resolution reconstruction: Theory and algorithm. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 31*, 4 (April 2009), 649–660.

[JSB*10] Jimenez J., Scully T., Barbosa N., Donner C., Alvarez X., Vieira T., Matts P., Orvalho V., Gutierrez D., Weyrich T.: A practical appearance model for dynamic facial color. In *ACM SIGGRAPH Asia 2010 Papers* (New York, NY, USA, 2010), SIGGRAPH ASIA '10, ACM, pp. 141:1–141:10.

[JTDP03] Joshi P., Tien W. C., Desbrun M., Pighin F.: Learning controls for blend shape based realistic facial animation. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), SCA '03, Eurographics Association, pp. 187–192.

[Kan] Kanade-Lucas-Tomasi (KLT) Feature Tracker. http://web.yonsei.ac.kr/jksuhr/articles/Kanade-Lucas-Tomasi%20Tracker.pdf. Accessed: 2015-03-15.

[KLT] Track points in video using Kanade-Lucas-Tomasi (KLT) algorithm. http://uk.mathworks.com/help/vision/ref/vision.pointtracker-class.html. Accessed: 2015-03-05.

[LCBT13] Li W., Cosker D., Brown M., Tang R.: Optical Flow Estimation Using Laplacian Mesh Energy. *2013 IEEE Conference on Computer Vision and Pattern Recognition* (June 2013), 2435–2442.

[LF96] Luong Q., Faugeras O.: The fundamental matrix: Theory, algorithms, and stability analysis. *International journal of computer vision* (1996), 1–46.

[lsqa] Solve constrained linear least-squares problems. http://uk.mathworks.com/help/optim/ug/lsqlin.html. Accessed: 2015-04-03.

[lsqb] Solve nonnegative least-squares constraints problem. http://uk.mathworks.com/help/matlab/ref/lsqnonneg.html. Accessed: 2015-04-03.

[Lui] Optical Flow Matlab/C++ Code. https://people.csail.mit.edu/celiu/OpticalFlow/. Accessed: 2015-02-29.

[LYYB13] Li H., Yu J., Ye Y., Bregler C.: Realtime facial animation with on-the-fly correctives. *ACM Trans. Graph. 32*, 4 (July 2013), 42:1–42:10.

[NN01] Noh J.-y., Neumann U.: Expression cloning. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 277–288.

[Nyq02] Nyquist H.: Certain topics in telegraph transmission theory. *Proceedings of the IEEE 90*, 2 (2002), 280–305.

[PHL*98] Pighin F., Hecker J., Lischinski D., Szeliski R., Salesin D. H.: Synthesizing realistic facial expressions from photographs. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 75–84.

[PKC*03] Pyun H., Kim Y., Chae W., Kang H. W., Shin S. Y.: An example-based approach for facial expression cloning. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), SCA '03, Eurographics Association, pp. 167–176.

[PSS02] Pighin F., Szeliski R., Salesin D.: Modeling and animating realistic faces from images. *International Journal of Computer Vision 50*, 2 (2002), 143–169.

[Sch05] Schmidt M.: Least squares optimization with l1-norm regularization. *CS542B Project Report* (2005).

[Sco] Scopus. http://www.scopus.com/. Accessed: 2015-05-21.

[SP04] Sumner R. W., Popović J.: Deformation transfer for triangle meshes. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 399–405.

[SS91] Sibson R., Stone G.: Computation of thin-plate splines. *SIAM J. Sci. Stat. Comput. 12*, 6 (Sept. 1991), 1304–1313.

[ST94] Shi J., Tomasi C.: Good features to track. *IEEE Conference on Computer Vision and Pattern Recognition* (1994), 593–600.

[Sur] Surrey Audio-Visual Expressed Emotion (SAVEE) Database. http://personal.ee.surrey.ac.uk/Personal/P.Jackson/SAVEE/Database.html. Accessed: 2015-02-20.

[TFM07] Takeda H., Farsiu S., Milanfar P.: Kernel regression for image processing and reconstruction. *Image Processing, IEEE Transactions on 16*, 2 (Feb 2007), 349–366.

[TH84] Tsai R., Huang T. S.: Multiframe image restoration and registration. *Advances in computer vision and Image Processing 1*, 2 (1984), 317–339.

[TK95] Tom B., Katsaggelos A.: Reconstruction of a high-resolution image by simultaneous registration, restoration, and interpolation of low-resolution images. In *Image Processing, 1995. Proceedings., International Conference on* (Oct 1995), vol. 2, pp. 539–542 vol.2.

[TM10] Tian J., Ma K.-K.: Stochastic super-resolution image reconstruction. *Journal of Visual Communication and Image Representation 21*, 3 (2010), 232 – 244.

[TM11] Tian J., Ma K.-K.: A survey on super-resolution imaging. *Signal, Image and Video Processing 5*, 3 (2011), 329–342.

[UV] Uncanny Valley. http://en.wikipedia.org/wiki/Uncanny_valley. Accessed: 2015-05-27.

[VBPP05] Vlasic D., Brand M., Pfister H., Popović J.: Face transfer with multilinear models. *ACM Trans. Graph. 24*, 3 (July 2005), 426–433.

[VF08] VEDALDI A., FULKERSON B.: VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008.

[Vic] Vicon. http://www.vicon.com/System/Markers. Accessed: 2015-05-20.

[WHL*04] WANG Y., HUANG X., LEE C.-S., ZHANG S., LI Z., SAMARAS D., METAXAS D., ELGAMMAL A., HUANG P.: High resolution acquisition, learning and transfer of dynamic 3-d facial expressions. *Computer Graphics Forum 23*, 3 SPEC. ISS. (2004), 677–686.

[WL00] WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 479–488.

[WLVGP09] WEISE T., LI H., VAN GOOL L., PAULY M.: Face/off: Live facial puppetry. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2009), SCA '09, ACM, pp. 7–16.

[WMP*06] WEYRICH T., MATUSIK W., PFISTER H., BICKEL B., DONNER C., TU C., MCANDLESS J., LEE J., NGAN A., JENSEN H. W., GROSS M.: Analysis of human faces using a measurement-based skin reflectance model. In *ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), SIGGRAPH '06, ACM, pp. 1013–1024.

[XCLT14] XU F., CHAI J., LIU Y., TONG X.: Controllable high-fidelity facial performance transfer. *ACM Trans. Graph. 33*, 4 (July 2014), 42:1–42:11.

[XY97] XUE G., YE Y.: An efficient algorithm for minimizing a sum of euclidean norms with applications. *SIAM Journal on Optimization 7*, 4 (1997), 1017–1036.

[YWHM10] YANG J., WRIGHT J., HUANG T., MA Y.: Image super-resolution via sparse representation. *Image Processing, IEEE Transactions on 19*, 11 (Nov 2010), 2861–2873.