Analyzing Facial Expressions for Virtual Conferencing

Peter Eisert and Bernd Girod
University of Erlangen

We present a model-based algorithm that estimates 3D motion and facial expressions from 2D image sequences showing head and shoulder scenes typical of video telephone and teleconferencing applications.

Video coding applications such as video conferencing, telepresence, and teleteaching have attracted considerable interest in recent years. Transmitting image sequences typically imposes high demands on storage, processing power, and network bandwidth. Common networks like POTS (Plain Old Telephone System), ISDN (Integrated Services Digital Network), and many computer networks provide only low bit rates and cannot handle transmitting uncompressed video. Therefore, encoding the sequences proves essential. Highly efficient coding schemes like H.263 and MPEG-1 and 2 achieve compression factors of about 1:100 with acceptable quality.

Using model-based coding techniques further improves compression while simultaneously providing us with the possibility of manipulating a scene's content interactively.[1,2] Model-based coding aims to create a virtual 3D world with distinct 3D objects described by their shape and texture. In this world we only have to specify how the objects move and deform. The objects' shape transmits only once, and we can then describe the scene in all the following frames with few parameters specifying the objects' position and deformation. We reconstruct the 2D image sequence by rendering the virtual world using computer graphics techniques.

Model-based coding of head and shoulder scenes

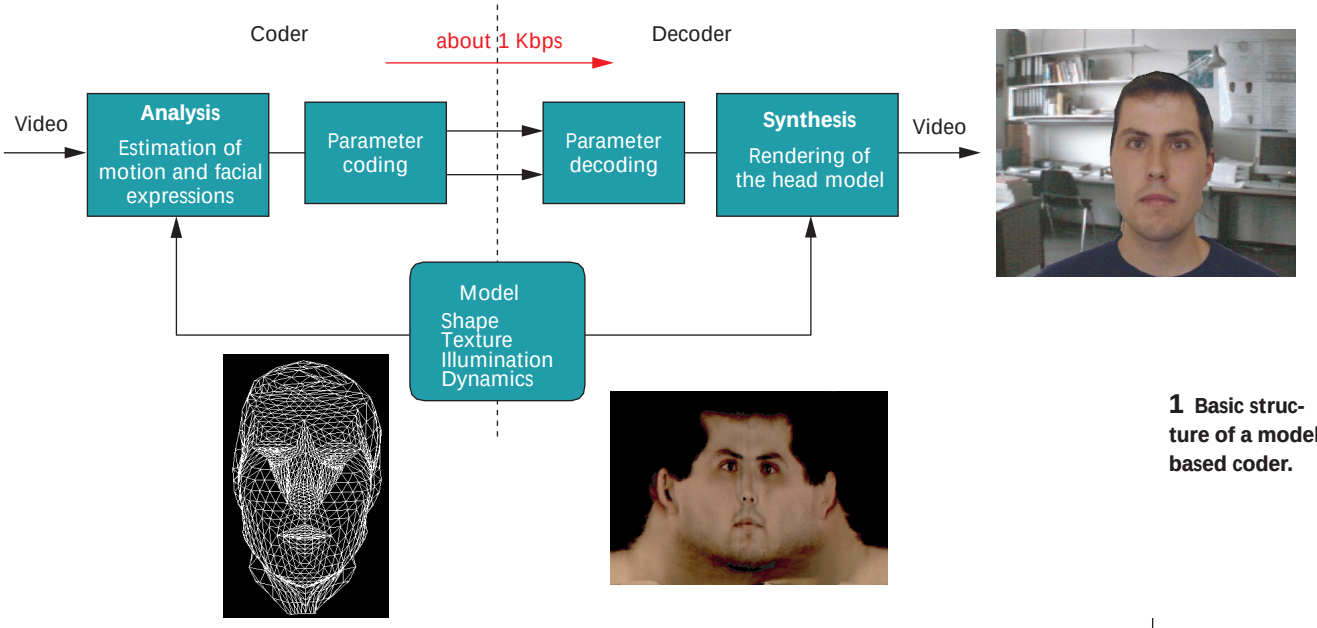
Model-based coding of image sequences requires 3D models for all objects in the scene. Therefore, we must put some restrictions on the scene content. In this article we focus on head and shoulder scenes for video telephone and conferencing applications. Let's see how this process works. A speaking person sits in front of the system while a single camera records video frames. The encoder analyzes the incoming frames and estimates the person's 3D motion and facial expressions. We obtain a set of facial-expression parameters that describe (together with the 3D model) the person's current appearance. We only need to encode and transmit a few parameters, leading to very low data rates (typically less than 1 Kbps). At the decoder we use the parameters to deform our head model according to the person's facial expressions and approximate the original camera frame by rendering our 3D model. Figure 1 shows the model-based coder's principal structure.

We can easily manipulate and interact with the scene content once segmented into single 3D objects. For example, the eye contact problem that occurs in conventional video telephone systems resolves if we position the virtual camera inside the monitor that displays the image of the person with whom we're talking. Or, we can look around objects if we move the synthetic camera according to the eye trajectory the coder estimates.

Virtual conferencing,[3,4] another application for model-based coding, extends the video telephone framework. Here's how it works. Several people are connected to a network talking to each other. A 3D model of each person specifies the person's individual shape, color, and motion characteristics. From the video frames, facial animation parameters (FAPs) are estimated and distributed to all participants. Given the head models and their corresponding motion parameters, we can put all people at a table in a virtual conference room and generate individual views for them.

Here we present a method for estimating a speaking person's FAPs from two successive video frames. This algorithm serves as the basis for the applications mentioned above. First, we show the generic head model used to describe a person's appearance. This 3D model also restricts the possible motions in the face, simplifying the facial parameter estimation. We then propose our gradient-based linear motion estimator, which works hierarchically with low computational complexity. Finally, we validate the algorithm with experimental results for synthetic and real image sequences.

0272-1716/98/$10.00 © 1998 IEEE

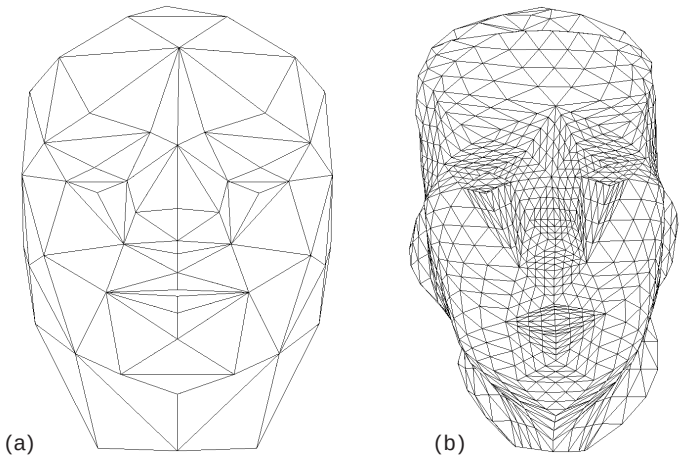


1 Basic structure of a model-based coder.

3D head model

To estimate motion, we can use a simple virtual scene consisting of a head model and camera model. If we want to generate synthetic views of a virtual conference, we have to combine several head models with other 3D models of objects in the room during rendering. However, for estimating FAPs, modeling one person suffices.

The head model in the virtual world specifies a speaking person's 3D shape and color, but also constrains the facial motion caused by expressions. Like other well-known facial models,[5,6] our proposed model consists of a triangular mesh and texture is mapped onto the surface to obtain a photorealistic appearance. The use of second-order triangular B-splines[7] for the definition of the surface's shape facilitates the modeling of facial expressions.



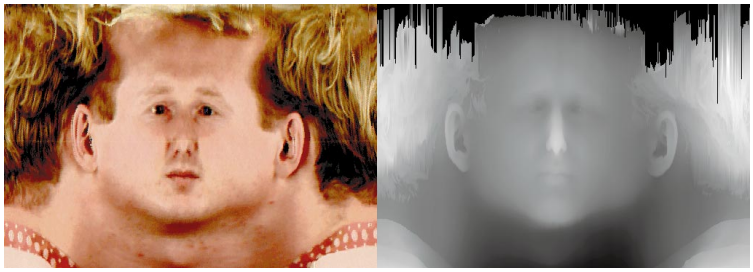
2 (a) Topology of the triangular B-patches. (b) Individually adapted triangular mesh used for rendering and motion estimation.

Our face model is a generic 3D model consisting of 101 triangular B-spline patches for the face and neck area. It also contains 36 rigid patches on the back of the head. Teeth and the mouth's interior form separate 3D objects. The patches' topology (see Figure 2a)—based on the Candide model[6]—and the definition of the FAP units remains fixed for all persons. To adjust the model to an individual person, we need only change the texture and control points' position using information from 3D laser scans. To simplify the rendering, we approximate the adjusted model's spline surface by a number of planar triangles, which results in a triangular mesh. Figure 2b shows an example of an individually adapted and subdivided mesh.

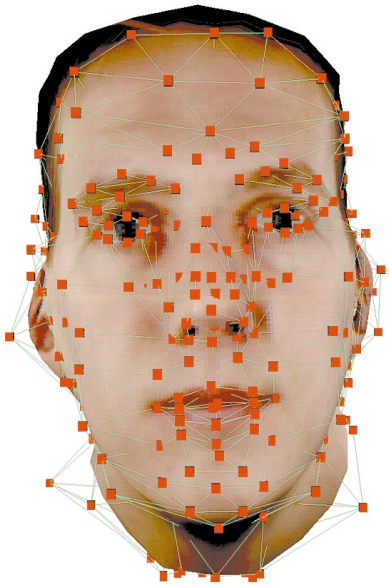
Triangular B-spline-based surface modeling

The set of vertices that define our model's shape has a very large number of degrees of freedom. Therefore, we can model complex objects with sharp edges and discontinuities. A person's face, however, has a smooth surface. Thus facial expressions result in smooth movements of surface points due to the anatomical properties of tissue and muscles. These restrictions on curvature and motion can be modeled by splines, which satisfy specific continuity constraints. Hoch et al.[8] have shown that B-splines model facial skin well. This type of spline exhibits some interesting properties useful for implementing the head model:

- *Smoothness*: A B-spline of order n is C^{n-1} continuous. For our model we use second-order splines leading to C^1 continuity of the surface.
- *Local control*: Movement of a single control point influences the surface just in a local neighborhood, which simplifies modeling facial expressions.
- *Affine invariance*: An affine transformation of the surface results from applying the same transformation on the control points. Facial movements are now defined by the transformation of a small number of



3 Texture and depth information from a 3D laser scanner.



4 Control points of the shaped spline surface.

5 Different synthesized expressions.

control points instead of applying transformations on each vertex, which reduces the computational complexity.

Conventional B-splines suffer from one well-known drawback: since they're defined on a rectangular topology, they don't refine curved areas locally. To overcome this restriction, we use *triangular B-splines*.[7] This spline scheme, based on triangular patches (shown in Figure 2a), can easily be refined while still preserving the above mentioned properties of normal B-splines. For rendering, we approximate the smooth spline and subdivide each patch into a number of planar triangles, leading to a mesh as illustrated in Figure 2b. We can vary the number of triangles to get either a good approximation or higher frame rate during rendering. The resulting triangular mesh is defined by a discrete set of vertices on the mathematically defined spline surface. We need to compute the B-spline basis functions[7] only at the discrete vertex positions on the surface (this can be done offline). Once we have determined the basis functions, we calculate the position of the vertices $\mathbf{v}_j$ by

$$\mathbf{v}_j = \sum_{i \in I_j} N_{ji} \mathbf{c}_i \qquad \textbf{(1)}$$

with N_{ji} the precalculated basis functions of vertex j with

$$\sum_{i \in I_j} N_{ji} = 1 \qquad \textbf{(2)}$$

where I_j is the index set of vertex j and $\mathbf{c}_i$ the ith control point. The index set I_j usually contains three to six indices for the control points that influence the vertex's position.

To individualize the generic model, we use a 3D laser scan of the person with a neutral expression. The scan provides us with information about color and depth as shown in Figure 3. We map the texture on the 3D model and optimize the control points' position to adapt the spline surface to the measured data. After the optimization, shape and texture coincide with the person's appearance. The model can now be deformed to show other expressions rather than the neutral one of the scan.

Modeling facial expressions

To estimate the FAPs requires animating our model to create different facial expressions. We simplified this task by using splines because they already constrain the neighboring vertices' motion. To parameterize the facial expressions, we adapted the MPEG-4 Synthetic-Natural Hybrid Coding (SNHC) group's proposal.[9] According to that scheme, every facial expression can be generated by a superposition of 68 action units. These include both global motion (such as head rotation) and local motion (eye or mouth movement). Currently, 46 of these 68 FAPs are implemented.

To model the local movements, our generic face model contains a table describing how the mesh's con-

trol points are translated or rotated for each FAP. This is done only for the small number of control points shown in Figure 4. For a given set of FAPs that specify a person's expression, all positions of the control points are updated. Then the vertex locations are computed according to Equation 1 by a linear combination of the control points. Figure 5 shows examples of rendered expressions for different persons.

Camera model

The camera model describes the relation between our virtual 3D world and the 2D video frames. It's used for both rendering and parameter estimation.

We use the perspective projection as shown in Figure 6, where the 3D coordinates of an object point $[x\ y\ z]^T$ are projected into the image plane according to

$$X = X_0 - f_x \frac{x}{z}$$
$$Y = Y_0 - f_y \frac{y}{z} \qquad \textbf{(3)}$$

Here, f_x and f_y denote the focal length multiplied by scaling factors in the x- and y-directions, respectively. These scaling factors transform the image coordinates into pixel coordinates X and Y. In addition, they allow the use of non-square pixel geometries. The two parameters X_0 and Y_0 describe the image center and its translation from the optical axis due to inaccurate placement of the charge-coupled device (CCD) sensor in the camera. To estimate the FAPs, we must model the real camera used for recording the video sequence in our synthetic world. The four parameters f_x, f_y, X_0, and Y_0 are therefore obtained from an initial camera calibration using Tsai's algorithm.[10] At the decoder we can then use arbitrary camera parameter settings if we don't want to reconstruct the original image exactly. This lets us zoom into the scene if desired.

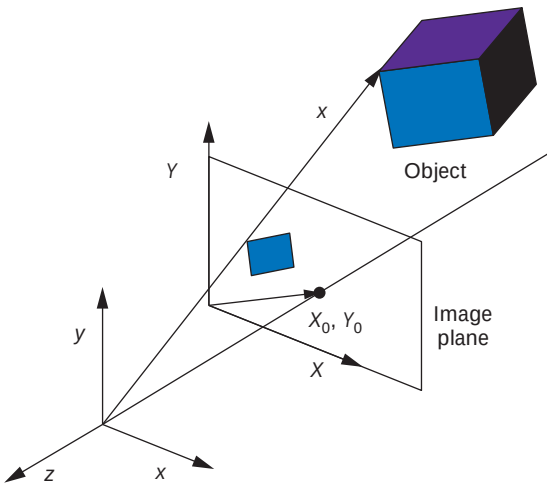
Motion estimation and facial expression analysis

Our motion estimation algorithm analyzes a speaking person's facial expressions by estimating changes of FAPs. We use the whole face image for the estimation, in contrast to feature-based approaches that exploit the information of discrete feature point correspondences. Estimating 3D motion vectors from 2D images proves difficult. The errors of the estimates become very large if we do not determine exactly the position of the small number of features. Therefore, we set up equations at each pixel location of the image, leading to a large number of correspondences used for parameter estimation. To keep the computational complexity low, we developed a linear algorithm for the estimation that solves the large set of equations in a least-squares sense. The small errors that arise due to linearization are corrected using a feedback structure in a hierarchical framework.

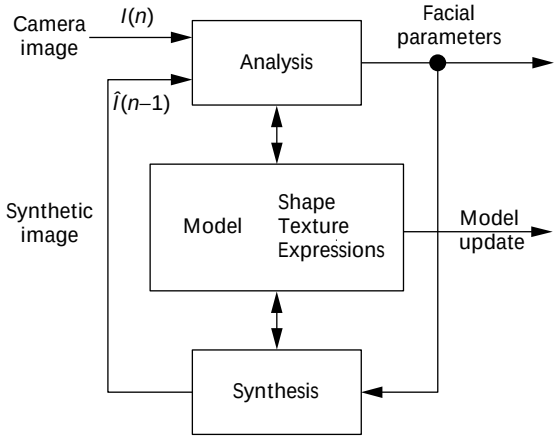
Feedback loop

The motion estimation algorithm presented here estimates the FAPs from two successive frames using the shape and texture information from the 3D model. An easy way of doing this is to estimate the motion between two camera frames and move the 3D head model according to these estimates in the virtual world. However, small errors in the estimation result in placing the 3D model—whose shape information proves necessary for computing the motion—incorrectly. In the next frame, the facial parameters become even worse because the shape does not recover from the model exactly. To avoid error accumulation in the long-term parameter estimation, we use a feedback loop[11] as depicted in Figure 7.

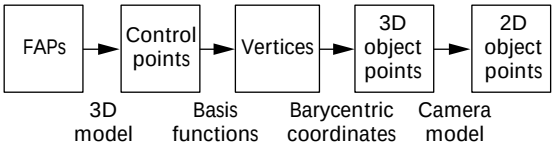
The head's model moves according to the parameters estimated from two successive frames. Rendering the model with a modified shape and position generates a synthetic image. The estimation is then performed between the actual camera frame $I(n)$ and the synthetic image of the previous frame $\hat{I}(n-1)$, which assures that no misalignment of the model occurs. An error in the estimated parameters again leads to placing the model incorrectly. However, the error can be removed during the estimation in the next frame because no mismatch exists between the 3D model and the synthetically rendered image $\hat{I}(n-1)$.



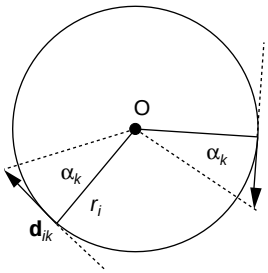
6 Camera model and its associated coordinate systems.



7 Feedback structure of the coder.



8 Transformation from FAPs to image points.



9 Approximation of rotations.

3D motion equation

Estimating the 3D motion of flexible bodies from 2D images proves difficult. However, we can constrain the deformation of most objects to simplify the estimation process. In our algorithm we use a parameterized head model whose local deformations are a function of the 68 unknown FAPs. For each 3D object point of the surface, we can set up one linear 3D motion equation that determines how the point moves in the 3D space if the FAPs change. Instead of the six parameters specifying a rigid body motion, we now have 68 unknowns. Due to the large number of equations for the parameter estimation, we still have an over-determined linear system of equations that we can solve easily.

To set up the 3D motion equation, we must combine several transformations, shown in Figure 8, to take into account the dependencies between the FAPs and the model's 3D object points. First, the surface's control points are moved according to the given FAPs. Using the basis functions of the splines, the algorithm calculates the position of the vertices from the control points. Three vertices form a triangle, and the 3D motion of all object points inside this triangle specified by their barycentric coordinates is determined. Finally, the 2D image point is obtained by projecting the 3D point into the image plane. We incorporated these transformations—all linear except for the projection—into our parameter estimation algorithm.

The new control point position $\mathbf{c}'_i$ can be determined from the position $\mathbf{c}_i$ in the previous frame by

$$\mathbf{c}_i' = \mathbf{c}_i + \sum_k \Delta FAP_k \mathbf{d}_{ik} \tag{4}$$

where

$$\Delta FAP_k = FAP_k' - FAP_k \tag{5}$$

is the change of the facial animation parameter k between the two frames and $\mathbf{d}_{ik}$ the 3D direction vector of the corresponding movement.

Strictly speaking, Equation 4 is valid only for translations. If a number of control points rotate around given axes by some action units, the description for the motion of control points becomes more complicated due to the combination of rotation (defined by rotation matrices) and translation. The order of these operations can no longer be exchanged, and the use of matrix multiplication results in a set of equations that is nonlinear in the parameters that must be estimated. However, we can also use the linear description from Equation 4 for rotation if we assume that the rotation angles between two successive frames are small. Then, the trajectory of a control point i that rotates around the center O can be approximated by its tangent $\mathbf{d}_{ik}$ as shown in Figure 9. This tangent differs for all object points, but we have to set up Equation 4 for all points individually anyhow because of local deformations in the surface.

For a rotation by the angle α_k,

$$\alpha_k = \Delta FAP_k \cdot s_k \tag{6}$$

defined by the facial animation parameter changes ΔFAP_k and the corresponding scaling factor s_k, the length of the translation vector $\mathbf{d}_{ik}$ can be calculated by

$$\left|\mathbf{d}_{ik}\right| = \Delta FAP_k \cdot r_i \cdot s_k \tag{7}$$

Here, r_i is the distance between the object point and the given rotation axis. With this assumption (the direction of $\mathbf{d}_{ik}$ is specified by the direction of the tangent), Equation 4 can also be used for rotation, leading to a simple linear description for all FAPs. Additionally, we can estimate both global and local motion simultaneously. The small error caused by the approximation is compensated after some iterations in the feedback structure shown in Figure 7.

Having modeled the shift in control points, we can determine the motion of the triangular mesh's vertices using Equation 1. The local motion of an object point $\mathbf{x}$ is calculated from that using

$$\mathbf{x} = \sum_{m=0}^{2} \lambda_m \mathbf{v}_m = \sum_{j \in J} \left(\sum_{m=0}^{2} \lambda_m N_{mj} \right) \mathbf{c}_j \tag{8}$$

where λ_m are the barycentric coordinates of the object point in the triangle that encloses that point. The motion equation for a surface point can be represented as

$$\mathbf{x}' = \mathbf{x} + \sum_k \Delta FAP_k \mathbf{t}_k = \mathbf{x} + \mathbf{T} \cdot \Delta \mathbf{FAP} \tag{9}$$

where $\mathbf{t}_k$'s are the new direction vectors to the corresponding FAP calculated from $\mathbf{d}_k$ by applying the linear transforms of Equations 1 and 8. $\mathbf{T}$ combines all the vectors in a single matrix and $\Delta\mathbf{FAP}$ is the vector of all FAP changes. The matrix $\mathbf{T}$ can be derived from the 3D model. Equation 9 then describes the change of the 3D point location $\mathbf{x}' - \mathbf{x}$ as a linear function of FAP changes $\Delta\mathbf{FAP}$. Due to the presence of local deformations, we cannot describe the transformation with the same matrix $\mathbf{T}$ for all object points (as in the case for rigid body motion) but have to set $\mathbf{T}$ up for each point independently.

Motion estimation

For motion estimation, we use the whole face image by setting up the optical flow constraint equation

$$I_X \cdot u + I_Y \cdot v + I_t = 0 \quad \textbf{(10)}$$

where $[I_X, I_Y]$ is the gradient of the intensity at point $[X, Y]$, u and v the velocity in x- and y-direction, and I_t the intensity gradient in the temporal direction. For gray-level images I represents the luminance—using color images leads to three independent equations for the three-color components.

Since u and v can take on arbitrary values for each point $[X, Y]$, this set of equations is under-determined, and we need additional constraints to compute a unique solution. Instead of determining the optical flow field by using smoothness constraints and then extracting the motion parameter set from this flow field, we estimate the FAPs from Equation 10 together with the 3D motion equations of the head's object points.[12,13]

This technique resembles the one described by DeCarlo and Metaxas.[14] One main difference is that we estimate the motion from synthetic frames and camera images using a feedback loop as shown in Figure 7. This allows the use of a hierarchical framework that can handle larger motion vectors between two successive frames. Beyond that, we don't need edge forces to avoid an error accumulation (as described by DeCarlo and Metaxas[14]) with the rendering feedback loop. Another difference between the two approaches is that we use a textured 3D model, which lets us generate new synthetic views of our virtual scene after estimating the motion.

Writing Equation 9 in its single components leads to

$$x' = x\left(1 + \frac{1}{x}\mathbf{t}_x \cdot \Delta\mathbf{FAP}\right) \quad \textbf{(11)}$$

$$y' = y\left(1 + \frac{1}{y}\mathbf{t}_y \cdot \Delta\mathbf{FAP}\right) \quad \textbf{(12)}$$

$$z' = z\left(1 + \frac{1}{z}\mathbf{t}_z \cdot \Delta\mathbf{FAP}\right) \quad \textbf{(13)}$$

$\mathbf{t}_x$, $\mathbf{t}_y$, and $\mathbf{t}_z$ are the row vectors of matrix $\mathbf{T}$. Dividing Equations 11 and 12 by Equation 13, inserting the camera model from Equation 3, and using a first-order approximation leads to

$$u = X' - X \approx -\frac{1}{z}\left(f_x\mathbf{t}_x + \left(X - X_0\right)\mathbf{t}_z\right)\Delta\mathbf{FAP} \quad \textbf{(14)}$$

$$v = Y' - Y \approx -\frac{1}{z}\left(f_y\mathbf{t}_y + \left(Y - Y_0\right)\mathbf{t}_z\right)\Delta\mathbf{FAP} \quad \textbf{(15)}$$

This equation serves as the motion constraint in the 2D image plane. Together with Equation 10 a linear equation at each pixel position can be set up:

$$\frac{1}{z}[I_X f_x\mathbf{t}_x + I_Y f_y\mathbf{t}_y + \left(I_X\left(X - X_0\right) + I_Y\left(Y - Y_0\right)\right)\mathbf{t}_z]\Delta\mathbf{FAP} = I_t \quad \textbf{(16)}$$

with z being the depth information coming from the model. We obtain an over-determined system that can be solved in a least-squares sense with low computational complexity. The system's size depends directly on the number of FAP.

Outlier removal

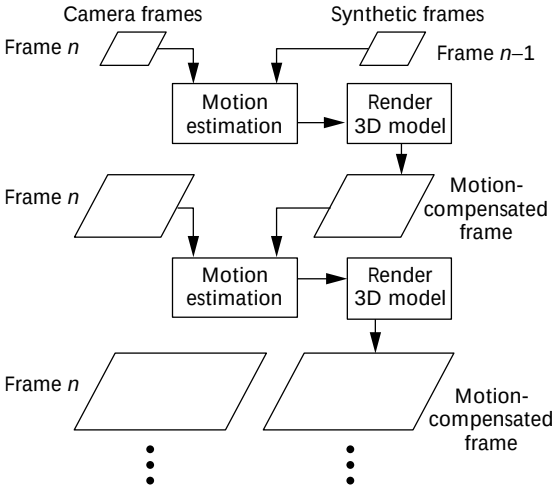
At each pixel position of the object, we can set up Equation 16 leading to a large number of equations. We need at least the same number of equations as the number of estimated FAPs, but due to the large number of face pixels—each of which contributes one additional equation—we can discard some possible outliers. These outliers can be detected by analyzing the partial derivatives of the intensity and the motion model. The optical flow constraint of Equation 10 is only valid for small displacement vectors due to the linearization of the intensity values. If the estimate of the displacement vector length for the pixel at position $[X, Y]$,

$$\hat{d}\left(X,Y\right) = \sqrt{\frac{I_t^2}{I_X^2 + I_Y^2}} \quad \textbf{(17)}$$

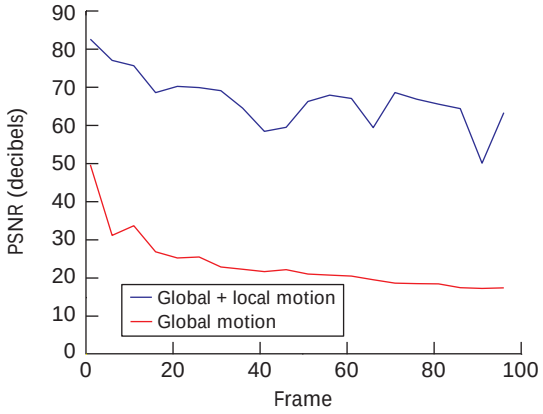
is larger than a threshold G, we classify the pixel as an outlier and don't use it for motion estimation.

Hierarchical motion estimation

The inherent linearization of the intensity in the optical flow constraint and the approximations used to obtain a linear solution prevent dealing with large displacement vectors between two successive video frames. To overcome this limitation, we use a hierarchical scheme for the motion estimation as shown in Figure 10. First, an approximation for the motion parameters between frame n and $n - 1$ is computed from low-pass



10 Image pyramid of the hierarchical motion estimation scheme.



11 PSNR after global and global plus local motion compensation.

12 Camera frames (left) and corresponding reconstructed synthetic frames (right).

filtered and subsampled versions of the camera frame $I(n)$ and the synthetic frame $\hat{I}(n-1)$. Due to using low-resolution images, the linear intensity assumption remains valid over a wider range. For subsampling, simple moving average filters reduces aliasing. A Gauss filter to smooth the edges before estimating the motion further filters the resulting images. The estimated parameter set generates a motion-compensated image by moving the 3D model and rendering it at the new position. Due to the motion compensation, the differences between the new synthetic image and camera frame decrease. Then, the procedure is repeated at higher resolutions, each time yielding a more accurate motion parameter set. Note that we use the same camera frame in all levels, and that the synthetic image is changed from $\hat{I}(n-1)$ to $\hat{I}(n)$ iteratively. In our current implementation we use three levels of resolution starting from 88×72 pixels. For each new level the resolution doubles in both directions, leading to a final Common Intermediate Format (CIF) resolution of 352×288 pixels. At the same time, the threshold G used to detect outliers for the motion estimation reduces from 5 (first level) to 0.5 (highest resolution), which means that at higher levels more pixels become classified as outliers. Experiments with this hierarchical scheme showed that it can estimate displacements of up to 30 pixels between two frames.

Experimental results

To validate the estimation algorithm, we performed experiments with synthetic and real video. To create the synthetic sequences, we rendered the 3D model with known FAPs, which let us compare the estimated FAPs with original ones. For real sequences recorded with a camera, we had to individualize the generic 3D model to the person in the video using a 3D laser scan of that person. Unfortunately, we couldn't determine the accuracy of the estimated FAP values for real image sequences, since we didn't know their correct values. Therefore, we could only compare the reconstructed synthetic image with the original camera frame. The error measure that we used for that purpose is the peak signal noise ratio (PSNR), defined as

$$PSNR = 10 \log \left(\sum_{i=0}^{N-1} \frac{255^2}{\left(I_{orig,i} - I_{synth,i} \right)^2} \right) \quad \textbf{(18)}$$

In this equation $I_{orig,i}$ is the luminance component of pixel i of the original camera image with values from 0 to 255, and $I_{synth,i}$ is the corresponding value in the synthetic image. N denotes the number of pixels in the image.

Synthetic sequence

In the first experiment we created a synthetic sequence (100 frames) with a resolution of 352×288 pixels (CIF resolution). Rendering the 3D model with well-defined FAPs and a viewing angle of about 25 degrees achieves this. We estimated 10 FAPs for every fifth frame using the proposed algorithm and compared the results to the correct values. Table 1 shows the relative error averaged over all estimates. The values in the table were measured relative to the maximum of each value that corresponded to an extreme expression. In this experiment we used the FAPs for global motion of the head (FAPs 48, 49, 50), opening of the jaw (FAP 3), and movements of eyelids (FAPs 19, 20), eyebrows (FAPs 35, 36), and lip corners (FAPs 12, 13).

The accurate values for the estimated FAPs also resulted in an excellent reconstruction of the original frames. Because we used the same model for creating the sequence and estimating the parameters, we achieved a nearly perfect reconstruction. You can see this in Figure 11, where we plot the measured PSNR between the original and synthetic image. Averaging the PSNR values computed only in the facial area and not for the background leads to a value of about 70 decibels. For comparison, Figure 11 also shows the PSNR for the reconstructed sequence we generated using only global motion parameters (head translation and rotation).

Table 1. Relative average error of the estimated FAPs (in percentages).

FAP	3	12	13	19	20	35	36	48	49	50
Relative error	0.26	0.17	0.19	0.11	0.07	0.04	0.02	0.04	0.41	0.20

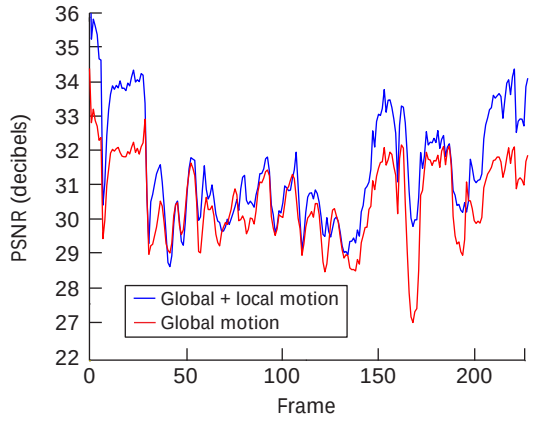
Camera sequences

In a second experiment, we recorded a video of a talking person with a camera in CIF resolution and a frame rate of 12.5 Hz. We estimated the FAPs for all 230 frames and rendered the corresponding synthetic sequence. We estimated 17 parameters including global head motion (six parameters) and movements of eyebrows (four parameters) and mouth motion (seven parameters). The total processing time for each frame was about 10.8 seconds on a 175-MHz Silicon Graphics O2 workstation; 3.03 seconds of the time was spent for the motion estimation algorithm, including setting up the equations and solving for the unknowns. Rendering the model and accessing files took up the remaining time. Figure 12 shows three frames of the camera sequence and the synthesized frames from the estimated parameters.

Since we used real camera sequences, we had to measure the estimation's accuracy by comparing the 2D images from the camera and the renderer. The PSNR between original and synthetic images computed in the facial area averaged 31.6 dB. Figure 13 shows the PSNR for each frame when using 17 parameters for the estimation (blue) or when performing only global motion estimation (six parameters). Comparable results are also achieved for other sequences. Figure 14 shows the original and synthetic views for a second sequence.

To estimate the bit rate needed for transmitting a head and shoulders sequence over a network, we used a Huffman coder to encode the FAPs. The 17 FAPs are predicted from the previous frame while the prediction error is quantized with 7 bits and then Huffman coded. This leads to an average bit rate of about 0.58 Kbps, which corresponds to about 47 bits per frame. This calculation assumes that the decoder has already received the model.

Once we have estimated the FAPs for our 3D model, we can use them to create virtual environments. Figure 15 shows an example for such a manipulation. Here, we recorded a sequence of a speaking person (left side) and estimated the FAPs. These parameters were transmitted with an average bit rate of about 0.57 Kbps. The decoder then rendered the 3D model together with a description of the background leading to the images shown on the right side of Figure 15.



13 PSNR of a coded talking head sequence.

14 Camera frames (left) and corresponding reconstructed synthetic frames (right).

15 Camera frames (left) and corresponding reconstructed synthetic frames in a virtual environment (right).

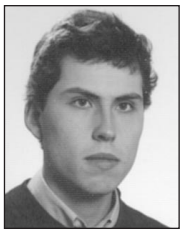
Conclusions

In this article we presented a method for the estimation of FAPs from 2D image sequences. The combination of the optical flow constraint with 3D motion models lead to a robust and low complexity algorithm. Experiments with real video sequences showed that we can achieve bit rates as low as 0.6 Kbps at 31.6 dB PSNR.

Currently, we're investigating the influence of illumination variation that violates the optical flow constraint. To reduce this influence, we add illumination models to our virtual scene and estimate both motion and illumination parameters like light direction and intensity.[15] Also, we're working on adding temporal 3D motion constraints to the estimation framework to further increase its robustness. ■

References

1. D.E. Pearson, "Developments in Model-Based Video Coding," *Proc. IEEE*, Vol. 83, No. 6, June 1995, pp. 892-906.
2. W.J. Welsh, S. Searsby, and J.B. Waite, "Model-Based Image Coding," *British Telecom Technology J.*, Vol. 8, No. 3, July 1990, pp. 94-106.
3. K. Aizawa and T.S. Huang, "Model-Based Image Coding: Advanced Video Coding Techniques for Very Low Bit-Rate Applications," *Proc. IEEE*, Vol. 83, No. 2, Feb. 1995, pp. 259-271.
4. I.S. Pandzic et al., "Towards Natural Communication in Networked Collaborative Virtual Environments," *Proc. Framework for Immersive Environments (FIVE) 96*, 1996, http://ligwww.epfl.ch/dt/papers.dir/five96B.html.
5. F.I. Parke, "Parameterized Models for Facial Animation," *IEEE CG&A*, Vol. 2, No. 9, 1982, pp. 61-68.
6. M. Rydfalk, *Candide: A Parameterized Face*, LiTH-ISY-I-0866, Image Coding Group, Linköping Univ., Linköping, Sweden, Oct., 1987.
7. G. Greiner and H.-P. Seidel, "Modeling with Triangular B-Splines," *Proc. ACM/IEEE Solid Modeling Symp. 93*, 1993, ACM Press, New York, pp. 211-220.
8. M. Hoch, G. Fleischmann, and B. Girod, "Modeling and Animation of Facial Expressions Based on B-Splines," *Visual Computer*, Vol. 11, 1994, pp. 87-95.
9. *SNHC Systems Verification Model 4.0*, ISO/IEC JTC1/SC29/WG11 N1666, Bristol, Great Britain, April 1997.
10. R.Y. Tsai, "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses," *IEEE J. of Robotics and Automation*, Vol. RA-3, No. 4, Aug. 1987, pp. 323-344.
11. R. Koch, "Dynamic 3D Scene Analysis through Synthesis Feedback Control," *IEEE Trans. PAMI*, Vol. 15, No. 6, June 1993, pp. 556-568.
12. H. Li, P. Roivainen, and R. Forchheimer, "3D Motion Estimation in Model-Based Facial Image Coding," *IEEE Trans. PAMI*, Vol. 15, No. 6, June 1993, pp. 545-555.
13. J. Ostermann, "Object-Based Analysis-Synthesis Coding (OBACS) Based on the Source Model of Moving Flexible 3D Objects," *IEEE Trans. Image Processing*, Vol. 3, No. 5, Sept. 1994, pp. 705-711.
14. D. DeCarlo and D. Metaxas, "The Integration of Optical Flow and Deformable Models with Applications to Human Face Shape and Motion Estimation," *Proc. Computer Vision and Pattern Recognition (CVPR) 96*, IEEE CS Press, Los Alamitos, Calif., 1996, pp. 231-238.
15. P. Eisert and B. Girod, "Model-Based Coding of Facial Image Sequences at Varying Illumination Conditions," *Proc. 10th Image and Multidimensional Digital Signal Processing Workshop 98*, IEEE Press, Piscataway, N.J., July 1998, pp. 119-122.



Peter Eisert *joined the image communication group at the Telecommunications Institute of the University of Erlangen-Nuremberg, Germany in 1995. As a member of the Center of Excellence's 3D Image Analysis and Synthesis group, he is currently working on his PhD thesis. His research interests include model-based video coding, image communication, and computer vision. He received his diploma in electrical engineering from the University of Karlsruhe, Germany in 1995.*

Bernd Girod *is a chaired professor of telecommunications in the Electrical Engineering Department of the University of Erlangen-Nuremberg. His research interests are in image communication, 3D image analysis and synthesis, and multimedia systems. He has been the director of the Center of Excellence's 3D Image Analysis and Synthesis group in Erlangen since 1996. He received an MS from Georgia Institute of Technology, Atlanta, and a PhD in engineering from the University of Hannover, Germany.*

Readers may contact the authors at the Telecommunications Laboratory, University of Erlangen, Cauerstrasse 7, D-91058, Erlangen, Germany, e-mail {eisert, girod}@nt.e-technik.uni-erlangen.de.