Journal for Geometry and Graphics
Volume 13 (2009), No. 2, 209–220.

A 3D Head Model From Stereo Images by a Self-Organizing Neural Network

Levente Sajó[1], Miklós Hoffmann[2], Attila Fazekas[1]

[1]*Faculty of Informatics, University of Debrecen*
P.O. Box 12, H-4010 Debrecen, Hungary
email: sajo.levente@inf.unideb.hu, fazekas.attila@inf.unideb.hu

[2]*Institute of Mathematics and Informatics, Eszterházy Károly College*
Leányka str. 4, H-3300 Eger, Hungary
email: hofi@ektf.hu

Abstract. In this paper a model based approach is presented for generating 3D face models from two stereo images. The proposed method works with a small number of feature points of the face. The automatically registrated point pairs are used to reconstruct their 3D correspondence, then a predefined general face model is adjusted to the reconstructed 3D points. Fitting the general face model to the feature points is made iteratively by applying a self-organizing neural network.

Key Words: 3D face model, stereo images, neural network

MSC 2000: 68U05, 51N05, 51N20

1. Introduction

Generating a 3D model from pictures is a challenging task and an active area of computer graphics research since decades. Although there are different possible views of this task, here we only deal with the problem where a complete 3D structure should be obtained from the 2D data. For a recent overview of different methods see, e.g., [5].

All methods for facial reconstruction from stereo images typically consist of three main steps: camera calibration, establishing point correspondences between pairs of points from the left and the right image and reconstruction of the 3D coordinates of the points (i.e., the head model) in the scene.

In several earlier approaches the process of generating a head model of a specific person entails extensive user intervention [14]. More recent works give more or less automatic reconstruction using two pictures captured from mutually orthogonal directions: typically a frontal and a profile photo [13] or even more photos or video sequences [4].

Here we focus on the problem of generating the 3D model from two stereo images captured from general viewpoints. A recently used method is a model-based approach, where a

ISSN 1433-8157/$ 2.50

predefined general face model is adjusted to the reconstructed 3D points. One of the crucial steps of reconstruction is to generate as much point correspondences as possible to obtain an algebraic adjustment with acceptable accuracy. These methods use the disparity map, the measure of the displacement of relative features, to register hundreds of corresponding point pairs on the stereo images (see [18, 4]).

In the proposed method we use only very few points of the face (the feature points) to reconstruct the head. The registration of point pairs is completely automatic as well as the reconstruction of the 3D points. Due to the relatively small number of points the general head model cannot be adjusted by simple algebraic methods, so here we apply self-organizing neural network [10] to iteratively fit the general model to the feature points. The method is fast and the head model is easily adjustable by different face characteristics and expressions.

2. Registration of the feature points

Registration of the feature points on faces is a pattern recognition problem. In general it can be considered as an object detection task. Many object detection techniques have been invented in the recent years and have been collected in different surveys [17, 6, 9]. These methods have been classified in four categories:

- *knowledge based methods*,
- *feature invariant approaches*,
- *template matching methods*, and
- *appearance-based methods*.

The most successful ones were appearance based methods. These rely on the same concept: scanning over the image with a window of fixed size, selecting the small image patches which are then passed to classifiers. For face localization we choose *Haar-classifier based boosted cascade detector* introduced by Viola and Jones in 2001 [16], because it has the same accuracy as the other techniques but exceeds them in speed. It can be applied in case of arbitrary objects having complex and varying textures, e.g., faces. The basic idea of this classifier is that the recognition process can be much more efficient if it is based on the detection of features that encode some information about the class to be detected. This is the case of Haar-like features that encode the existence of oriented contrasts between regions in the image. A set of these features can be used to encode the contrasts exhibited by a human face and their spacial relationships. Haar-like features are so called because they are computed similar to the coefficients in Haar wavelet transforms.

Facial features are generally stable points from the faces, like corners of the eyes, corners of the eyebrows, corners of the nostrils, tip of the nose, corners of the lips. Since the texture of these points is quite simple and contains little gray level information, usual object detection techniques can not deliver good results. They should be combined also with some heuristics about the estimated place of these features and their relative locations to each other. The *Active Appearance Model* (AAM), proposed first by Cootes in 1998 [2], is a method which uses both shape and appearance for detecting the precise location of facial features. In our project, we have used a variation, called *independent AAM*, described in [12], because it has demonstrated a good performance in speed and therefore it is possible to be used in real time applications. From this point up to the spatial reconstruction we are dealing with shapes and information obtained from 2D images.

2.1. Independent AAM

Independent AAM also contains a shape and appearance model but in contrast to Cootes' AAM, it treats them separately. In our case the landmark points are the facial feature points (see Fig. 1.)

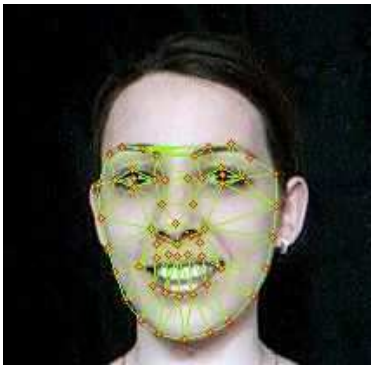

Figure 1: The selected feature points on an image from our database

Building shape and appearance models is done similarly. In case of the shape model the coordinates of the landmark points (vertices) are aligned into a common coordinate frame and organized into a vector. If the coordinates of the vertices are $\mathbf{v}_i = (x_i, y_i)$, $i = 1, \ldots, v$, then let's define a shape $\mathbf{s}$ as

$$\mathbf{s} = (x_1, y_1, x_2, y_2, \ldots, x_v, y_v)^T. \tag{1}$$

Applying PCA on these vectors, if the mean shape is considered as a *base shape* and eigenvectors to the n largest eigenvalues as *shape vectors*, then the shape $\mathbf{s}$ can be expressed as a base shape $\mathbf{s}_0$ plus a linear combination of the n shape vectors $\mathbf{s}_i$

$$\mathbf{s} = \mathbf{s}_0 + \sum_{i=1}^{n} p_i \, \mathbf{s}_i, \tag{2}$$

where p_i are the *shape parameters*.

Before sampling the grey level information, each training image is shape normalized so that its control points match the base shape. For shape normalization linear shape transformations and global shape transformations are used.

For performing the linear shape normalization, a triangulated mesh is defined to the shape model, which describes the connectivity between vertices. If we take a pair of meshes $\mathbf{s}_0$ and $\mathbf{s}$ a piecewise affine warp can be defined between them. For each triangle in $\mathbf{s}_0$ there is a corresponding triangle in $\mathbf{s}$ such that the vertices of the first triangle map to the vertices of the second triangle. Piecewise affine warp will be denoted with $W(\mathbf{x}, \mathbf{p})$. For a pixel $\mathbf{x} = (x, y)^T$ in $\mathbf{s}_0$ the corresponding pixel in $\mathbf{s}$ is $\mathbf{x}' = W(\mathbf{x}, \mathbf{p})$, ($\mathbf{p}$ is the vector of shape parameters p_i).

The linear shape normalization is followed by global shape transformations of 2D similarity transformations (translation, rotation and scale). Instead of these 2D similarity transformations one can define a global warp which has equal effects on the mesh. Let's denote the global shape normalizing warp with $N(\mathbf{x}, \mathbf{q})$. Suppose the base mesh is $\mathbf{s}_0 = (x_1^0, y_1^0, \ldots, x_v^0, y_v^0)^T$, then the shape vectors can be chosen as $\mathbf{s}_1 = (1, 0, \ldots, 1, 0)^T$, $\mathbf{s}_2 = (0, 1, \ldots, 0, 1)^T$, $\mathbf{s}_3 = \mathbf{s}_0 = (x_1^0, y_1^0, \ldots, x_v^0, y_v^0)^T$, $\mathbf{s}_4 = (y_1^0, -x_1^0, \ldots, y_v^0, -x_v^0)^T$. This results in a global shape normalizing

warp $N(\mathbf{x}, \mathbf{q})$ with parameters $\mathbf{q} = (q_1, q_2, q_3, q_4)$ which works similarly to the linear shape warp $W(\mathbf{x}, \mathbf{p})$.

In the following, let's denote with $T(\mathbf{x}, \mathbf{pq})$ the consecutive execution of the two warps described above, where $\mathbf{pq}$ means the concatenation of the two parameter vectors:

$$T(\mathbf{x}, \mathbf{pq}) = N(W(\mathbf{x}, \mathbf{p}), \mathbf{q}), \quad \mathbf{pq} = (\mathbf{p}, \mathbf{q}). \tag{3}$$

The shape normalized image with $T(\mathbf{x}, \mathbf{pq})$, is used to sample the gray level information (appearance) within the base mesh. A vector of these gray values is formed on which PCA is applied. Over the $\mathbf{x} = (x, y)^T$ pixels from inside the base mesh $\mathbf{s}_0$ can be defined the appearance $A(\mathbf{x})$ of AAM. Then applying PCA results in the following:

$$A(\mathbf{x}) = A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbf{s}_0, \tag{4}$$

where λ_i are the *appearance parameters*.

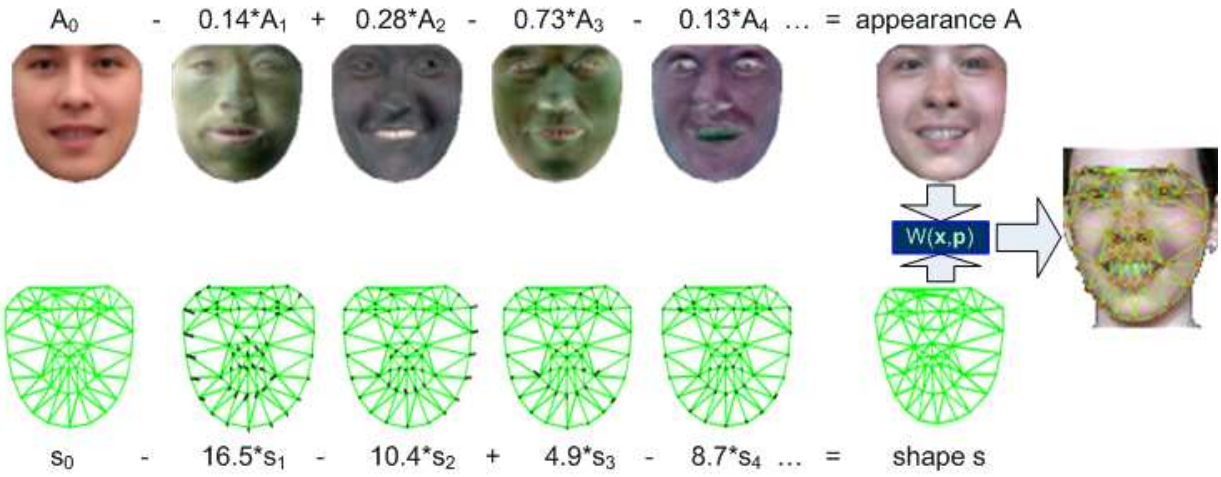

Figure 2: An instance of the AAM results from base models plus a linear combination of transformation vectors

Given an input image $I(\mathbf{x})$, the goal of the fitting process is to find out the optimal $\mathbf{p}$, $\mathbf{q}$ and λ parameters to minimize the error between the input image and the model instance. This will be done in the coordinate frame of the AAM. If $\mathbf{x}$ is a pixel in $\mathbf{s}_0$ then the minimum of the following equation is determined:

$$\sum_{\mathbf{x}} [A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(T(\mathbf{x}, \mathbf{pq}))]^2. \tag{5}$$

Many different algorithms can be used to minimize the equation. Probably, the most straightforward one is the standard gradient descent algorithm. The problem with this algorithm is that it is slow. A better solutions is using the Inverse Compositional Algorithm described in [12]. This algorithm has two main steps. The first one is the image alignment. It works on a constant template image. The best location of the constant template image is determined in an input image by using linear shape variations followed by global shape transformation. In the second step the appearance variation is included. In the following, these steps will be presented in more detail.

2.1.1. Inverse compositional image alignment

Let's denote the constant template image with $A_0(x)$. Then Eq. (5) changes to

$$\sum_{\mathbf{x}} [A_0(\mathbf{x}) - I(T(\mathbf{x}, \mathbf{pq}))]^2. \tag{6}$$

This equation should be minimized with respect to the warp parameters $\mathbf{p}$. This is a nonlinear optimization problem. If we assume that the initial estimate of $\mathbf{p}$ is known, then the problem can be linearized by iteratively minimizing for the parameters $\Delta\mathbf{p}$

$$\sum_{\mathbf{x}} [A_0(\mathbf{x}) - I(T(\mathbf{x}, \mathbf{pq} + \Delta\mathbf{pq}))]^2, \tag{7}$$

and then updating $\mathbf{p}$ by

$$\mathbf{pq} \leftarrow \mathbf{pq} + \Delta\mathbf{pq}. \tag{8}$$

Instead of solving this equation, in an inverse compositional algorithm a variation of (7) is used, where the roles of template and example image are reversed. The incremental warp is computed with respect to the template A_0

$$\sum_{\mathbf{x}} [I(T(\mathbf{x}, \mathbf{pq})) - A_0(T(\mathbf{x}, \Delta\mathbf{pq}))]^2. \tag{9}$$

In every iteration, first $\Delta\mathbf{p}$ is determined and then the warp parameter is updated using

$$T(\mathbf{x}, \mathbf{pq}) \leftarrow T(\mathbf{x}, \mathbf{pq}) \circ T(\mathbf{x}, \Delta\mathbf{pq})^{-1}. \tag{10}$$

For determining $\Delta\mathbf{pq}$ the Taylor series expansion of Eq. (9) is taken:

$$\sum_{\mathbf{x}} [I(T(\mathbf{x}, \mathbf{pq})) - A_0(T(\mathbf{x}, 0)) - \nabla A_0 \frac{\partial T}{\partial \mathbf{pq}} \Delta\mathbf{pq}]^2. \tag{11}$$

From this $\Delta\mathbf{pq}$ can be expressed in the following form:

$$\Delta\mathbf{pq} = \mathbf{H}^{-1} \sum_{\mathbf{x}} [SD(\mathbf{x})]^T [E(\mathbf{x})], \tag{12}$$

where E($\mathbf{x}$) is the error image

$$E = I(T(\mathbf{x}, \mathbf{pq})) - A_0(T(\mathbf{x}, 0)), \tag{13}$$

SD($\mathbf{x}$) is the steepest descent image

$$SD(\mathbf{x}) = \nabla A_0 \frac{\partial T}{\partial \mathbf{pq}}, \tag{14}$$

and H is the Gauss-Newton approximation of the Hessian matrix

$$H = \sum_{\mathbf{x}} [SD(\mathbf{x})]^T [SD(\mathbf{x})]. \tag{15}$$

The benefit of this approach is that, since A_0 is a constant template image and $\frac{\partial T}{\partial \mathbf{pq}}$ is evaluated at $\mathbf{pq} = 0$ (identity transform), the computation of the Hessian matrix of the objective function can be moved into a pre-computation step.

2.1.2. Including appearance variation

By now, we have presented all the steps to fit a constant template image to an input image. The last remaining step is to include appearance variations. For this, rewrite Eq. (5) as

$$\sum_{\mathbf{x}} [A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(T(\mathbf{x}, \mathbf{pq}))]^2 = \left\| A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(T(\mathbf{x}, \mathbf{pq})) \right\|^2 =$$

$$\left\| A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(T(\mathbf{x}, \mathbf{pq})) \right\|^2_{span(A_i)^\perp} + \left\| A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(T(\mathbf{x}, \mathbf{pq})) \right\|^2_{span(A_i)}, \tag{16}$$

where $\|.\|$ is the L_2 norm, $span(A_i)$ denotes a linear subspace spanned by a collection of vectors A_i and $span(A_i)^\perp$ its orthogonal complement. The first term can be simplified:

$$\|A_0(\mathbf{x}) - I(T(\mathbf{x}, \mathbf{pq}))\|^2_{span(A_i)^\perp} + \left\| A_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(T(\mathbf{x}, \mathbf{pq})) \right\|^2_{span(A_i)}; \tag{17}$$

for any $\mathbf{pq}$ the minimum of the second term is always 0. Therefore the minimum value can be found in two steps. First, the first term should be minimized with respect to $\mathbf{pq}$ by applying the inverse compositional algorithm with no appearance variation. The only difference is that it should be done in the linear subspace of $span(A_i)^\perp$ rather than in the full vector space. What is needed to do is to project the steepest descent image into that subspace, using

$$\nabla A_0 \frac{\partial T}{\partial \mathbf{pq}} - \sum_{i=1}^{m} \left[\sum_{\mathbf{x}} A_i(\mathbf{x}) \nabla A_0 \frac{\partial T}{\partial \mathbf{pq}} \right] A_i(\mathbf{x}) \tag{18}$$

In the second step, by using the optimal value of $\mathbf{pq}$ as a constant, the minimum value of the second term is computed with respect of λ:

$$\lambda_i = \sum_{\mathbf{x}} A_i(\mathbf{x}) \, [I(T(\mathbf{x}, \mathbf{pq})) - A_0(\mathbf{x})]. \tag{19}$$

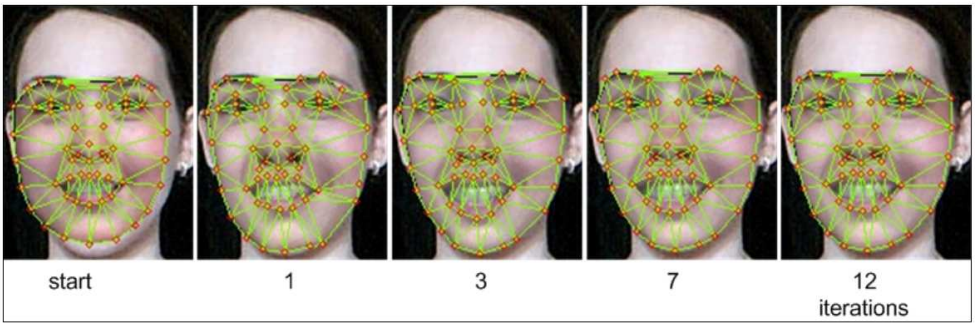

Figure 3: The convergence of the fitting process

2.1.3. The algorithm of feature points registration

Pre-computation:

- Evaluate the gradient ∇A_0 of the template $A_0(\mathbf{x})$;
- Evaluate the Jacobian $\frac{\partial T}{\partial \mathbf{pq}}$ at $(\mathbf{x}, \mathbf{0})$;
- Compute the modified steepest descent image using Eq. (18);
- Compute the Hessian matrix using Eq. (15).

Iterate:

- Warp I with $T(\mathbf{x}, \mathbf{pq})$ to compute $I(T(\mathbf{x}, \mathbf{pq}))$;
- Compute the error image $E(\mathbf{x})$ using Eq. (13);
- Compute $\Delta\mathbf{pq}$ by multiplying the inverse Hessian and the dot product of the modified steepest descent image with the error image using Eq. (12);
- Update $\mathbf{pq}$ using Eq. (10).

Post-computation:

- Compute λ_i using Eq. (19).

3. 3D reconstruction

After registration of the 2D facial feature points on both images, the 3D coordinate of these points are calculated using a 3D reconstruction method. For this, first, in a calibration step the intrinsic and extrinsic parameters of the camera should be calculated. Then, the coordinates of the 3D point is given by the intersection of the two lines, which connect the focuses of the cameras and the 2D points. For the geometrical background and detailed description of the reconstruction see [11] and [15].

3.1. Camera calibration

The aim of the calibration is to determine the orientation parameters for both cameras (left – index l, right – index r). These parameters are either intrinsic or extrinsic parameters. The intrinsic parameters are the following:

- focal lengths: f_l and f_r; Since the x and y size of the camera pixels are not equal, we compute with focal lengths measured separately in x and y directions: $(fx_l,\ fy_l)$ for the left and $(fx_r,\ fy_r)$ for the right camera;
- the pixel coordinates of the projection of focus to the image: $(c_l,\ r_l)$ and $(c_r,\ r_r)$;
- distortion parameters.

The extrinsic parameters describe the connection between the coordinate system of the camera (its center is the focus, the z-axis is perpendicular to the screen, x and y-axis are parallel to pixel axis) and the world coordinate system. These can be written by a rotation matrix (R_l and R_r) and a shifting vector (t_l and t_r). The connection between the two cameras coordinate systems is given by (R_{cam}, t_{cam}), where the left camera coordinate system is used to describe the right camera coordinate system. These parameters can be expressed with the combination of first two: $R_{cam} = R_l R_r^T,\ t_{cam} = t_l - R_{cam} t_r$.

If $P(x_1, x_2, x_3)$ is a point in the world coordinate system and its projection is $P_l(xl_1,\ xl_2)$ on the left and $P_r(xr_1,\ xr_2)$ on the right cameras (in pixel coordinates) then the connection between them can be described using standard equations (c.f. [11]). For determining these parameters, many different softwares can be found on the web. Usually, the common idea is that a chessboard-like shape is moved to different angles in front of the two cameras and pictures are taken. The corresponding pixel-coordinate pairs (the edges of the black and white squares) are used by the calibration software to calculate the intrinsic and extrinsic parameters of the camera. As calibration software we have used [1].

3.2. Reconstruction

The goal of the reconstruction is to calculate the coordinates of $P(x_1, x_2, x_3)$ using the two corresponding 2D points: $P_l(xl_1, xl_2)$ and $P_r(xr_1, xr_2)$. For this, we have used a method based on ray-casting: the two lines, which connect the focuses of the cameras and the 2D points, should intersect in P. In practice, because of errors (distortion), usually these are skew lines and their transversal has to be determined and the center of this transversal is considered as the location of P (see Fig. 4).

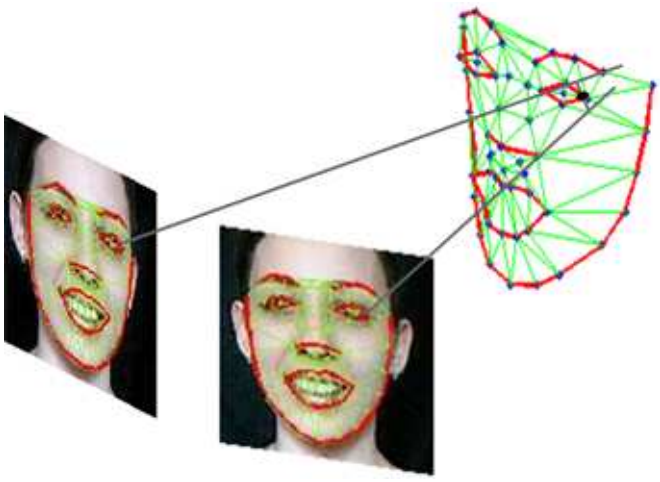

Figure 4: Reconstructing a 3D point using the two corresponding 2D points

The coordinates of $P_l(xl_1,\ xl_2)$ in the left camera's coordinate system can be expressed as $P_{lcam}(xlcam_1, xlcam_2, xlcam_3)$, where third coordinate is the focal lenght $xlcam_3 = f_l$, and the first two coordinates can be calculated using equations:

$$xlcam_1 = \frac{(xl_1 - c_l)}{fx_1} f_l, \quad xlcam_2 = \frac{(xl_2 - r_l)}{fx_2} f_l.$$

The vector pointing from the origin of the left camera's coordinate system is

$$\mathbf{v}_l = P_{lcam} - Origo_l = (xlcam_1, xlcam_2, xlcam_3)$$

Similarly, one can calculate the vector $\mathbf{v}_r$ starting from the other camera. This is expressed in the coordinate system of the right camera. For working in a single coordinate system, keeping $\mathbf{v}_l$, $\mathbf{v}_r$ should be transformed to the left coordinate system by $\overline{\mathbf{v}}_r = R_{cam}^{-1}\mathbf{v}_r^T$.

The transversal vector is given by the vector product of the two vectors: $\mathbf{v}_n = \mathbf{v}_l \times \overline{\mathbf{v}}_r$. Finally, a vector equation with three scalar parameter, a, b and c is written:

$$a\mathbf{v}_l + b\mathbf{v}_n - c\overline{\mathbf{v}}_r = \mathbf{t}_{cam}\,.$$

Solving this equation, the wanted 3D point is given by the middle point of the section defined by the endpoints $Origo_l + a\mathbf{v}_l$ and $Origo_l + \mathbf{t}_{cam} + c\overline{\mathbf{v}}_r$:

$$P = Origo_l + a\mathbf{v}_l + \frac{b}{2}\,\mathbf{v}_n.$$

In this way the 3D point is written in the coordinate system of the left camera. It can be transformed into the world coordinate system using R_l and $\mathbf{t}_l$.

3.3. Constructing the 3D feature points

Applying the method described above for the registered feature point pairs separately their 3D correspondences can be calculated. Since only a limited number of feature points are used, this will result in a very simple 3D object. A general head model should be fit to the feature points. In the fitting process a self-organizing neural network is used which is presented in the next section.

4. The self-organizing neural network

The *Kohonen net* is a two-layered, unsupervised, continuous valued artificial neural network. Applying the method developed in [8], this network can be used for spatial organization of scattered data. The great advantage of this network, which will be used in this problem, is the ability of fast organization of any number of unordered points. The training procedure will result in a topology-preserving grid following the 3D positions of the reconstructed points.

Here the number of input neurons is three, since the network will be trained by the coordinates of the 3D input points. The output neurons form a quadrilateral grid, and this topology will be preserved during the whole procedure. All the output nodes are connected to each input node and a weight is associated to every connection. Hence a three dimensional weight vector is assigned to each output node.

Now consider these weights as the spatial coordinates of points of the grid. During the training process the weights will be changing, hence this grid will move slowly in the three dimensional space toward the input points, meanwhile the topology of the grid will remain the same. A short description of the training is the following: one of the input points is selected randomly to be the input vector of the net. A winning unit is determined by the minimum Euclidean distance of this point to the output nodes. The node with the minimum distance is the winning unit. Around this node a neighborhood of output points is determined according to the topology of the grid (this neighborhood decreases in time). Finally the weights of the nodes in this neighborhood are updated, i.e., change slightly toward the value of the input vector. After updating the weights in the neighborhood, a new input point is presented and the above steps are repeated. For a more detailed description of the Kohonen network see [10].

4.1. The training procedure

In a standard example mentioned also by KOHONEN the network is trained by points selected from a uniformly distributed area [10]. A main difference between this kind of applications and our problem is that we normally have a finite number of points and they are in the 3D space, while the grid remains two-dimensional. Now we give the exact algorithm of the applied Kohonen network, which produces a rectangular grid onto the 3D reconstructed points.

Let a set of feature points $p_i(x_{1i}, x_{2i}, x_{3i})$, $i = 1, \ldots, n$, be given. The coordinates of these points will form the input vectors of the net. The net itself contains two layers: the input layer consists of three nodes and the output layer consists of m nodes. These m output nodes form a grid with arbitrary, but predefined topology, which is quadrilateral in our case. Each node of the output layer connected to each of the nodes of the input layer. Every connection has a weight: w_{ij} denotes the weight between the input node i and the output node j. In our case $n = 56$ and $m = 576$.

- Coordinates of the feature points: $p_i(x_{1i}, x_{2i}, x_{3i})$, $i = 1, \ldots, n$;
- Coordinates of the output points: $q_j(w_{1j}, w_{2j}, w_{3j})$, $j = 1, \ldots, m$;
- STEP 1. Initialize the weights w_{sj}, $s = 1, 2, 3$, $j = 1, \ldots, m$, as small random values around the average of the coordinates of the input points. Let the training time be $t = 1$;
- STEP 2. Present new input values $(x_{1i_0}, x_{2i_0}, x_{3i_0})$, as the coordinates of a randomly selected input point p_{i_0};
- STEP 3. Compute the Euclidean distance of all output nodes to the input point

$$d_j = \sum_{s=1}^{3} (x_{si_0} - w_{sj})^2;$$

- STEP 4. Find the winning unit q_{j_0} as the node which has the minimum distance to the input point, so where j_0 is the value for which $d_{j_0} = \min(d_j)$;
- STEP 5. Compute the neighborhood $N(t) = (j_0, j_1, \ldots, j_k)$;
- STEP 6. Update the weights (i.e., the coordinates) of the nodes in the neighborhood by the following equation:

$$w_{sj}(t+1) = w_{sj}(t) + \eta(t)(x_{si_0} - w_{sj}(t)) \quad \forall j \in N(t),$$

  where $\eta(t)$ is a gain term decreasing in time;
- STEP 7. Let $t = t + 1$. Repeat STEP 2 – 7 until the network is trained.

If the number of input points is relatively small, as in our case, then the network is said to be trained if all the input points are on the grid. The radius of the neighborhood and the gain term are important factors of computing, both are decreasing in time. Here we used the functions defined in [8] and [7]. The crucial point of the training process is the number of iterations t_0, when the radius diminishes to zero. After this point the fundamental order of points is determined and only the non-approximated input points attract the nearest output.

5. Conclusion

In this paper we have presented our method to generate a 3D face model from two stereo images. It is a model based approach in which only a very few point correspondences are used to calculate the location of the 3D points. These point pairs are usually feature points from the face like corner of the eyes, tip of the nose, etc. Their registration and the reconstruction of the 3D points are completely automatic. Since a relatively small number of points are used and the reconstruction does not result in a complete face model, a predefined general face model is adjusted to the reconstructed 3D points. A self-organizing neural network is applied to iteratively fit the general model to the 3D feature points.

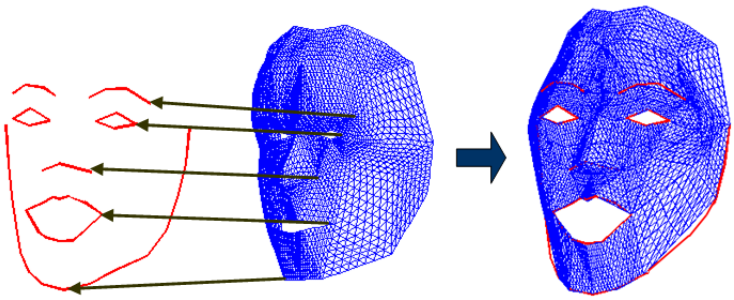

Figure 5: Fitting the general face model to the reconstructed 3D feature points

The resulted face model is carrying the characterics of the faces and reflects the different facial expressions. These benefits urge us to use the reconstructed 3D head for facial emotion recognition in our next research.

Acknowledgement

This work is financially supported by the research project "Theoretical foundations of the technology of man-machine communication" (TÁMOP 4.2.2-08/1/2008-0009). The second author is supported by the János Bolyai Fellowship of the Hungarian Academy of Science.

References

[1] Camera Calibration Toolbox for Matlab: `http://www.vision.caltech.edu/bouguetj/calib\_doc/htmls/example5.html`, referenced in March 2009.

[2] T. Cootes, G. Edwards, C. Taylor: *Active appearance models.* In Proc. of the European Conference on Computer Vision 1998, vol. 2, pp. 484–498, .

[3] T. Cootes, C. Taylor, D. Cooper, J. Graham: *Active Shape Models-Their Training and Application.* Computer Vision and Image Understanding **61**, no. 1, 38–59 (1995).

[4] R. Enciso, J. Li, D. Fidaleo, T.Y. Kim, J.Y. Noh: *Synthesis of 3D faces.* Proc. Workshop on Digital and Computational Video, 2000.

[5] C.E. Esteban, F. Schmitt: *Silhouette and Stereo Fusion for 3D Object Modeling.* Proc. Fourth Internat. Conference on 3-D Digital Imaging and Modeling, 2003, pp. 46–54.

[6] E. Hjelmas, B.K. Low: *Face detection: A survey.* CVIU **83**, 236–274 (2001).

[7] M. Hoffmann: *Modified Kohonen Network for Surface Reconstruction.* Publ. Math. Debrecen **54**, 857–864 (1999).

[8] M. Hoffmann, L. Várady: *Free-form surfaces for scattered data by neural networks.* J. Geometry Graphics **2**, 1–6 (1998).

[9] A. King: *A Survey of Methods for Face Detection.* Computer Vision Course, 2003.

[10] T. Kohonen: *Self-organization and associative memory.* 3rd edition, Springer, 1989.

[11] Yi Ma, St. Soatto, J. Košecká, S.Sh. Sastry: *An Invitation to 3D-Vision.* Springer, New York 2004.

[12] I. Matthews, S. Baker: *Active Appearance Models Revisited.* International Journal of Computer Vision **60**, no. 2, 135–164 (2004).

[13] I.K. Park, H. Zhang, V. Vezhnevets: *Image-Based 3D Face Modeling System*. Journal on Applied Signal Processing **13**, 2072–2090 (2005).

[14] F. Pighin, J. Hecker, D. Lischinski, R. Szeliski, D.H. Salesin: *Synthesizing realistic facial expressions from photographs*. Proc. of SIGGRAPH Conference 1998, pp. 75–84.

[15] H. Stachel: *Descriptive geometry meets computer vision – the geometry of two images*. J. Geometry Graphics **10**, 137–153 (2006).

[16] P. Viola, M. Jones: *Robust real-time object detection*. Technical Report CRL 20001/01, Cambridge Research Laboratory, 2001.

[17] M.H. Yang, D. Kriegman, N. Ahuja: *Detecting faces in images: A survey*. IEEE Trans. on PAMI **24**(1), 34–58 (2002).

[18] Y. Zheng, J. Chang, Z. Zheng, Z. Wang: *3D face reconstruction from stereo: a model based approach*. Proc. of Internat. Conference on Image Processing 2007, pp. 65–68.

Received April 27, 2009; final form December 2, 2009