Real-time Free-Viewpoint Navigation from Compressed Multi-Video Recordings

Benjamin Meyer, Christian Lipski, Björn Scholz, Marcus Magnor
Computer Graphics Lab
TU Braunschweig, Germany
{Meyer, Lipski, Scholz, Magnor}@cg.tu-bs.de

Abstract

We present an interactive player for free-viewpoint video. Free-viewpoint video data typically consist of large amounts of video datasets captured from multiple cameras and additional inter-view and frame information. Neither the size nor the representation of this data is appropriate for real-time rendering. Inspired by a new image interpolation approach based on correspondence fields, we propose a data compression algorithm for streamable multi-view video sequences allowing a real-time decoding and playback for videos of arbitrary length without loading them completely into cache.

1. Introduction

In the last few years, several free viewpoint video players have been introduced. Based on image interpolation algorithms for different camera views or points in time, they allow the user to navigate freely through the video, generating new camera views on the fly. But these image interpolation algorithms rely on additional warping data increasing the input data vastly. Existing free viewpoint players countermeasure this problem by preloading the complete video data into the cache, allowing a fast access but limiting the player to short videos. Alternatively, some restrictions are 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. In order to guarantee real-time capability, we design a container format which allows caching, multithreading, and prefetching.

The paper is organized as follows. After giving an overview of related work in Section 2, we summarize the used rendering approach in Section 3. We present and discuss our approach for multi-view video data compression in Section 4. Implementation details ensuring real-time playback are given in Section 5, followed by a presentation of our results in Section 6 and a short discussion in Section 7.

2. Related Work

Evolving real-time playback of free-viewpoint multi-view video involves many tasks that need to be taken care of. Multiple video-streams and warp-fields must be compressed in order to keep the amount of data as small as possible. Image interpolation methods must be chosen wisely and implemented efficiently to ensure smooth playback and navigation. All these areas of research received a lot of attention in computer graphics.

2.1. Multi-view video compression

Multi-view video compression can have a significant impact on quality, size and computational effort. Regarding quality, wavelet-based multi-view video compression (MVC) schemes (like [5]) provide high quality video compression. But especially wavelet-compression usually has high computational demands. Other approaches like MVC using layered depth images (LDI) (e.g. [6], [7], [8], [9], [10]) produce plenty of overhead. As backward decompression causes severe performance fall-offs, these MVC schemes are not suitable for the realtime player. An efficient coding scheme has been proposed by Magnor et al. [12]. However, it relies on a priori knowledge of the scene geometry.

2.2. Floating-point data compression

The creation of deformation vector fields as described in [3] is quite time-consuming, disqualifying it for real-time

purposes. In order to achieve this capability, these warp-fields need to be precomputed and stored to disk. Usually consisting of 32-bit floating point data, the amount of data is huge and needs to be significantly reduced for real-time playback purposes. Streamable floating-point data compression algorithms like [13] or [14] usually support very good compression rates paired with acceptable computational effort, but are still not suitable for our purpose. Neither the size, nor the computational effort is small enough for real-time playback in our multi-view video environment.

2.3. Interpolation

Image interpolation is the key task in free-viewpoint multi-view video playback. Many recent approaches originate from Image-based rendering (IBR). IBR approaches (eg. [2], [15], [16], [17]) 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. Further improvements can be made when the data is fitted to a scene or actor model [1]. In contrast, we can cope with scenes, where the geometry is unknown, changes over time or is even impossible to reconstruct. In [2], a real-time player quite similar to this one is presented by Zitnick et al., but being based on a specific camera setup. All cameras are lying on a horizontal plane next to each other and point roughly into the same direction. Hence, the complexity of possible camera warps shrinks enormously. In addition, Zitnick et al. only allow spatial camera interpolation. Our approach also allows interpolation in time and works with unsynchronized cameras. Recently, a free viewpoint system for static scenes has been proposed by Hornung et al. [18].

Other approaches are based on Optical flow. Referring to a certain flow field created by spatiotemporal trajectories of image-regions within an image sequence, this technique can be used to interpolate between two or more images. Several approaches [19] using these flow fields have been proposed in the last years creating promising interpolation results.

Recently, Stich proposed a novel approach [3] outperforming optical-flow based interpolation algorithms in quality and IBR-algorithms in configuration effort. Since our video player is based on his approach, we will discuss it in more detail in Section 3.

3. Virtual Video Camera

Our video player utilizes the image interpolation algorithm proposed by Stich et al. [3].

Stich's basic assumption is that robustly matching visible edges is the key to visually plausible image interpolation. First, he finds predominant edges [20] and homogeneous regions [21] in two images. A bipartite matching of edge pixels in the two images yields sparse correspondences. For each homogeneous region these correspondences define the overall transformation into the other image. The resulting dense deformation vector fields, containing 2D float data, describe a complete $1:1$ relation between the pixels of the source and target image. Therefore, they can be applied by any factor between 0 and 1 to create as many in-between images as desired.

Since our goal is not only to interpolate between two views of a scene, but to freely change viewing direction and playback rate, we use an extension of Stich's original rendering. We represent each input image by a three-dimensional vertex, encoding horizontal and vertical viewing direction as well as playback time.

The resulting point-cloud can be tessellated using the Delaunay-tessellation. Within this mesh, each imaginable point lies in exactly one tetrahedron (Fig. 1). Thus, interpolation-weights as demanded by Stich's interpolation approach can be easily estimated by computing the barycentric-coordinates of the desired view-point in its tetrahedron.

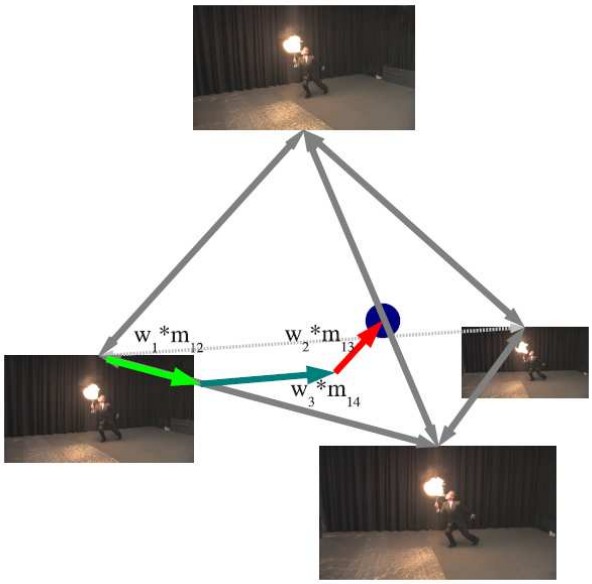

Figure 1. Warping within a 2D or 3D environment. The green, turquoise and red arrows describe the weights, the gray the correspondence fields m_{ij} and m_{ji}. The blue circle is the target point.

Let us assume we like to render an image v at an arbitrary viewing direction and point of time. Due to the Delaunay triangulation, v lies inside a single tetrahedron λ. Rendering of this image is done by forward warping and blending of all four input images $v_1, ..., v_4$ associated with λ (Fig. 2). E.g., for image v_1 rendering is done as follows. For every pixel position in the correspondence maps m_{ij} from the images v_i to v_j, a vertex is created and displaced according m_{12},

m_{13} and m_{14}. These displacements are weighted by the barycentric coordinates $w_1, ..., w_4$ of v in λ, resulting in a forward warped image $v_1^{'}$. The same procedure is applied to $v_2, ..., v_4$. The final image is created by blending $v_1^{'}, ..., v_4^{'}$, again weighted by $w_1, ..., w_4$. For further information, e.g. occlusion handling and image alignment, we refer to [3].

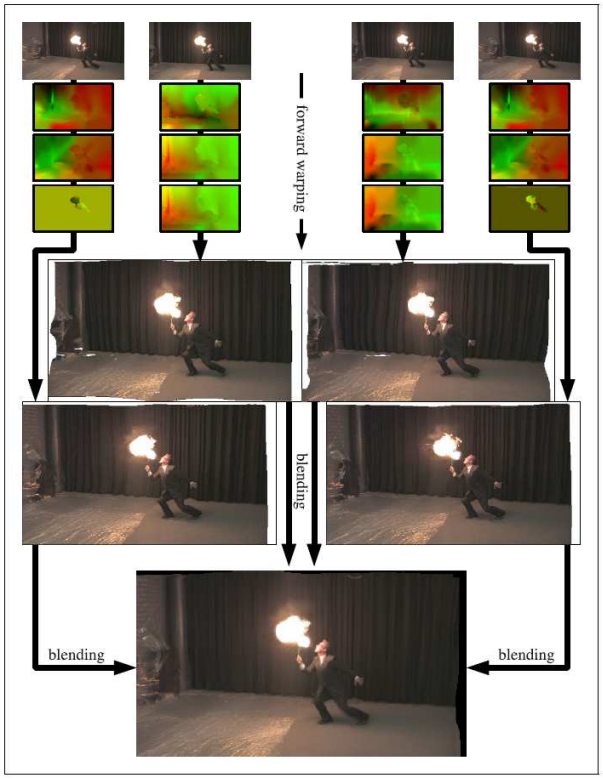

Figure 2. Forward warping, with (from top to down): the original input images, the single correspondence fields (exported as red/green images), the forward warped input images and the result from blending the warped images.

4. Data Compression

As mentioned in the introduction, 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. Further, the appropriate correspondence fields, presented in [3], have to be processed on the fly. Thus, we will try to reduce the file size of the required data as much as possible while still preserving low decompression effort.

4.1. 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 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. Hence, the data needs to be compressed in order to gain further performance improvements and appropriate file sizes.

In order to reduce the file size of the correspondence fields, we benefit from existing and well known image compression algorithms. As for one warp, two float values have to be compressed, the HDR-image format OpenEXR comes to mind, as it is capable of dealing with floats directly. But even with the usage of 16-bit floats, as supported by OpenEXR as *half*, still the resulting files exceeded a manageable file size. The combination of these halfs with a float-compression algorithm such as PXR24 yields a computational effort regarding the decompression exceeding the maximum of 40 ms needed for framerates of 25 fps and higher. Similar restrictions apply to video codecs such as mpeg or H.264.

In contrast, using JPEG to compress the single warps guarantees small correspondence files and a short decompression. But as JPEG consists of three 8-bit (RGB) integer, a pre-transformation from float to byte is necessary. However, the hence made error, combined with the compression flaw, has a negative influence on the rendering result. Further, the third color channel remains unused. Thus, we decided to use the lossless PNG-compression instead.

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

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

and can be used to express the single displacement warps:

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

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:

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

The hereby made error is propotional to the stepsize and thus stays relatively small compared to the scene motion.

Furthermore, a PNG pixel consists of four channels (RGBA) while only two are needed for saving one pixel warp. Hence, this enables us to encode both correspondence fields resulting from comparing two images in a single file, reducing harddisk lookups.

Combining this float conversion with the PNG compression yields average file sizes of 13 KB (for small correspondence fields of 240x135). Compared to the raw float data of 506 KB, this is a stunning compression factor of nearly 39 with a very low decompression cost.

4.2. 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.

5. Implementation Details

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. 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.

For further performance improvements, the warp decompression and the image warping as well as the rendering are computed in separated threads. To avoid locking issues, each thread manages its own cache. Depending on user input, we estimate whether or not different frames or warpfields will be demanded in the next rendering loop. During GPU-rendering, the CPU can prefetch these data. A specifically designed user interface gives the user the opportunity to change view direction and control playback rate (Fig. 3).

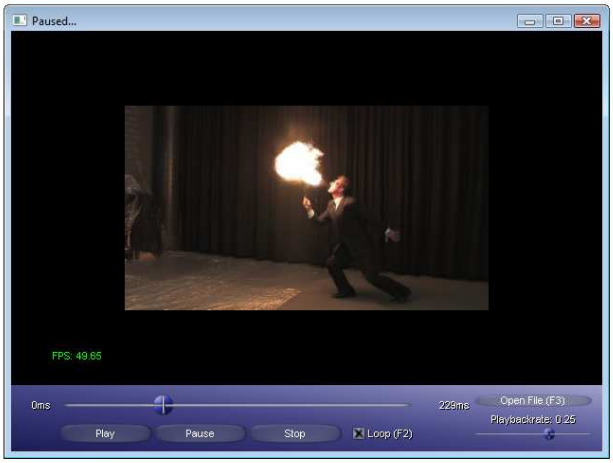

Figure 3. The graphical user interface of the free viewpoint video player. Resembling to a standard media player interface with play, pause and stop buttons as well as a progress bar, our player provides additionally the possibility to adjust the playbackrate. By dragging the left mouse button, the viewpoint can be changed interactively.

6. Results

Due to a low decompression effort and a compact data storage, as well as the in Section 5 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 standard 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 1.

Two different scenes have been captured with our technique, the firebreather sequence and the dancer sequence (see Fig. 4). We chose these two scenes because they pose different challenges to our system. The firebreather sequence contains dynamic illumination and volumetric effects. Whereas the dancer sequence exhibits large scene motion. In both examples we used the input of twelve cameras arranged in a 4x3 array.

	Resolution	Grid resolution	FPS (playback)
Low	480x270	480x270	$\geq$ 60 (27)
Med	480x270	480x270	$\geq$ 60 (35)
High	480x270	480x270	$\geq$ 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 1. 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 $\geq$60 FPS results from vertical sync being active limiting the rendering speed to the refresh-rate of the monitor.

Figure 4. Dancer test sequence with large scene motion.

6.1. Compression Rate

As mentioned before in Section 4.1, we achieve a high data compression rate, especially on the correspondence fields. For small correspondence fields (as 240x135), the raw 32-bit float data would have a size of $240*135*4*2*2$ $Byte \approx 506$ 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. 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 JPEG compression of the video files does not result in a further downsizing compared to standard video codecs, but is the only possibility to ensure a free navigation in time during playback. 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.

The impact of the data compression on the rendering quality is rarely noticeable. Hence, the output video looks as the offline results presented in [4]. We included a video of our FVV player and a Windows executable, downloadable at: http://graphics.tu-bs.de/publications/mvcWin32.zip (60 mb). For subjected quality assessment we would like to refer to this extra material.

7. 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 overall 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 1, 2

[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. 1, 2

[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. 1, 2, 3

[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 1, 5

[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. 1

[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. 1

[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. 1

[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. 1

[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. 1

[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. 1

[11] 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.

[12] 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 1

[13] P. Ratanaworabhan, J. Ke, and M. Burtscher. Fast lossless compression of scientific floating-point data. In *Proceedings of the Data Compression Conference,* 133–142, Washington, DC, USA, 2006. 2

[14] P. Lindstrom and M. Isenburg. Fast and Efficient Compression of Floating-Point Data. *Transactions on Visualization and Computer Graphics,* 1245–1250, 2006. 2

[15] 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. 2

[16] 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. 2

[17] S. Vedula. Image Based Spatio-Temporal Modeling and View Interpolation of Dynamic Events. *PhD thesis, University of Washington,* 2001. 2

[18] A. Hornung and L. Kobbelt. Interactive Pixel-Accurate Free Viewpoint Rendering from Images with Silhouette Aware Sampling In *Computer Graphics Forum,* 2090–210, 2009 2

[19] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. In *11th International Conference on Computer Vision,* 1–8, Rio de Janeiro, 2007. 2

[20] M. Ruzon, C. Tomasi. Color Edge Detection with the Compass Operator. In *Proceedings of Conference on Computer Vision and Pattern Recognition,* 160–166, 1999. 2

[21] Efficient Graph-Based Image Segmentation. P. Felzenswalb and D. Huttenlocher. In *International Journal of Computer Vision,* 59(2), 2004. 2