Georgia Institute of Technology, Graphics, Visualization and Usability Center, Technical Report # GIT-GVU-98-24, August 1998.

Head Tracking Using a Textured Polygonal Model

Arno Schödl *schoedl@cc.gatech.edu*
Antonio Haro *haro@cc.gatech.edu*
Irfan A. Essa *irfan@cc.gatech.edu*

College of Computing, GVU Center
Georgia Institute of Technology
Atlanta, GA 30332-0280, U. S. A.
http://www.gvu.gatech.edu/perception/projects/head-track/

Abstract

We describe the use of a three-dimensional textured model of the human head under perspective projection to track a person's face. The system is hand-initialized by projecting an image of the face onto a polygonal head model. Tracking is achieved by finding the six translation and rotation parameters to register the rendered images of the textured model with the video images. We find the parameters by mapping the derivative of the error with respect to the parameters to intensity gradients in the image. We use a robust estimator to pool the information and do gradient descent to find an error minimum.

1. Introduction

Head tracking is an important processing step for many vision-driven interactive user interfaces. The obtained position and orientation allow for pose determination and recognition of simple gestures such as nodding and head shaking. The stabilized image obtained by perspective de-warping of the facial image according to the acquired parameters is ideal for facial expression recognition [7] or face recognition applications.

In this paper we present a novel method for head tracking using a textured head model. We also present some previous work in head tracking, and discuss the benefits of our approach.

1.1 Previous Work

Much research effort has been expended on locating and tracking heads and recognizing pose and facial expressions from video. Face detection is still considered a 2D problem where facial features, facial color, and the shape of the face are obtained from the image plane for locating the head [10,11]. To extract 3D parameters, a model that encodes head orientation and position must be used. All approaches discussed here including our own method require some initialization of a model to a face.

Black and Yacoob [4] use a rectangular planar patch under affine transformation as a face model. Similar patches are attached to the eyebrows and the mouth. They follow the movements of the underlying facial patch but detect differential movements of their facial parts. Affine motion, like any 2D model, has its limitations because it has no concept of self-occlusion occurring at the sides of the head and around the nose. Affine transformations also distort the frontal face image when they are used to model larger rotations.

Azarbayejani et al. [1] use feature point tracking projected on an ellipsoidal model to track the head position. Feature point tracking has the drawback that tracking fails when the feature points are lost due to occlusions or lighting variations. New feature points are acquired, but only at the cost of excessive error accumulation.

Jebara and Pentland [9] also use feature point tracking, but with automatically located head features like eyes and mouth corners. The 3D position of the feature points is estimated using a structure from motion technique that pools position information over the image sequence with an extended Kalman filter. The estimate of the feature point position is filtered using Eigenfaces to restrict the measurements to match an expected facial geometry.

Basu, Essa and Pentland [2] couple an ellipsoidal model with general optical flow computation for tracking. First, optical flow is computed independently of face position and orientation using a gradient-based method. Then the motion of an ellipsoidal mesh regularizes the flow. The method's strengths are also its weaknesses. It copes well with large head rotations since it does not rely on any fixed features. For the same reason, it has no means to ground the model to the face, thus the error accumulates and the mesh slowly drifts off the face.

La Cascia, Isidoro and Sclaroff [5] use a textured cylinder as a head model. The approach is most similar to ours in that it uses a three-dimensional textured model. The technique differs from ours since it uses a cylinder instead of a full head model and it uses a dynamic texture

for tracking. The lack of fixed features again leads to error accumulation although confidence maps are used to minimize this problem.

DeCarlo and Metaxas [6] use a polygonal head model that is hand-positioned on the subject's face. While our method uses texture, theirs extracts optical flow at some feature points and regularizes it by the model movements. The measurements are stabilized using a Kalman filter. Using optical flow leads to a similar error accumulation as in [2]. However, their system additionally uses face edge information to prevent divergence. This work also extracts face shape and facial expressions.

1.2 Our Approach

We use a 3D-textured polygon model that is matched with the incoming video stream. We merge graphics and vision by using graphics hardware to produce renderings that are good enough to match the real images. The initialized and textured head model is rendered. The intensity difference between rendered image and video image, in conjunction with the image gradient, is then mapped to derivatives of our six model parameters. They lead to a local error minimum that is the best possible match between the rigidly transformed model and the real video image. The technique can be seen as a more sophisticated regularization of optical flow, in principle similar to [3].

An additional important feature is the exploitation of graphics hardware. While special vision hardware such as user-programmable DSPs are still expensive and not present in an ordinary PC, graphics hardware is ubiquitous and can be used to render models and to perform hidden surface elimination.

2. The Method

2.1 Mapping Model Parameters to the Image Gradient

Let p be a set of 3D model points with an associated intensity $M(p)$, and let $I(x)$ be a camera picture. Then using a transformation T from some model point to screen coordinates with a parameter set $\{\alpha_i\}$, we minimize the sum over a robust error norm:

$$E = \sum_{\{p\}} \rho\Big(I(T(p,\{\alpha_i\})) - M(p)\Big),$$

where ρ is the Geman & McClure robust error norm [5] defined as

$$\rho(x) = \frac{x^2}{\sigma + x^2}.$$

Here, σ controls the distance beyond which a measurement is considered an outlier [3].

The error function derived with respect to a particular parameter α_j is

$$\frac{dE}{d\alpha_j} = \sum_{\{p\}} \psi\Big(I(T(p,\{\alpha_i\})) - M(p)\Big) \frac{dI(T(p,\{\alpha_i\})}{d\alpha_j},$$

where ψ designates the derivative of ρ. Expanding the second term to evaluate it as a linear combination of the image intensity gradients we get

$$\frac{dI(T(p,\{\alpha_i\}))}{d\alpha_j} = \frac{dI(T(p,\{\alpha_i\}))_x}{dT(p,\{\alpha_i\})_x} \cdot \frac{dT(p,\{\alpha_i\}_x}{d\alpha_j} + \frac{dI(T(p,\{\alpha_i\}))_y}{dT(p,\{\alpha_i\})_y} \cdot \frac{dT(p,\{\alpha_i\}_y}{d\alpha_j}.$$

The first factors of the terms are just the image intensity gradient at the position $T(p,\{\alpha_i\})$ in the x and y directions, written as I_x and I_y:

$$\frac{dI(T(p,\{\alpha_i\}))}{d\alpha_j} = I_x(T(p,\{\alpha_i\})) \cdot \frac{dT(p,\{\alpha_i\})_x}{d\alpha_j} + I_y(T(p,\{\alpha_i\})) \cdot \frac{dT(p,\{\alpha_i\})_y}{d\alpha_j}.$$

We use the Sobel operator to determine the intensity gradient at a point in the image. Note that in this formulation the 3D model, its parameterization, and the 3D to 2D transformation used are arbitrary provided the model can be rendered onto the screen.

2.2 Special Case: Rigid Transformation and Perspective Projection

For our more specific case we use the transformation matrix, written in homogenous coordinates, going from 3D to 2D:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & f^{-1} & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot R_z R_y R_x,$$

with R_x, R_y, and R_z being rotation matrices around the x, y, and z axis, for some angles r_x, r_y, and r_z respectively. t_x, t_y, and t_z are translations along the axes. We assume that the focal length f is known and found that a rough estimate gives good results. Without loss of generality, we assume that $r_x=r_y=r_z=t_x=t_y=t_z=0$, which corresponds to a camera at the origin looking down the positive z-axis. Note that in this case the order of rotations does not affect our calculations.

Omitting the parameters of I, I_x, and I_y, we obtain as derivatives of the intensity with respect to our parameters for a particular model point $p = (p_x, p_y, p_z)^T$

$$\frac{dI}{dr_x} = -f\,\frac{I_y(p_y^2+p_z^2)+I_x p_x p_y}{p_z^2},$$

$$\frac{dI}{dr_y} = f\,\frac{I_x(p_x^2+p_z^2)+I_y p_x p_y}{p_z^2},$$

$$\frac{dI}{dr_z} = f\,\frac{I_y p_x - I_x p_y}{p_z},$$

$$\frac{dI}{dt_x} = -f\,\frac{I_x}{p_z},$$

$$\frac{dI}{dt_y} = -f\,\frac{I_y}{p_z},$$

$$\frac{dI}{dt_z} = f\,\frac{I_x p_x + I_y p_y}{p_z^2}.$$

For color video footage we use the Geman & McClure norm on the distance in RGB space rather than on each color channel separately. σ is set such that anything beyond a color distance of 50 within the 256^3 color cube is considered an outlier. We sum the results of each color channel to obtain a minimum error estimate.

2.3 Model Parameterization with Uncorrelated Feature Sets

The parameterization given above is not adequate for implementation because the rotations and translations of the camera used in the parameterization look very similar in the camera view resulting in correlated feature sets. For example, rotating around the *y*-axis looks similar to translating along the *x*-axis. In both cases the image content slides horizontally, with slight differences in perspective distortion. In the error function space this results in long, narrow valleys with a very small gradient along the bottom of the valley. It destabilizes and slows down any gradient-based minimization technique.

A similar problem is translation along and rotation around the z-axis, which causes the image of the head not only to change size or to rotate, but also to translate if the head is not exactly in the screen center.

To overcome these problems we choose different parameters for our implementation than those given in 2.2. They are illustrated in figure 1.

We again assume that the camera is at the origin, looking along positive *z*, and that the object is at some point (o_x, o_y, o_z).

Note that the directions of the rotation axes depend on the position of the head. r_{cx} and r_{cy} have the same axis as r_{ox} and r_{oy}, respectively, but rotate around the camera as opposed to around the object. r_{cx} and r_{ox} are orthogonal to both the connecting line between the camera and the head, and to the screen's vertical axis. Rotation around r_{cx} causes

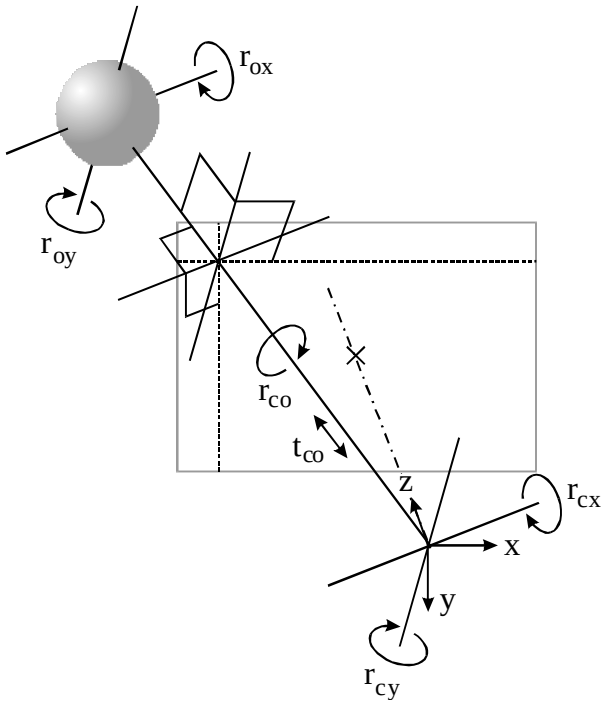

Figure 1: Parameterization of the model with nearly uncorrelated feature sets

the head image to move vertically on the screen, but it is not rotating relative to the viewer and it does not change its horizontal position. Rotation around r_{ox} causes the head to tilt up and down, but keeps its screen position unchanged.

r_{cy} and r_{oy} are defined similarly for the screen's horizontal coordinate. Note that r_{cy}/r_{oy} and r_{cx}/r_{ox} are not orthogonal except when the object is on the optical axis. However, the pixel movement on the screen that is caused by rotation around them is always orthogonal. It is vertical for the r_{cy}/r_{oy} pair and horizontal for the r_{cx}/r_{ox} pair.

Rotation around r_{co} causes the head's image to rotate around itself, while translation along t_{co} varies the size of the image. Neither changes the head screen position.

The parameterization depends on the current model position and changes while the algorithm converges. Different axes of rotation and translation are used for each iteration step.

We now have to find the derivatives of the error function with respect to our new parameter set. Fortunately, in Euclidean space any complex rotational movement is a combination of rotation around the origin and translation. We have already calculated the derivatives for both in section 2.2. The derivatives of our new parameter set are linear combinations of the derivatives of the naïve parameter set.

The parameterization change does not eliminate the long valleys we started with. Instead, they are now oriented along the principal axes of the error function space. Simple parameter scaling can convert the narrow

valleys to more circular-shaped bowls; these allow for efficient nonlinear minimization. In 2D subspace plots of the error function we visually inspect the shape of the minimum and adjust the scaling appropriately.

For convergence we use simple steepest descent. While more sophisticated schemes like conjugate gradients brought relatively large improvements to the naïve parameterization, we did not see any improvements with our final model. One reason is certainly that our error function is discretely defined and possibly far from quadratic. Additionally, the Hessian matrix may rapidly change during the iterations.

Our a priori efforts to find good parameters and suitable scales exploit the underlying structure of the problem and are crucial for speeding up convergence.

2.4 Relationship between Uncorrelated Feature Sets and Preconditioning

Using uncorrelated features is equivalent to making the Hessian matrix, the second derivative of the error function, almost diagonal near the error function minimum. Choosing an appropriate scale for each of the parameters brings the condition number of the Hessian matrix close to unity, which is the goal of preconditioning in a general nonlinear minimizer. In our case we use a priori knowledge to avoid any analysis of the error function during run-time.

3. Implementation

3.1 Gaussian Pyramid

To allow larger head motions, a 2-level Gaussian pyramid is used. At each level of the pyramid, starting from the lowest resolution, the parameters are optimized and propagated to the next higher level, up to the original image resolution. At each resolution level of the pyramid we do a fixed number of 20 line minimizations, each requiring a minimum of 2 evaluations of the derivative. Less iteration will suffice for smaller movements. However, we did not implement a stopping criterion for the minimization process.

3.2 Using OpenGL

For each evaluation of the derivatives, two OpenGL renderings of the head are performed. The first rendering is for obtaining the textured image of the head using trilinear mipmapping to cope with the varying resolution levels of the pyramid. The second rendering provides flat-filled aliased polygons. Each polygon has a different color for visible surface determination in hardware. All transformations into camera and screen coordinates and the calculations of gradients to compute the parameter derivatives are done only for visible surfaces.

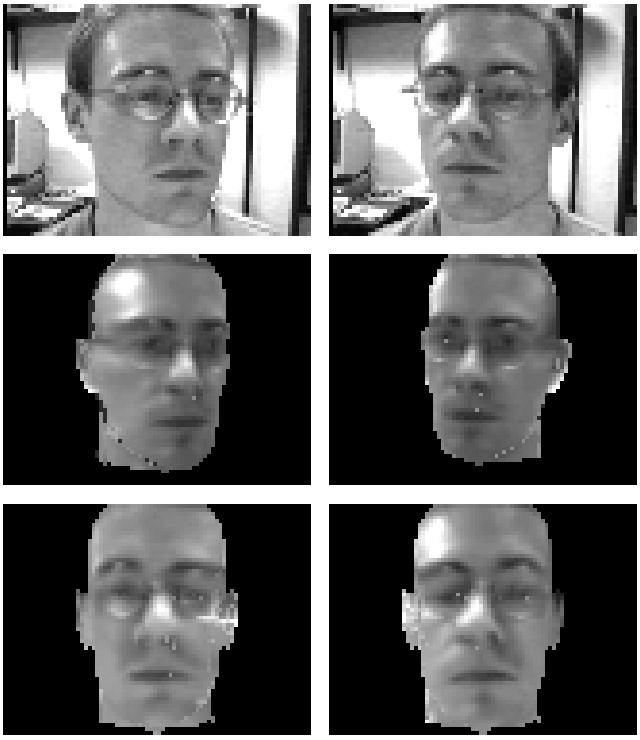

Figure 2: *Large head rotation between two frames. Top: video image, middle: rendered model, bottom: stabilized image obtained by reprojection onto the model.*

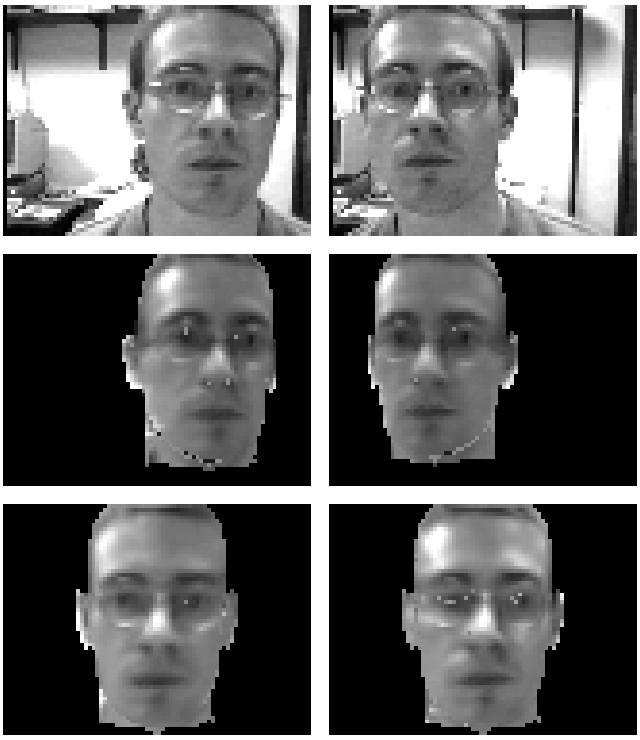

Figure 3: *Large head translation between two frames. Top: video image, middle: rendered model, bottom: stabilized image.*

Figure 4: The test sequence at 0 s, 2 s, 4 s, 6 s, 8 s and 9 s. Top: video, middle: model, bottom: stabilized image.

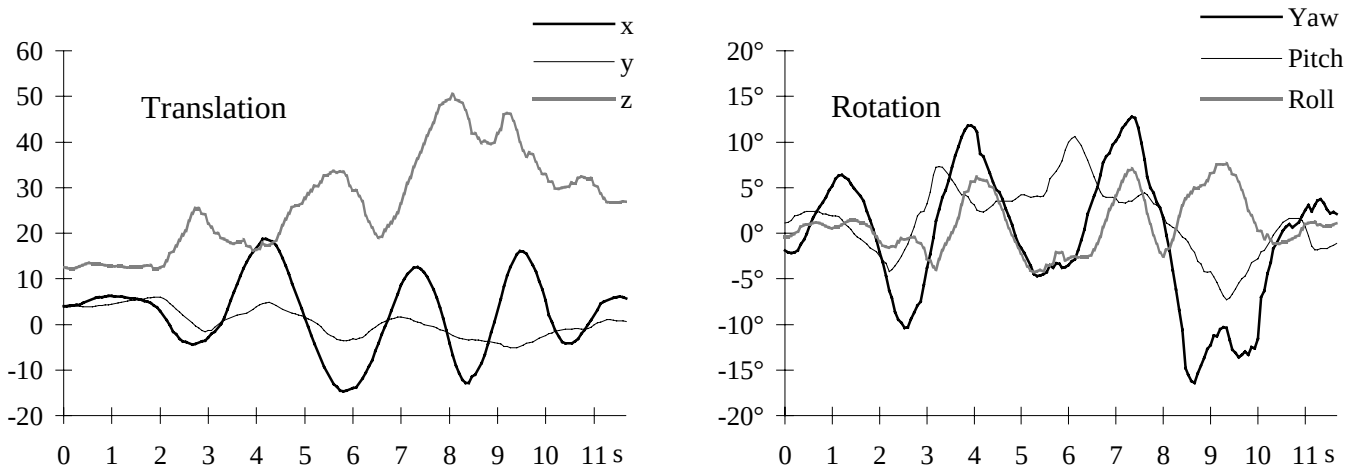

Figure 5: The translation and rotation parameters for the whole sequence.

Note that for our application, aliasing during the rendering step that determines the visible polygons is not a problem. Although visible polygons might be missed because of aliasing, polygon vertex information is only used to determine the 3D coordinates of a point in the image but nothing is done to the polygon itself. When initially projecting the texture onto the model visible surface computation is done at a higher resolution to reliably texture all visible polygons.

4. Results

We implemented the system on a Pentium II 300 MHz running under Windows NT with a ATI Rage Pro 3D graphics accelerator. The program is written entirely in C++. We used a resolution of 80x60 pixels, with a second Gaussian pyramid level of 40x30 pixels. We used both live video feeds and prerecorded sequences. All our footage was in color.

Our model is a male head from Viewpoint Data Labs with about 7000 triangles. The model is taken as is and is not customized to the test user's head.

4.1 Accuracy and Robustness

The robustness of the algorithm against large changes in the head parameters is impressive; rotations of more than 25 degrees and translations of more than half of the head size from one image to the next still converge. Figures 2 and 3 show one example of rotation and one of translation each showing large movements between two frames. Note that the test subject wears glasses and there are specular reflections on the subject's forehead from the ceiling lights. This does not cause significant problems for our tracking algorithm.

Our algorithm has problems when no texture data is available or when the 3D model differs extensively from the actual head geometry of the subject. We only project a

single texture from a frontal view onto the model. The sides of the model thus remain untextured or are only textured in low resolutions due to the small angle of incidence. Thus, head rotations where only the sides remain visible are registered with poor accuracy. Figure 4 shows images of a test sequence taken at 15 fps. During the first 8 seconds tracking is stable and the obtained stabilized image is usable. During the ninth second of the sequence the combination of a large head rotation and an incorrect registration of a specular reflection in the video with a reflection on the model lead to incorrect measurements. Figure 5 shows a sudden change in yaw around the ninth second, which corresponds to the incorrect registration. In such cases accurate tracking is usually reacquired when the head returns to a more front-facing orientation.

A major gain in performance can be expected when not only a single frontal image is used for projection but pictures from the sides as well. Another improvement would be a customizable head model to more accurately exploit head surface details such as nose and ear positions.

4.2 Speed and the Impact of OpenGL Acceleration

With all parameters set as described, the program runs at 9 seconds per frame using Microsoft's software OpenGL implementation. Using the ATI Rage Pro accelerator the computation time is 7 seconds per frame. The reason for the small gain in performance is the small frame size of only 80 by 60 pixels and the large triangle count of 7000. The Rage Pro only accelerates scan conversion but has no geometry engine. This greatly reduces the accelerator's benefit for scenes with many small triangles because all vertex transformations still have to be done in software.

More interestingly, profiling our system showed that for software rendering 91.6% and for hardware rendering 90.0% of the runtime was spent executing OpenGL commands. Only the remainders were computations for the main processor. Rapid advances in cheap graphics accelerator technology can be effectively used to improve the performance of our system. The ratio of OpenGL rendering time to main processor computation time changes with model complexity. Generally, similar systems with more complicated models will benefit more from OpenGL acceleration.

5. Discussion

Following the trend of merging computer vision with graphics we demonstrate a model-based approach to head tracking. This approach allows us to robustly track a person's pose and head orientations, which is essential for vision-based interfaces.

Our approach can be viewed as the top-down end of a hierarchy of solutions to object recognition and tracking problems. More bottom-up approaches pose the problem of extracting the right features and detecting their patterns to indicate the presence of a known object. In our approach we have constructed the object in enough detail that we can render what we want to see and then do the matching at the very low level of pixel differences and gradients. This is easier when there is exact knowledge available of what to expect in the scene and the number of degrees of freedom is small. The limitations of our approach become apparent when the model is not flexible enough and its set of parameters fails to make it closely resemble reality.

6. Conclusions and Future Work

Our model-based method allows for robust tracking of heads from video. We have experimented with tracking under normal lighting conditions and have found the results robust, reliable, and reproducible. We are now optimizing the system with superior graphics and vision hardware and intend to use it for accurate pose and facial expression recognition. Staying within the model-based framework, we want to increase the realism of our model by texturing it from all sides and by making its geometry adaptive to the user.

References

[1] A. Azarbayejani, T. Starner, B. Horowitz, and A. Pentland. Visually controlled graphics. PAMI, 15(6), 1993.

[2] S. Basu, I. Essa, and A. Pentland. Motion regularization for model-based head tracking. ICPR, 1996.

[3] M. Black and P. Anandan. The robust estimation of multiple motions: parametric and piecewise-smooth flow fields. TR P93-00104, Xerox PARC, 1993.

[4] M. J. Black and Y. Yacoob. Tracking and recognizing rigid and non-rigid facial motions using local parametric models of image motions. ICCV, 1995.

[5] M. La Cascia, J. Isidoro, S. Sclaroff. Head tracking via robust registration in texture map images. CVPR, 1998.

[6] D. DeCarlo and D. Metaxas. The Integration of Optical Flow and Deformable Models with Applications to Human Face Shape and Motion Estimation. CVPR 1996.

[7] I. Essa and A. Pentland. Coding analysis, interpretation, and recognition of facial expressions. PAMI, 19(7): 757-763, 1997.

[8] S. Geman and D. E. McClure. Statistical methods for tomographic image reconstruction, Bull. Int. Statist. Inst. LII-4, 5-21, 1987.

[9] T. S. Jebara, A. Pentland. Parametrized Stucture from Motion for 3D Adaptive Feedback Tracking of Faces. CVPR 1997.

[10] A. Pentland, B. Moghaddam, T. Starner. View-Based and Modular Eigenspaces for Face Recognition. CVPR 1994.

[11] H. A. Rowley, S. Baluja, T. Kanade. Human Face Detection in Visual Scenes. CVPR 1998.