

Multi-video Coding with Dense Correspondence Fields

Benjamin Meyer, Christian Lipski, Björn Scholz and Marcus Magnor

Computer Graphics Lab

TU Braunschweig

Email: {Meyer;Lipski;Scholz;Magnor}@cg.tu-bs.de

Website: <http://www.cg.cs.tu-bs.de>

Abstract—We present a fast decoder for free-viewpoint video navigation. Our free-viewpoint video data consist of simultaneously recorded video streams captured with multiple cameras plus additional dense correspondence information. Neither the size nor the representation of the data is appropriate for real-time rendering. We propose a data compression approach for multi-view video sequences that allows for a real-time decoding and playback of long sequences from hard drive.

I. INTRODUCTION

Free viewpoint video players use image interpolation algorithms to generate novel views of a recorded scene. The user may freely navigate through the video, altering both viewing direction and time. But this functionality comes at a cost, additional warping data is needed, increasing the input data vastly. Using the raw input data, free viewpoint players are not able to cope with this amount of data in realtime, requiring to load the complete video data into cache instead. Thus, the video length is limited by the cache size. Alternatively, some restrictions can be made concerning either the scene geometry (e.g. [1]) or the camera setup (e.g. [2]), reducing the complexity of possible camera warps and hence the needed interpolation data.

Inspired by a new image interpolation algorithm based on correspondence fields ([3], [4]), we present a data compression method reducing these correspondence fields to a size where the implementation of a streamable video player becomes feasible. Hence, our player is able to playback multi-video sequences of arbitrary length and without any restrictions to the scene or camera setup, still offering full user interactivity.

For data reduction, we convert the floating point values of correspondence fields to byte size and employ single image compression.

The paper is organized as follows. After giving an overview of related work in Section II, we summarize the used rendering approach in Section III. We present and discuss our approach for multi-view video data compression in Section IV. Implementation details ensuring real-time playback are given in Section V, followed by a presentation of our results in Section VI and a short discussion in Section VII.

II. RELATED WORK

Recently, wavelet-based multi-view compression schemes [5] and approaches using layered depth images (e.g. [6], [7],

[8], [9], [10]) have been proposed. Although achieving high quality, their performance does not fulfill our needs. The same applies to recent floating point compression algorithms. As the used dense correspondence fields [3] have to be calculated beforehand in an offline step, we need an appropriate floating point compression scheme. Streamable floating-point data compression algorithms (e.g. [11], [12]) usually support good compression rates paired with acceptable computational effort, but are still not suitable for our real-time purpose. Neither the size, nor the computational effort is small enough for real-time playback in our multi-view video environment. Magnor et al. proposed two efficient codec schemes. But unlike our approach, [13] relies on a priori knowledge of the scene, whereas [14] is based on static scenes and cannot be applied to dynamic content. The same limitation is made in [15] by Hornung et al..

Many recent approaches in free viewpoint playback originate from Image-based rendering (IBR). IBR approaches (eg. [2], [16], [17], [18]) are capable of producing high-realistic results but usually depend on camera-calibration, time-synchronization or other additional information like scene depth, geometry or epipolar constraints.

Recently, Stich proposed a novel approach [3] outperforming optical-flow based interpolation algorithms in quality and IBR-algorithms in configuration effort.

III. RENDERING

Our video player utilizes the image interpolation algorithm proposed by Stich et al. [3]. His basic assumption is that robustly matching visible edges is the key to visually plausible image interpolation. For each input image pair he obtains a correspondence field encoding per pixel object motion (a 2D float vector per pixel).

Based on this approach, we generate new views of the scene where the user may freely navigate through the scene, i.e. he may influence horizontal and vertical viewing direction as well as playback time. As this requires new images interpolation in three dimensions, we do not apply the original rendering approach between two images. Instead, we generate new images in this volume where each input image is represented by a three-dimensional vertex (two viewing directions+time).

A Delaunay tessellation of this point cloud into tetrahedra is computed, it ensures that each novel view is located in exactly

one tetrahedron. It can be rendered by forward warping and blending the four original images (Fig. 1). For rendering, we thus need four color images in addition to the twelve floating point correspondence maps between them. Even though a correspondence field with the quarter size of the actual image still provides nearly flawless interpolation results, for a 960x540 video, this would result in a memory usage of $960 * 540 * 4 * 3$ byte for the image data and an additional $480 * 270 * 12 * 8$ byte for the correspondence data, resulting in approximately 17.8 MB per rendered frame.

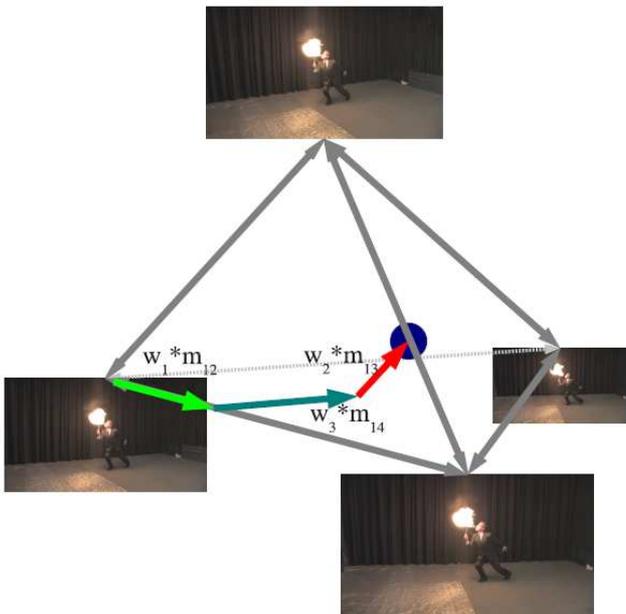


Fig. 1. Warping within a 2D or 3D environment. The green, turquoise and red arrows describe the weights, the gray the correspondence fields m_{ij} and m_{ji} . The blue circle is the target point.

IV. DATA COMPRESSION

The presented player will enable the user to arbitrarily navigate through time and space within the video. For preserving the real-time capability, a fast access to the video data as well as an easy handling is essential.

A. Correspondence Field Compression

Simplified, a correspondence field contains in each entry a pixel displacement vector, normally encoded in two 32-bit floats (displacement in x and in y-axis). For a video with a resolution of 960x540 this results in a file size of approximately 4 MB. As for navigation along the negative time axis the reverse correspondence fields are needed as well, the huge amount of 8 MB warp data has to be handled per camera pair per frame, in addition to the pure video data.

Even though in most cases the correspondence field can be reduced to the quarter size of the actual image still providing nearly flawless interpolation results, the amount of raw data

is still too much for desirable file-sizes and smooth playback, specifically from local storage like hard disks.

To ensure a high compression rate as well as an appropriate decompression speed and accuracy, we make use of the PNG single image compression. For each image pair there exist two opposite correspondence fields. We encode these two 2D correspondence fields into a single PNG-file, making use of the four color channels. Thus, we employ a discretization scheme from the original float values to bytes. Instead of simply rounding the original float values to integer (as this would restrict us to a maximal displacement between -127 and 127), we first determine the maximal (max_x and max_y) and minimal (min_x and min_y) displacement values of the entire correspondence field. The respective stepsize is given as

$$step_{\{x,y\}} = \frac{max_{\{x,y\}} - min_{\{x,y\}}}{2^8 - 1} \quad (1)$$

and can be used to express the single displacement warps:

$$R = \frac{fX - min_x}{step_x} ; G = \frac{fY - min_y}{step_y} \quad (2)$$

with fX and fY the displacement values and R and G the first two entries in the PNG-pixel. Saving min_x and min_y as well as the stepsizes in x and y in the PNG-header enables a simple reconstruction:

$$\begin{aligned} fX &= min_x + (R * step_x) \\ fY &= min_y + (G * step_y) \end{aligned} \quad (3)$$

The hereby made error is proportional to the step size and thus stays relatively small compared to the scene motion.

B. Video Compression

Besides warp compression, the high amount of video data is the second problem to be considered. Existing video compression algorithms all share the aspect of exploiting similarities between multiple frames. More exactly, most compressed frames depend on their predecessor (referred to as I-, P- and B-frames). For a backward playback of the video however, this results in an intensive CPU- and HDD-load since multiple images have to be decoded per frame. To avoid this, every image has to be compressed on its own.

We decided to choose the JPEG compression for this task, as it yields a manageable file size with low decompression effort. The compression is not lossless, but however, artifacts could rarely be seen in the resulting video.

V. IMPLEMENTATION

A. An Adaptable Container

Even though the final results regarding compression reduce the data to a manageable amount, the data still resides in single files across the storage preventing a fast access. All of the resulting data has to be combined and besides video frames and warps, the camera configuration, tetrahedra and playback information needs to be integrated, too. Thus, we designed a special container (Fig. 2). The configuration and tetrahedra are stored at the beginning of the container file.

Next, all the compressed video frames and warps are stored. Finally, a frame and warp index-table containing their in-file position, their size and an identifier is created. The index-table provides the opportunity to jump arbitrarily at any position within the container to obtain the desired data. We make use of single image compression techniques and encode the data in the correct temporal order. Therefore, single frames can be read and displayed consecutively making streaming possible.

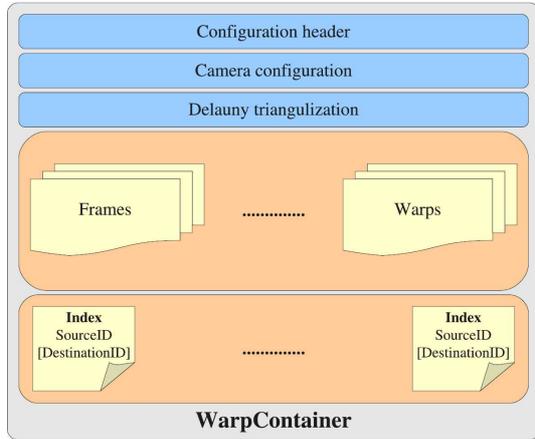


Fig. 2. Container for data storage, containing the configuration and tetrahedra, the compressed video frames and warps as well as a frame and warp index-table.

B. Caching and Multithreading

Multithreading can significantly increase performance according to the scenario, extend and number of CPU cores used. But to avoid deadlocks, the amount of reading threads is set to two: one for the frames and one for the warp-fields. A third thread is used for rendering, as this is done on the GPU.

The fetched data is saved in a cache waiting for the next rendering step. As in a multithreading environment, shared data needs to be locked for writing access, using a single cache would result in total blockage during the acquiring operation. To countermeasure this effect, the reading threads grabbing the frames from the container-file have their own caches. This results in the major advantage, that when less than four of the demanded images or correspondence fields need to be read from the container and the others still lie in the cache, the player can already access the others and transfer them to the GPU without the need to wait for the reading/decompression of the other thread. Data transferred from the second to the first cache is instantly removed from the second cache.

In combination with the in Section V-A achieved data locality and a pre-ordering of the frame/warp requests according to their position within the container, it makes the SpaceTime-Player even capable of replaying the test-scenes from an USB Flash drive.

C. Prefetching

By splitting acquisition and rendering into distinct threads, another opportunity to further improve performance is created:

prefetching. Besides just a few copies in memory, the CPU is freely available during the rendering itself. By trying to estimate whether or not different frames or warp-fields will be demanded in the next loop, this can be used to at least partially preload the new frames. Therefore, the motion-vector of the mouse and the time de-/increment is re-applied to the tetrahedral query and the possibly demanded frames were added to the frame- or warp-queue. Thus, the acquisition thread can preload and decompress the data during rendering.

D. GUI

The graphical user interface needs to ensure an intuitive handling of the player. So considering the appearance of most other video players, the interface is split into two parts: video and controls (Fig. 3). The only difference is the opportunity to change the point of view. This is done by simply clicking and dragging the mouse on the playback area of the player.



Fig. 3. The Graphical User Interface of our free viewpoint video player. In addition to traditional interface elements the user can control view direction (by dragging the mouse on the video) and playback rate (slider at bottom right).

VI. RESULTS

As mentioned before in Section IV-A, we achieve a high data compression rate, especially on the correspondence fields. For small correspondence fields (as 240×135), the raw 32-bit float data would have a size of $240 * 135 * 4 * 2 * 2 \text{ Byte} \approx 506 \text{ KB}$, reduced to a PNG of only 13 KB (one factor of 2 results from a warp consisting of two float values, in x and y-axis, and the second factor of 2 is due to the backward correspondence field saved in the same PNG). This yields a size reduction factor of approximately 39 (only for the correspondence fields). In combination with the JPEG video data a compression factor of approximately 30 is still achievable. Further, the backtransformation from the saved stepsizes to the original float data is done in a few multiplications and takes no time, whereas the PNG decompression can be done on the CPU during rendering.

The main disadvantage of such a high compression rate of the correspondence fields is the resulting quality loss. The decompression error results from the discretization of the single warps relative to the stepsize. Hence, for scenes with higher motion also the error increases, but in all regarded scenes, the error was not noticeable.

Due to a low decompression effort and a compact data storage, as well as the in Section V presented acceleration approaches, a high rendering speed can be achieved. Depending on the video resolution and the correspondence field size, frame rates beyond the 25-fps are reached. As the original video is normally recorded at this frame rate, a higher display rate cannot enhance the output quality any more. Besides, the used hardware, especially the graphics card, influences the rendering speed as well. Thus, we tested three different graphics cards in a normal pc, varying from a ATI 3850 HD, over a Geforce 8800GTS to a Geforce 260GTX (entitled as low, med and high). The results are shown in Table I.

	Resolution	Grid resolution	FPS (playback)
Low	480x270	480x270	≥ 60 (27)
Med	480x270	480x270	≥ 60 (35)
High	480x270	480x270	≥ 60 (50)
Low	960x540	960x540	7 (7)
Med	960x540	960x540	15 (15)
High	960x540	960x540	17 (17)
Low	960x540	480x270	12 (12)
Med	960x540	480x270	20 (17)
High	960x540	480x270	35 (25)

TABLE I
RENDERING PERFORMANCE COMPARISON IN RESPECT TO THE TESTED COMPUTER SYSTEMS. GRID-RESOLUTION DENOTES THE ACTUAL RESOLUTION OF THE VERTEX MESH USED FOR COMPUTING THE FORWARD WARPS. THE FPS-VALUES ARE SPLIT IN PURE-RENDERING AND DURING PLAYBACK (IN BRACKETS). THE ≥ 60 FPS RESULTS FROM VERTICAL SYNC BEING ACTIVE LIMITING THE RENDERING SPEED TO THE REFRESH-RATE OF THE MONITOR.

For quality and performance assessment, we refer to the downloadable version of the player:

<http://graphics.tu-bs.de/publications/mvcWin32.zip>

VII. CONCLUSION

In this paper, a whole framework for free viewpoint videos has been introduced and a simple and user-friendly UI has been designed. Furthermore, video and warp compression has been evaluated and implemented and all necessary data regarding the playback has been concatenated in a special container format. Finally, a novel image interpolation algorithm has been integrated to provide the actual FVV capability.

The results clearly state the capabilities of this approach. Depending on the resolution of the video, the player is capable of rendering free-viewpoint video in real-time. Even the playback of videos from USB flash drives is possible with more than 25FPS at resolutions of quarter-HD. Furthermore, the final compression-rate of the source data is high enough to make the FVV player available as a downloadable application including sample videos for live demonstration purposes.

But even though the player is thoroughly usable as a real-time FVV system, several tasks reside in the future. First of all, a dynamic density adaption of the vertex mesh could significantly reduce the load of the GPU and thus

improve rendering speed up to full-HD capabilities. From the implementation point of view, other multithreading schemes or even multiprocessing should be evaluated because it could significantly improve the all-over performance. Specifically the new features of DirectX 11 regarding multithreaded rendering should be evaluated, since those probably open completely new opportunities compared to DirectX 9/10.

But besides demonstration purposes of one single interpolation approach, the space-time player could also be used to compare other state-of-the-art and future interpolation algorithms with each other since those are easily integrated in this framework.

REFERENCES

- [1] J. Carranza, C. Theobalt, M. Magnor, H.P. Seidel. Free-Viewpoint Video of Human Actors. In *Proceedings of ACM SIGGRAPH*, 569–577, 2003
- [2] C.L. Zitnick, S.B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. *ACM Transactions on Graphics*, 23(3):600–608, 2004.
- [3] T. Stich, C. Linz, G. Albuquerque, and M. Magnor. View and Time Interpolation in Image Space. In *Proceedings of Pacific Graphics*, 27(7), 2008.
- [4] T. Stich, C. Linz, C. Wallraven, D. Cunningham, M. Magnor. Perception-motivated Interpolation of Image Sequences. In Symposium on Applied Perception in Graphics and Visualization (APGV), 97–106, 2008
- [5] J. Garbas, B. Pesquet-Popescu, M. Trocan, and A. Kaup. Wavelet-based multi-view video coding with joint best basis wavelet packets. In *15th IEEE International Conference on Image Processing*, 1232–1235, 2008.
- [6] X. Cheng, L. Sun, and Yang S. A multi-view video coding approach using layered depth image. In *9th Workshop on Multimedia Signal Processing*, 143–146, 2007.
- [7] J. Shade, S. Gortler, L. He, and R. Szeliski. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 231–242, 1998.
- [8] S.U. Yoon, S.Y. Kim, and Y.S. Ho. Preprocessing of depth and color information for layered depth image coding. *Lecture Notes in Computer Science (LNCS)*, 3333:622–699, 2004.
- [9] A. Smolic, K. Müller, P. Merkle, C. Fehn, P. Kauff, P. Eisert, T. Wiegand. 3D Video and Free Viewpoint Video Technologies, Applications and MPEG Standards. In *Proceedings of International Conference on Multimedia and Expo*, 2006.
- [10] P. Merkle, K. Müller, A. Smolic, T. Wiegand. Efficient Compression of Multi-View Video Exploiting Inter-View Dependencies Based on H.264/MPEG4-AVC. In *Proceedings of International Conference on Multimedia and Expo*, 2006.
- [11] P. Ratanaworabhan, J. Ke, and M. Burtcher. Fast lossless compression of scientific floating-point data. In *Proceedings of the Data Compression Conference*, 133–142, Washington, DC, USA, 2006.
- [12] P. Lindstrom and M. Isenburg. Fast and Efficient Compression of Floating-Point Data. *Transactions on Visualization and Computer Graphics*, 1245–1250, 2006.
- [13] M. Magnor and P. Ramanathan and B. Girod. Multi-View Coding for Image-based Rendering using 3D Scene-Geometry. In *IEEE Trans. Circuits and Systems for Video Technology*, 1092–1106, 2003
- [14] M. Magnor and B. Girod. 3D TV: Data compression for light-field rendering. In *International Conference on Computer IEEE Transactions on Circuits and Systems Video Technology*, 338–343. 2000.
- [15] A. Hornung and L. Kobbelt. Interactive Pixel-Accurate Free Viewpoint Rendering from Images with Silhouette Aware Sampling In *Computer Graphics Forum*, 2090–210, 2009
- [16] W. Matusik and H. Pfister. 3D TV: a scalable system for realtime acquisition, transmission, and autostereoscopic display of dynamic scenes. In *International Conference on Computer Graphics and Interactive Techniques*, 814–824. ACM New York, NY, USA, 2004.
- [17] N. Snavely, S.M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. In *International Conference on Computer Graphics and Interactive Techniques*, 835–846. ACM New York, NY, USA, 2006.
- [18] S. Vedula. Image Based Spatio-Temporal Modeling and View Interpolation of Dynamic Events. *PhD thesis, University of Washington*, 2001.