A shorter version of this paper appears in:
Proceedings of International Workshop on Synthetic - Natural Hybrid Coding
and Three Dimensional Imaging Rhodes, Greece, September 1997

Extended Report on Very Low Bit Rate Facial Texture Coding

J. Ström, F. Davoine, J. Ahlberg, H. Li and R. Forchheimer

Abstract

In this paper we present two methods of enhancing the eigen face approach for the compression of images of human faces. In earlier work, a series of image basis functions (eigen faces) is created using the Karhunen-Loeve transform on a large set of face images (the training set). The image to be encoded can then be approximated by a linear combination of these basis functions. We propose a reversible, non-linear geometrical normalization step to be performed on both the training set and the image before it is being encoded. To further increase quality, a partitioning step of the images into blocks can be performed before the KL transform. An improvement on JPEG of 4 dB has been measured on individual images for low bit rates.

1 Introduction

An efficient way to encode images of human faces at low bit rates would be of great importance to the field of model based coding. If using traditional waveform coding approaches, a small facial texture (eg 200x200 pixels) of reasonable quality would demand about 2 Kbytes, whereas head motion and facial expressions can be represented with around 500 bits per frame [3]. If the same data rate was to be used for the facial textures, it would take about 30 frames to transmit a single facial texture. If only one facial texture is sent per sequence, this will generate problems only in the startup process. Unfortunately though, facial textures can vary over time; different facial expressions and lighting conditions create shadows that a single texture hardly can compensate for. Updating the texture data with traditional waveform coding approaches would be too bit-expensive to be feasible. If, however, a facial texture of high quality could be represented with a number of bits of the same order of magnitude as the motion and expression data, texture updates could become a useful technique to improve the performance of model based coding, as well as improving the time lag problems in the start-up process.

2 Background

Most texture coding methods can be modeled as transform coders. The main drawback of current transform coding methods is that they are not targeted for a specific class of images, but for arbitrary images. The basis functions that are used are thus signal independent. When compressing a very clearly defined class of images such as facial texture images, all-purpose basis functions are obviously not the best choice. In [1] Moghaddam and Pentland use the Karhunen-Loeve transform to create a set of basis functions that does indeed depend on the input class. A number of images are used to build up the training set. All images in the training set are first scaled to the right size, and then a KL transform is performed. This creates a set of basis functions called eigen faces, which are combined linearly to approximate the image to encode.

3 Convexity

If we represent each image with a vector where each vector element is the intensity level of a specific pixel, we can represent each image with a point in a high dimensional *image space*. The class of all face images, or the "cloud" of all face-image points in image space, would then form a data manifold called the *face space*. Figure 1 illustrates what such a face space could look like for 600x600 pixel images. Depending upon the shape of this face space, the KL transform will yield different

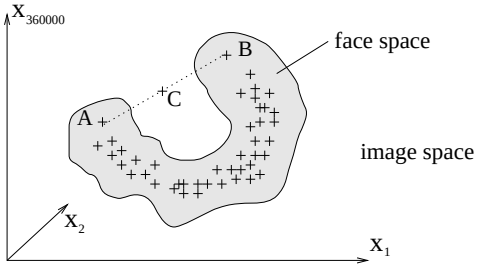

Figure 1: Illustration of what a face space could look like.

performance – the KL transform will work better if the space is convex. Unfortunately enough, our face space is not, which is easy to see; Let point A in figure 1 represent the face of a very narrow eyed person, whereas B would represent the face of a person with a very long distance between his or her eyes. Now, if the face space would be convex, all convex combinations

$$\alpha A + (1 - \alpha)B, \quad \alpha \in [0, 1]$$

would be inside the face space. But $C = 0.5A + 0.5B$ (depicted in figure 2), would result in an image with four eyes, which hardly can be thought of as a face, and must thus be outside the face space. Hence, the face space is not convex. It is clear that purely linear basis functions such as the ones achieved through the KL transform are sub-optimal. Either we will not be able to compress narrow eyed

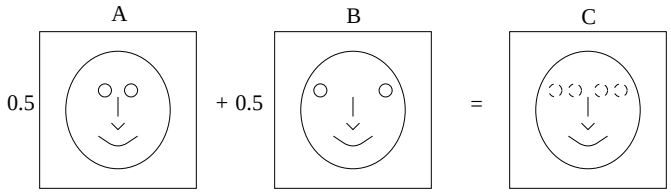

Figure 2: The image C is clearly not a face image.

faces, or we will have an overcomplete base which can represent four-eyed creatures that do not exist. The narrow eyes problem is not the only one: the same kind of problems will occur when trying to compress both open and closed mouths, asymmetric faces, tilted eyes etc.

4 Geometrical Normalization

Since linear transforms are not adequate for non-convex sets, a preprocessing step of geometrical normalization is proposed in this paper. It is a reversible non-linear transformation that will change the face space to a more convex shape, as depicted in figure 3. The transformation process is illustrated in figure 4. First

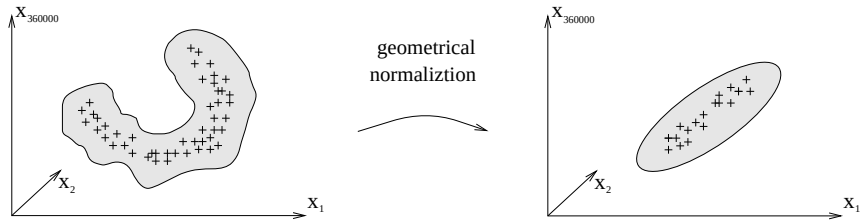

Figure 3: Geometrical normalization: A non-linear image transform that will make the face space more convex.

the original face is segmented out from the background (first image). Then the CANDIDE model [2] is manually aligned to the face, as seen in the second image. The CANDIDE mesh is then warped to a standard geometry (third image), and the result is the normalized face (fourth image). In figure 5, three different faces are shown before (top row) and after (bottom row) geometrical normalization. Note that the faces are more similar to each other after the normalization process than before.

5 The Encoding Process

All N images in the training set are first geometrically normalized. In our experiment we used $N = 100$ images of size 600x600 pixels that were geometrically normalized manually. The mean image of the N images is calculated and subtracted from the training set. Now the KL transform is performed, and N eigen

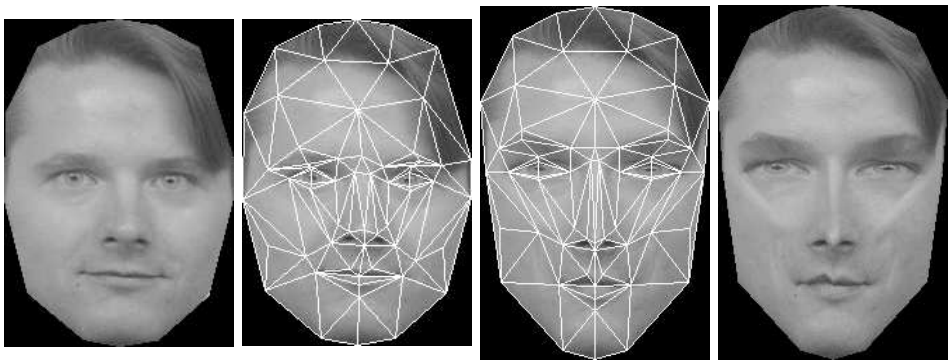

Figure 4: Geometrical normalization. From left: Original image, original with aligned mesh, mesh warped to standard geometry and resulting image.

textures are generated, sorted in decreasing eigenvalue order. This set of eigen textures is called the *eigen set*. A new set called the *data base* is created from the first n eigen textures in the eigen set. The terminology is illustrated in figure 6. The image to encode is first geometrically normalized and then projected onto the eigen textures in the data base. The face image is now represented by the n KL coefficients achieved in the projection and the mesh data needed to reverse the geometrical normalization. To decode the image, the KL coefficients are used to linearly combine the eigen textures in the data base, the mean is added to get the normalized image, and then the mesh information is used to warp the image back to its original shape. The entire process is shown in figure 7. The left most image is the original, which was not part of the training set. After normalization, we get the top middle image. This image is projected onto the space spanned by the data base (of size 100, in this case), resulting in the bottom middle image. After warping the image back, we get the right most image. Without quantizing the coefficients at all, we would need 400 bytes for the coefficients and 195 bytes for the mesh data. In this specific case this means a compression ratio of over 600:1, for a PSNR of 27 dB. The byte count can be lowered by quantizing the coefficients; one byte per coefficient should be sufficient. Further, if this algorithm was to be used in a model based coder, the mesh data (which in effect is the head motion- and facial expression data) will already be sent once to the decoder, so the 100 bytes of KL coefficients would be the only additional information needed for representing the texture.

6 Increasing the Quality

The bit rate of 100 bytes per frame is of the same order of magnitude as the head motion- and facial expression data of 500 bits per frame. The interesting question now is how to improve the quality. A PSNR of 27 dB is certainly not sufficient for all applications. The most straightforward way to increase the quality would be to increase the size of the training set. Figure 8 shows what happens then. The

Figure 5: Top row: original images. Bottom row: geometrically normalized images.

x-axis is the size of the data base, and the y-axis is the measured PSNR between the normalized versions before and after coding. For the dotted curve, the size of the training set and the size of the data base are kept equal ($n = N$). Not surprisingly, the PSNR increases as the $n = N$ goes from 10 to 106. The solid curve is more interesting. Here we have kept the size of the training set (N) to be 106 all the way, while n (the size of the data base) goes from 10 to 106. We can see that the big training set pays off already for small values of n. For instance, if we have 70 images in our training set and use all the eigen textures to encode our image (marked with a circle in figure 8), we will be no better off than if we had used the first 30 eigen textures from the full training set of 106 images (marked with a square). Bear in mind that it is only the data base that is being stored in the encoder and the decoder. Once the data base is built the size of the training set no longer influences the complexity nor the storage demands of the algorithm. For a fixed size of the data base, it would thus make sense to increase the size of the training set until no further significant gain is achieved.

6.1 Enhancing the training set

A drawback with our geometrical normalization step is that it now costs more to increase the size of the training set. Before this procedure can be done automatically (see section 7), it is just not feasible to build training sets bigger than a few

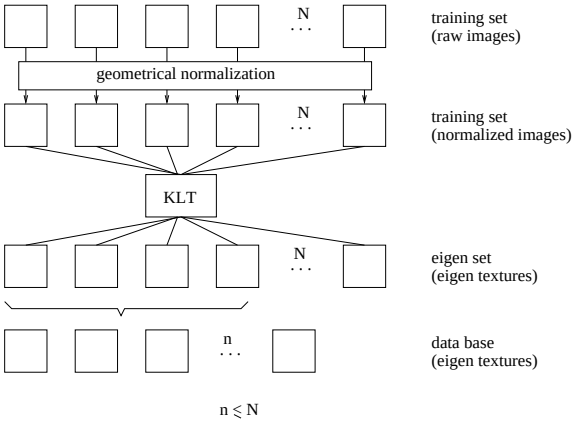

Figure 6: The terminology used.

hundred images. Therefore it is interesting to see if there are ways to enhance the training set without actually adding new faces to the training set. One evident approach would be to mirror the data. If human faces were perfectly symmetric (and the normalization step was done in exactly the same way on both halves of the face), no gain would be expected. On the other hand, if face halves in general were totally different, mirroring would be as good as actually adding new faces to the training set. The reality seems to be somewhere in between, as illustrated in figure 9. Here we start out with a training set of size $N = 50$. Keeping the size of the training set at 50, the solid curve shows how the PSNR goes up when increasing the size of the data base (n). For the dashed curve, the training set is enhanced with 50 new images and kept at the size of 100, with increasing PSNR as the data base grows. As expected, this curve is always better in PSNR than the solid one. The dotted curve is produced with a training set of 50 images, that has been enhanced with mirroring to the size of 100. This curve seems to go right in the middle between the two others. Thus the gain of mirroring does not seem to be insignificant.

Another way to enhance the training set would be to translate the images in the training set by one pixel in each horizontal and vertical direction. A training set would then be five folded without any new faces added. In figure 10 the result of such an experiment is presented. The solid curve is the original training set of 50 images. The dotted curve shows the performance of a training set of 250 images. Two things are to be noted. Firstly, the fact that the 50 first eigen textures generated from the translated training set actually outperforms the non-translated version. Secondly, that the PSNR seems to improve very little when going from a data base size of 50 to a size of 250. Five-folding the number of coefficients does not even give a full dB in PSNR improvement.

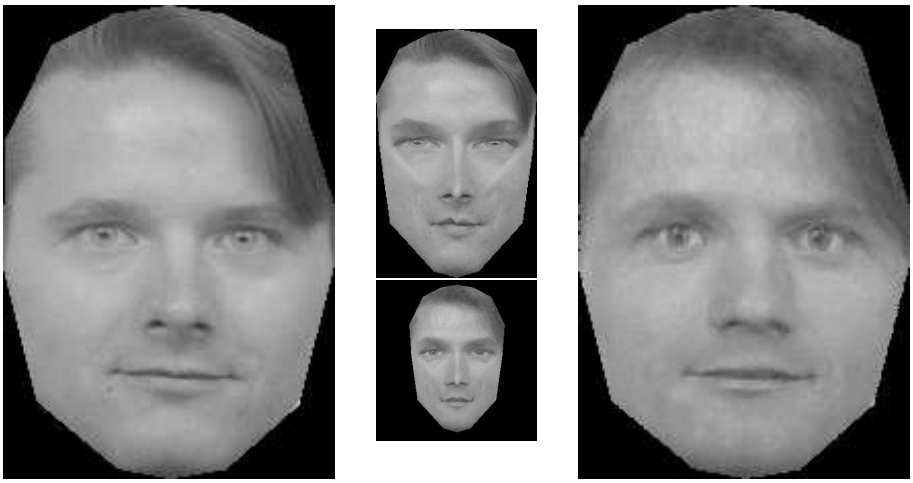

Figure 7: Left: original image, (not part of the training set). Top middle: normalized image. Bottom middle: coded with N=n=100. Right: restored image.

6.2 Blurring the training set

If comparing the first 50 eigen faces in the example above, one would probably discover that the ones generated by the translated training set are slightly more blurred than the others. If this were the reason for the improvement in PSNR, could the same effect be achieved by just blurring the training set? In figure 11 the result of such an experiment is presented. The solid curve is generated from a training set of 50 images without blurring. The dotted, dashed, and dashed-dotted curves corresponds to training sets that are blurred with a Hamming window of size two, four and eight respectively. As seen in the diagram, the PSNR goes up the more blurred the image get. However, at a Hamming window size of eight, the blurring becomes so evident in the decoded image that it is visually worse than the non-blurred version.

6.3 Partitioning the image into blocks

If we were given an automatic method to increase the training set to any desired size, the above methods for enhancing the training set would be superfluous. Even mirroring would be unnecessary, since doubling the number of faces would always be better than mirroring the old ones. Would an increase of the size of the training set be a good way to achieve the needed quality? According to figure 8, it seems like the biggest increases in PSNR are in the beginning – going from 20 to 30 eigen textures improves more than going from 90 to 100. It is hard to predict how many eigen textures a data base would need to achieve a PSNR of for instance 40 dB. Would 500 eigen textures be enough? Maybe even 5000 eigen textures would be too few? This would create storage problems as well as complexity problems. Just storing 5000 eigen textures would demand a storage capacity of 1.7 Gbytes, considering 600x600 pixel images. For each frame, one floating point operation must

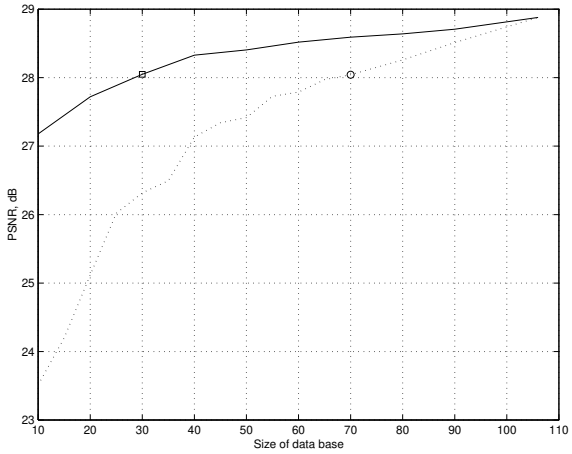

Figure 8: Dotted line: Size of training set = size of data base. Solid line: Size of training set = 106.

be carried out for each pixel, resulting in a complexity of 41 GFLOPS on both the encoder and the decoder side (calculated at 25 frames per second). Today such a complexity is, for reasons of power consumption, not feasible to implement in, for instance, cellular telephones with video capabilities.

Instead of increasing the size of the data base, it is possible reduce the dimensionality of the image space, by partitioning the image into blocks. If the number of coefficients that describe each pixel increases, quality is also expected to rise. We partition the normalized training set into blocks and perform the KL-transform on each type of block. Such an approach based on a modular eigenspace description of the image has already been considered by Pentland in [4]. If we start with a training set of 50 images, we first do a geometrical normalization, we then take the top left block from each image and perform a KL-transform on those 50 blocks, and we get a data base of "eigen-top-left-blocks". The top left block in the (normalized) image to encode is then projected onto this data base. The same thing is done with the next block and so forth. Instead of having 50 coefficients to describe the entire image, we now get 50 coefficients for each block. In figure 12 we present the result of this technique. A block size of 32x32 pixels is used, and a training set of 100 images mirrored to 200 is built without blurring the images. The data base is of size 10 for all blocks, and the coefficients are quantized with a scalar quantizer to 7 bits per coefficient. One float (32 bits) per block is stored to encode the maximum absolute value of the coefficients in that block. Each block then constitutes of $32 + 7 * 10 = 102$ bits. The image is split into 156 blocks, which gives a total of 2184 bytes, including the 195 bytes of mesh information needed to geometrically restore the image. In figure 12, the top left image is the original image. This image was not part of the training set. The top right is coded with our block based algorithm presented above, yielding a PSNR of 35.3 dB. The bottom left image is coded with JPEG to a size of 2508 bytes, with a PSNR of

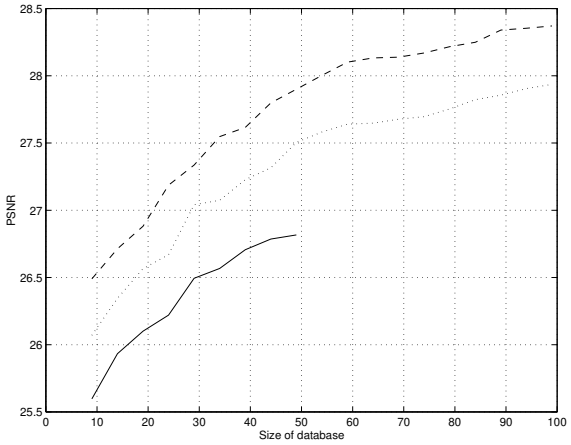

Figure 9: Mirroring the training set. Solid curve: A training set of 50 images. Dotted curve: The same training set, mirrored to the size of 100. Dashed curve: Training set of 100 images.

26.0 dB. Since this is far out of the scope of JPEG optimal operational limits, we subsampled the image, JPEG coded it to 2194 bytes and upsampled it again. This resulted in the bottom right image, with a PSNR of 31.7 dB. The complexity of this approach is the same as the full-face version for the same size of the data base. What we have gained in complexity is that, instead of having a data base of 100 eigen textures, we now have one of 10 eigen textures which still generates a higher quality output. The coding gain however, that is the PSNR per bit, is expected to go down since we limit ourselves to work only inside a block – no information is exchanged between the blocks.

7 Conclusions and Future Work

In this paper we have suggested the use of a geometrical normalization step in the eigen face compression approach. This non-linear transform is meant to have the effect of making the face space more convex, and thus better suited for a linear transform such as the KL transform. In this paper we also investigate different methods for improving the training set without having to add new face images. To mirror and blur the training set seems to have a positive effect on the performance. Moreover, we propose a method to partition the training set into blocks before the KL transform, in order to increase the quality without a dramatic increase in complexity and storage demand. A comparison to JPEG is made, showing an improvement of 4 dB for low bit rates.

Future work is needed to figure out how to best quantize the coefficients in the block based scheme. Right now, the same scalar quantizer is used for all coefficients in the data base, and more than 30% of the bits are spent on transmitting the absolute value of the largest coefficient, which is far too much. The quantiza-

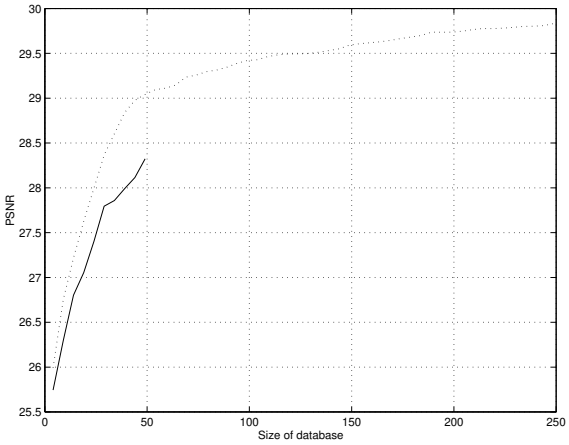

Figure 10: Translating the training set. Solid curve: Training set of 50 images. Dotted curve: Translated version of the same training set.

tion method from [1] could be a starting point for a better quantization. There, Moghaddam and Pentland normalize the coefficients with the corresponding eigenvalues, and a Lloyd-Max quantizer is used with a number of levels that is proportional to the size of the corresponding eigenvalue. Automating the geometrical normalization step is very essential for the feasibility of this entire scheme. Of course, it would also be of great use in order to enhance the training set. Our two step approach with a non-linear warping followed by a linear transform would be interesting to compare to other non-linear methods, such as letting a neural network find a useful representation of the data.

References

[1] Baback Moghaddam and Alex Pentland, "An Automatic System for Model-Based Coding of Faces", M.I.T. Media Laboratory Perceptual Computing Section Technical Report No. 317

[2] M. RydFalk, "CANDIDE, a parameterized face", Technical Report, LiTH-ISY-I-0866, Linköping University, 1987.

[3] H. Li, A. Lundmark and R. Forchheimer, "Image sequence coding at very low bit rates: A review,", *IEEE Trans. Image Proc.*, vol. 3, pp. 589–609, Sept 1994.

[4] A. Pentland, B. Moghaddam and T. Starner, "View-Based and Modular Eigenspaces for Face Recognition", Proceedings 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 21-23 June 1994.

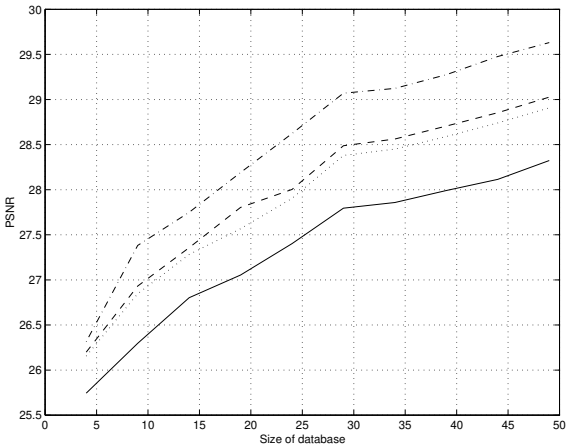

Figure 11: Blurring the training set. Solid curve: Original training set. Dotted, dashed and dot-dashed curve: Training set blurred with a Hamming filter of size 2, 4 and 8 respectively.

Figure 12: Top left: Original image, not included in the training set. Top Right: Our algorithm, 35.3 dB Bottom left: JPEG, 26.0 dB. Bottom right: Subsampled JPEG, 31.7 dB. Images of size 2184, 2508 and 2194 respectively.