Multi-Video Compression in Texture Space using 4D SPIHT

Gernot Ziegler, Hendrik P.A. Lensch, Marcus Magnor and Hans-Peter Seidel
MPI Informatik, Saarbrücken, Germany

***Abstract*— We present a model-based approach to encode multiple synchronized video streams which show a dynamic scene from different viewpoints. By utilizing 3D scene geometry, we compensate for motion and disparity by transforming all video images to object textures prior to compression.**

A 4D SPIHT wavelet compression algorithm exploits inter-frame coherence in both temporal and spatial dimension. Unused texels increase the compression, and a shape mask can be omitted at the cost of higher decoder complexity. The proposed coding scheme is intended for use in conjunction with Free-Viewpoint Video and 3D-TV applications.

I. Introduction

Recent advances in imaging technology have made the acquisition of multiple synchronized videos of real-world dynamic scenes economically feasible. This new media modality is useful, e.g., to re-display the recorded action from an arbitrary perspective (3D-TV, Free-Viewpoint Video) [1], [2]. Although only a sparse set of cameras might be used, the resulting data amounts are still tremendous. In uncompressed state, the data expenditure equals to one video stream per recording camera. Available bandwidth as well as DVD storage capacity currently make distribution of such uncompressed media impossible. While standardized video compression techniques could be applied, these cannot exploit the high degree of redundancy inherent in multiple video recordings of the same object. Especially, no inter-stream coherence would be exploited. In addition, synchronized, random access of video frames is not easily possible.

Using multiple images as texture, also known as multi-view textures, is a relatively new research area. The benefits of having multiple images available to render view-dependent effects have first been shown by Debevec et al. [3]. Compression of multi-view image data in the texture domain has been explored by a few researchers. Nishino et al. apply eigenvector decomposition to a number of textures created from images and a 3D model of the object [4]. Magnor et al. make use of a 4D wavelet decomposition to exploit textural coherence also between textures [5]. All previous work was concerned with encoding multiple textures of a static object. This paper, in contrast, presents an efficient approach to compress multi-video image data of a dynamic scene in the texture domain.

Since the differences in the recorded frames are mainly due to disparity as well as object motion, we separate the recorded information into the object's dynamic shape and its surface texture. Object movement and disparity are compensated in texture space. Differences between texture maps are much smaller than between the original images. A similar approach has been used in [6], applying average textures and shape-adaptive 2D wavelet compression.

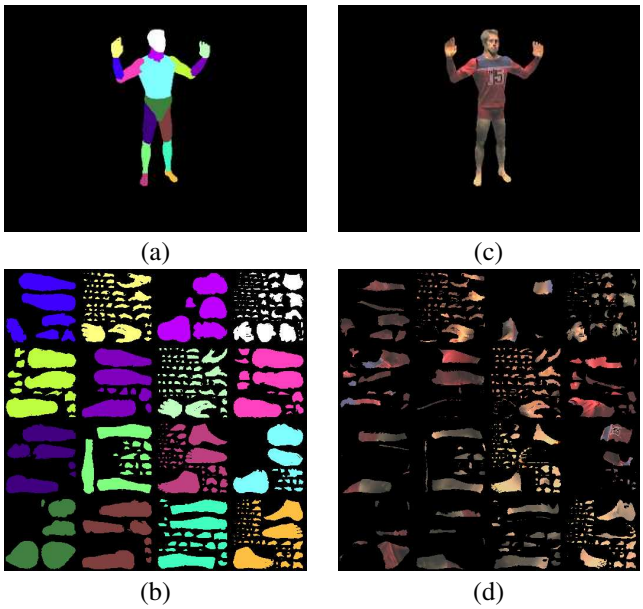

Fig. 1. Resampling into the texture atlas: (a) Color coded body parts. (b) Corresponding regions in texture space. (c) Original frame (320x240). (d) Resampled frame considering visibility (512x512).

A threedimensional, shape-adaptive SPIHT algorithm has been implemented in [7].

We present a complete, model-based 4D SPIHT wavelet encoder for multiple viewpoint video. The 4D SPIHT has been adapted to work with sparsely filled textures and achieves therefore significantly higher compression rates than [6].

II. Overview

Several computing steps are necessary to transform the raw multi-view camera recordings into a surface mesh with motion parameters and corresponding texture maps. We demonstrate the principles on the basis of capturing human actors. The motion parameters are obtained by the free-viewpoint video approach by Caranza et al. [2]. Eight synchronized videostreams of a human actor are captured by calibrated cameras and seperated into fore- and background pixels. Motion parameters are estimated for each frame in each stream by silhouette matching. Using the camera and motion parameters the input frames are resampled into a texture atlas, i.e., a different texture map is created for each time step and camera while the texture atlas parameterization itself is constant. The resampling

is carried out using graphics hardware and is explained in more detail in Section III. To efficiently exploit temporal and spatial redundancy, we propose applying a 4D SPIHT wavelet compression on the reconstructed texture maps. First, we group the texture maps into blocks of spatial and temporal coherency, yielding fourdimensional data blocks of YUV samples. U and V values can optionally be subsampled. Unused texels (currently: black pixels) in these 4D blocks are now filled with averages of the surrounding valid texels, ensuring best possible data compression. The resulting 4D texel data is now compressed with a 4D SPIHT encoder, based on work done in [5]. The encoder is currently able to handle a 4D data block with pairs of equal dimensions (e.g. 8 timesteps of 8 camera view textures at 512 x 512 resolution). Our experimental results with 8 synchronous texture streams show that the usable compression ranges supersede MPEG compression if all codec features are used, which, in contrast to MPEG simulcast transmission, includes the possibility to render arbitrary viewpoints in the decoder.

III. Texture Atlas

The input frames are resampled into a texture atlas using the camera and motion parameters, i.e., a different texture map is created for each time step and camera while the texture atlas parameterization itself is constant. The resampling is carried out using graphics hardware.

A projection of the mesh into the 2D plane can be performed by minimizing texture stretch and distortion on the surface, as for example proposed in [8], [9]. Our texture atlas maps the surface of the 3D model into the 2D domain. It is constructed by projecting the triangles of one patch orthogonally onto a plane defined by the average surface normal.

The selection of patches ensures a minimum sampling density of the surface: Starting with one arbitrary seed triangle, neighboring triangles are added to the patch until the triangle normal deviates too much from the average normal. If one patch cannot grow any further another seed triangle starts a new patch. The body parts' textures are then joined into a single texture as demonstrated in Figure 1a) and c). In order to compute visibility, we perform a traditional shadow mapping approach with the camera position used as the light source. All non-visible texels will be rendered black as can be seen in Figure 1d).

IV. Data grouping and Filling

Compression of a data block commences when all necessary camera images are available as textures.

After resampling, we group the texture maps into blocks of spatial and temporal coherency, yielding four-dimensional data blocks of YUV samples. The block division corresponds to the GOP (group of picture) block structure commonly used in MPEG video formats, and allows for limited random access as long as the whole 4D block containing a certain texture

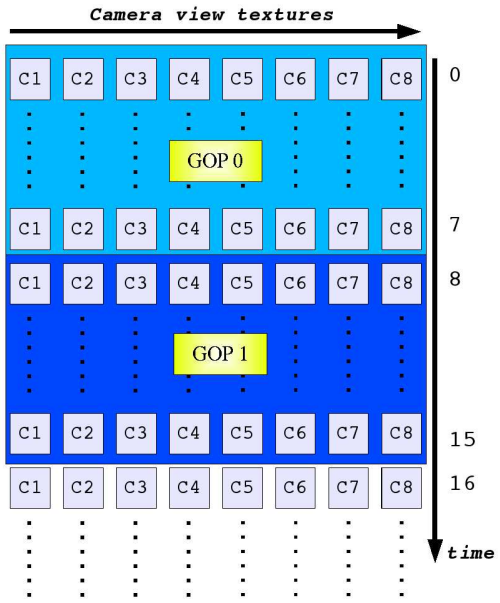

Fig. 2. Group of Pictures arranged from multi-view texture maps.

is decoded, see Figure 2. U and V values can optionally be subsampled, but we currently work with reduced bitrates for these color components, see the SPIHT encoder below.

Unused texels (currently: black pixels) in these 4D blocks are now filled with averages of the surrounding valid texels, see Figure 3 for an example. This ensures best possible data compression under the subsequently applied algorithm, as described in [5].

To serve this purpose, the whole 4D data block is first downsampled in a Laplacian 4D pyramid, all the way to the lowest resolution of 1x1, taking the different dimension extents into consideration (a division by two remains one if the result would be smaller than one). Afterwards, the pyramid is traversed backwards from the lowest to the highest resolution, and each unused (black) texel receives the color of its associated, average parent in the previous level. This way, it is ensured that all unused texels are filled with a color value that corresponds to the average of all valid texels in its support region.

V. Wavelet Encoding

The following 4D wavelet transformation uses Haar wavelets. We take the 4D data block that was filled in the previous step, and sequentially apply a 1D Haar wavelet tranform in all four dimensions until even the image dimension sizes have been reduced to 2. Finally, compression commences. The compression algorithm is based on the widely-used SPIHT algorithm, although in a new adaptation, making it suitable for 4D data. It is based on work done in [5]. The encoder is currently able to handle a 4D data block with pairs of equal dimensions (e.g. max(s,t,u,v) = {8,8,512,512}., that is, 8 timesteps of 8 camera view textures at 512 x 512 resolution).

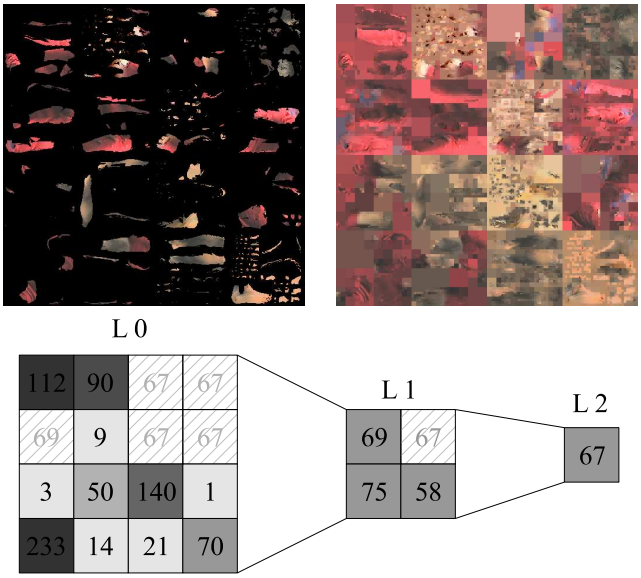

Fig. 3. Top left: input texture map. Top right: Same map after filling operation. Bottom: fill, downsampling from L0 to L2, upsampling L2 to L0. Hatched boxes mark unused texels that were filled during upsampling.

The SPIHT encoder very much resembles its classic 2D

```
for all timesteps of camera images T(t,c):
  for each color component Y, U and V:
    group s cameras with t timesteps into
      4D array (textures, u x v texels)
    downsample 4D array log(max(s,t,u,v)) times.
    upsample selectively, filling unused texels
    apply 1D Haar wavelet transformation
      repeatedly on (cam, time) dimensions s, t
    apply 1D Haar wavelet transformation
      repeatedly on (image) dimensions u, v
    apply 4D SPIHT onto 4D array
      result: 4ST code stream
```

Fig. 4. Encoding (pseudo-code).

counterpart (see [10]), with one exception: Since the 4D dimensions are of different size, the codec has to be able to detect boundary conditions that generate a different number of descendants in the wavelet data traversal.

Pseudocode for the overall encoding pipeline is listed in Figure 4.

VI. The Shape Mask

As mentioned earlier, unused texels are filled with auxiliary data that improves compression. The information about which texels are unused is thus lost in the encoding process. A classical approach to solve this problem is to provide the decoder with a bitmask that masks out originally unused texels after the SPIHT decoding process and the wavelet reconstruction. This bitmask is not needed if the original 3D model is available: If only the original camera views are reconstructed, then only valid and used texels will be accessed in the rendering phase, since only those were exposed to the camera in the given view. Therefore it suffices to provide the renderer with the unmasked texture maps.

If, on the other side, *intermediate views* shall be rendered using multi-view interpolation techniques, then the unused texels must be known. This is necessary as each image pixel is composed of texture lookups from the nearest camera views, under the precondition that the texel is *valid* in the respective camera texture map. If the whole texture map is still filled with auxiliary data, it would affect the multi-view interpolation algorithm with unwanted texels.

Therefore this bitmask, also called the *texel usage mask*, needs to be applied to the texture maps before rendering.

This can be done in two ways: One way is to store it in the bitstream, which requires some extra storage space. The other approach is to reconstruct the relevant bitmasks from the 3D model in the decoder, using a shadow casting algorithm. This improves compression, but requires a minimum of two extra rendering passes for each time step to reconstruct the adjacent texture maps' bit masks, and thus impacts negatively on rendering performance.

VII. Decoding

Given the aforementioned encoding pipeline, decoding the streams is straight forward and explained in Figure 5.

```
for all required camera images and timesteps:
  for each color component Y, U and V:
    choose 4ST code stream containing T(t,c)
    restore 4D array with 4D SPIHT
    apply 1D Haar wavelet transformation
      repeatedly on (image) dimensions u, v
    apply 1D Haar wavelet transformation
      repeatedly on (cam, time) dimensions s, t
    extract T(t,c) from 4D array
  (apply shape mask to T(t,c) )
```

Fig. 5. Decoding (pseudo-code).

Since time steps are usually read sequentially, several time steps can be extracted at once from the 4D code stream. The bit mask can only be applied if it has been transmitted to the decoder. If this is not the case, shadow casting must be applied for masking if multi-view interpolation is intended (as noted in the previous section).

Figure 6 shows example output from the reference decoder. Notice the typical wash-out effect of wavelet compression. The outer contours were transmitted in an additional shape-mask.

VIII. Results

For testing the codec, we encoded and decoded the texture maps found in [11]. This footage was created during experiments done by Theobalt et al, which used free viewpoint-video recording and supplied an animated 3D model of a human actor and according camera views.

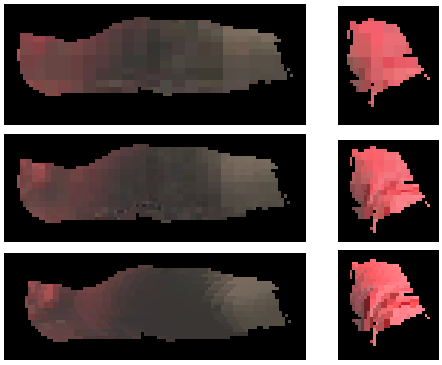

Fig. 6. Two texture map details. Decoding was performed with equal Y, U, V datarate, and using the fill feature. Top: 0.05 bpp; Middle: 0.25 bpp; Bottom: Encoder input.

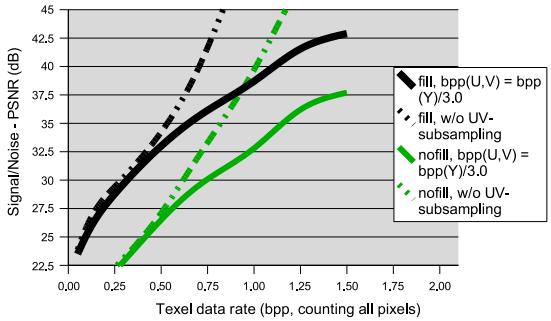

Fig. 7. *Codec results for dancer scene's textures.*

Our experimental results with 8 synchronous texture streams of 40 frames each show that the compression ratio ranges up to 288:1, if a texel data rate of 0.05 bpp, the fill feature and lower UV bitrate are used. In this compression mode, the PSNR reached 23.5 dB. A PSNR of 30 dB is reached at a texel data rate of 0.30 bpp.

Without UV bitrate reduction and the sparse fill feature, the usable compression ratio ranges up to 50:1. This mode can be used if no bitmask shall be transmitted and authentic color resolution is of high importance. An overall compression rate of 20:1 (Y: 0.75 bpp, U & V: 0.25 bpp) in this mode can safely be used without sacrifices in reproduction quality (PSNR: 30 dB).

The compression ratios of up to 288:1 compete easily with MPEG standard compression of the texture maps as video streams. Our codec's advantage is synchronous access to the texture maps, which is hard to achieve with parallel MPEG2 decoders. But it is important to keep in mind that either bit masks or an animated, textured 3D model and camera parameters must be transmitted to reverse the sparse filling.

IX. Conclusions

We have presented a coder/decoder for a recently introduced data type, synchronous multi-video image data, needed for Free-Viewpoint Video as well as 3D-TV applications. Our method relies on having an (approximate) 3D geometry model of the dynamic scene available that can be used to map the video images to textures. By working in texture space, we compensate for disparity between camera views as well as for object motion over time. The codec competes well with MPEG standard compression of the video streams (simulcast), with the additional advantage of reproducing arbitrary viewpoints.

To further improve coding efficiency, we are going to investigate how to retrieve surface reflectance characteristics in addition to the diffuse reflection component that is currently exploited, and find estimates on the optimal area ratio between texture map and rendered image.

Other texture parameterization methods could create continous texture atlases over large areas of the mesh surface (e.g. whole torso, arms, legs, head laid out in only one texture each, altogether 6 continous body textures). Such larger atlases provide better efficiency, as less discontinuities (and thus less high-frequency components) have to be encoded in the SPIHT compression.

It would also be worth investigating whether graphics hardware (GPUs) could aid in wavelet transformation and possibly even the SPIHT coefficient separation.

If this approach is going to be pursued, replacing the four sequential 1D Haar wavelet transforms with two 2D Haar transforms should be considered. This way, the graphics hardware's texture scaling capabilities would be used more efficiently.

References

[1] C. Fehn, P. Kauff, M. O. de Beeck, F. Ernst, W. Ijsselsteijn, M. Pollefeys, E. O. L. Van Gool, and S. I., "An evolutionary and optimised approach on 3D-TV," *Proc. Int. Broadc. Conf. (IBC'02)*, pp. 357–365, Sept. 2002.

[2] J. Carranza, C. Theobalt, M. Magnor, and H.-P. Seidel, "Free-viewpoint video of human actors," *ACM Trans. on Graphics*, vol. 22, no. 3, July 2003.

[3] P. Debevec, C. Taylor, and J. Malik, "Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach," *SIGGRAPH 96*, pp. 11–20, Aug. 1996.

[4] K. Nishino, Y. Sato, and K. Ikeuchi, "Eigen-texture method: appearance compression based on 3d model." *Proceedings of IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 618–624, June 1999.

[5] M. Magnor, P. Ramanathan, and B. Girod, "Multi-view coding for image-based rendering using 3-D scene geometry," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, no. 11, pp. 1092–1106, Nov. 2003.

[6] G. Ziegler, H. Lensch, A. Naveed, M. Magnor, and H. P. Seidel, "Multi-video compression in texture space," in *IEEE ICIP*, 2004, submitted.

[7] H. Danyali and A. Mertins, "Fully scalable texture coding of arbitrarily shaped video objects," *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'03)*, pp. 393–396, Apr. 2003.

[8] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe, "Texture mapping progressive meshes," in *SIGGRAPH 2001*, Aug. 2001, pp. 409–416.

[9] X. Gu, S. J. Gortler, and H. Hoppe, "Geometry images," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 355–361, July 2002.

[10] A. Said and W. A. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, pp. 243–250, 1996. [Online]. Available: citeseer.ist.psu.edu/said96new.html

[11] Theobalt, Ahmed, and Ziegler, "MPI Dancer - a test sequence for multi-view reconstruction and codec research," 2004. [Online]. Available: www.mpi-sb.mpg.de/ gziegler/mpidancer/