VIDEO COMPRESSION USING DATA-DEPENDENT TRIANGULATIONS

Burkhard Lehner[1], Georg Umlauf[2]
Geometric Algorithms Group
Department of Computer Science
University of Kaiserslautern
67663 Kaiserslautern, Germany

Bernd Hamann[3]
Institute for Data Analysis and Visualization (IDAV)
Department of Computer Science
University of California, Davis
CA 95616-8562, USA

ABSTRACT

We present a method for compression of video clips using data-dependent triangulations. This technique utilizes the time coherence of a video to transfer information from one frame to the next, reducing the computation time for the compression. The results of this method are compared to those obtained with MJPEG and MPEG-2.

KEYWORDS

Video compression, data-dependent triangulations, comparison to MPEG.

1. INTRODUCTION

In today's multimedia environments, video clips play an important role, since the resolution of digital video cameras and low-cost web-cams have increased, and web 2.0 platforms have led to a wide propagation of transmitting video content over the internet, and storing them on servers. This leads to high memory and transmission bandwidth requirements. To handle these, the content can be compressed. Lossy compressions achieve high compressions ratios, while preserving the essential information for human perception.
We use data-dependent triangulations to store the frames of a video clip in a compressed form, but its computation requires substantial computational effort. To speed this process up, we make use of the time coherence of a video, reusing the optimized triangulation of a frame for the next frame. The compressed frame can be displayed efficiently on current graphics hardware with fast rendering of shaded triangles.
In Section 2 we review some methods to construct data-dependent triangulations for images, and describe some wide spread video compression approaches. After defining our notation in Section 3 we describe in Section 4 our method of triangulation reuse for the compression. Section 5 shows some timing and numerical approximation quality error results, and compares our method to some other video compression methods.

2. RELATED WORK

A color image can be stored in compressed form by a triangulation of the image domain approximating the image when interpolating the color of the triangle vertices linearly within each triangle. Only the triangula-

[1] lehner@informatik.uni-kl.de
[2] umlauf@informatik.uni-kl.de
[3] hamann@cs.ucdavis.edu

tion and the vertex colors need to be stored to reconstruct the image approximation. The task is to find a triangulation with a small number of vertices and a low approximation error.
Algorithms to find such a triangulation have been discussed in Demaret et al. (2006), Garland & Heckbert (1995), and Hoppe (1996) where greedy strategies were used to minimize the approximation error. These methods can get stuck in local minima. Another approach to optimize a triangulation is to apply a series of local modification operations to an initial triangulation. Some quality measure is used to judge the improvement achieved by the operation. In Dyn at al. (1990) and Schumaker (1993) the only operation is the edge swap. The vertices' positions are defined by the initial triangulation. To determine the next operation, a greedy method is used. To escape local minima, Schumaker (1993) used the concept of *simulated annealing.*
Kreylos & Hamann (2001) introduced a *vertex move* as additional modification operation. Optimizing the connectivity *and* the position of the vertices further improves the approximation quality. Lehner et al. (2007) used a set of heuristics, so-called *guides*, to determine the next modification operation to reduce the number of necessary iterations significantly. The resulting method is called *guided simulated annealing (GSA).*
The most common compression format for video compression is MPEG-2, as described in Watkinson (2004). It takes advantage of pixel coherence in space and in time. Each frame of the video is stored as one of three types: *I-frames* simply store a frame as a JPEG image, see Pennebaker & Mitchell (1992). *P-frames* store the difference to the preceding reference frame (I-frame or P-frame) using motion vectors. *B-frames* use a preceding and a following reference frame to further improve the prediction and reduce the required storage space. MotionJPEG (MJPEG) is a simpler video compression standard than MPEG. Every frame is stored as I-frame ignoring time coherence. The compression results are usually worse than MPEG-2.

3. NOTATION

A video **V** is a series $(f_1,\ldots,f_n)$ of frames, where every frame f_i: $\Omega\to\mathrm{IR}^3$ is an image in the domain $\Omega = \mathrm{IN}_w \mathrm{x} \mathrm{IN}_h$ of width w and height h pixels. Each pixel is mapped to a color, represented by the L, a, and b value of the CIEL*a*b* color model, see Wyszecki & Stiles (1982).
A triangulation $t = (V, E, F)$ consists of a set of vertices $V = \{v_1,\ldots, v_m\}\subset\Omega$, edges $E\subset V^2$, and faces $F\subset V^3$ which form a simplicial complex. The approximation A_t: $\Omega \to \mathrm{IR}^3$ induced by a triangulation t of frame f_i is the piecewise linear interpolant of f_i at the vertices. The approximation error $\Delta_G(t, f)$ of a triangulation t for a frame f is the *root mean square error* (RMSE) or the widely used *peak signal noise ratio* (PSNR) [dB]

$$\Delta_G(t,f)=\sqrt{\sum_{(x,y)\in W}\left(A_t(x,y)-f(x,y)\right)^2/(h\cdot w)},\qquad \mathrm{PNSR} = 20\,\log_{10}\left(\Delta_{\max}/\Delta_G\right),$$

where $\Delta_{\max}$ is the maximum possible peak signal, which is $\Delta_{\max}$=150.584 for the CIEL*a*b* color space. Thus, an approximation with lower Δ_G or higher PSNR has a better approximation quality. For videos a PSNR of 30–40 usually is sufficient, since the human eye can see every frame for only a fraction of a second, and is not able to detect approximation artifacts during that time.

4. VIDEO COMPRESSION

For compressing a video **V,** every frame f_i is compressed using guided simulated annealing (GSA). If k is the number of guided simulated annealing iterations used to find a good triangulation for a frame f_i, compressing **V** has complexity $O(nk)$ if we start from scratch for every frame. We call this method *brute force* (BF).
The difference between two successive frames f_i and f_{i+1} is usually small, since objects and cameras move continuously. Therefore, to speed up the calculation of t_{i+1}, we can use the triangulation t_i of frame f_i as initial triangulation for the GSA process for f_{i+1}. If the difference between f_i and f_{i+1} is small, t_i will likely be a good initial triangulation for f_{i+1}.
If a GSA process on an image starting with an arbitrary triangulation requires k_1 iterations, using a good initial triangulation can reduce this number to $k_2 \ll k_1$. Since f_0 has no preceding frame, calculating t_0 requires k_1

iterations, but for all other frames, k_2 iterations are sufficient. The time complexity to compress the video **V** is $O(k_1 + (n - 1)\, k_2)$. We call this method *triangulation reuse* (TR).
One disadvantage of TR is that the first frame is processed with $k_2 \gg k_1$ iterations, because there is no preceding triangulation we can use as initial triangulation. If in a real-time scenario the frames arrive with a constant delay, the following frames would have to be buffered before being processed. A variant of the TR approach is to use $k_1 = k_2$, i.e., a small number of iterations also for the first frame. Consequently the compression time for every frame is about the same. We call this variant *triangulation reuse to go* (TRG). Presumably, t_1 of TRG has a lower quality than t_1 of TR, and since it starts with an initial triangulation of lower quality, also t_2 of TRG will probably have lower quality than t_2 of TR. Our experiments have shown that the quality of TRG catches up with that of TR after a few frames, see Section 5 and Figure 3. If it is acceptable that the first few frames of a video have a lower quality and if it is inevitable that the compression time for all frames is about the same, TRG is an alternative to TR.
To store the compressed video, we simply use the file format described in Lehner et al. (2007). Since the number of vertices is the same for every t_i, the storage space for the vertex positions and the vertex colors is exactly the same. The connectivity using EdgeBreaker of Rossignac (1999) requires two bits per triangle, and since the number of triangles is about twice the number of vertices, this is also about the same for every t_i. Spending some extra bits we can achieve a constant size of every t_i and by this get quick random access to every frame.

5. RESULTS

We used one small video clip **V** we shot on our own to test our method. It shows a man standing in front of a white wall, moving his head and arms and talking. Its resolution is 640 x 480 pixels. It consists of 203 frames recorded at 17 frames per second, i.e., 11.9 s length. Figure 1 shows one frame f_i of each clip on the left, the corresponding triangulation t_i in the middle, and the induced approximation A_{t_i} on the right.
The original video, together with the results of the compression (using our TR method, MJPEG, and MPEG), can be found in the supplemental material at `http://www-umlauf.informatik.uni-kl.de`.

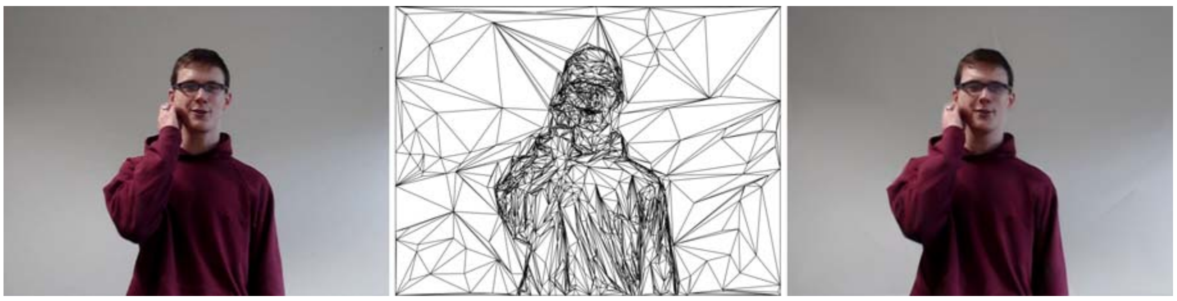
Figure 1: Video **V** compressed with 1500 vertices per frame: original frame f_i (left), triangulation t_i (middle), approximation A_{t_i} (right).

We compare the methods described in Section 4: brute force (BF) with k = 300,000 iterations, triangulation reuse (TR) with k_1 = 300,000 and k_2 = 10,000 iterations, and triangulation reuse to go (TRG) with $k_1 = k_2$ = 10,000 iterations. For comparison, we also compressed the videos with the brute force method, but only using k = 10,000 iterations per frame (brute force short, BFS). This approach requires the same number of iterations as TRG, but starts with an arbitrary triangulation for each frame. The experiments were performed on an Intel Core Duo CPU, 1.73 GHz, 2 MB cache, 1 GB RAM. All triangulations have 1,500 vertices.
The diagrams in Figures 3 and 4 show the approximation quality for every frame of the compressed videos for the four methods. The quality is measured as the peak signal noise ratio (PSNR) for every frame. As expected, the BF method leads to the best results but its execution time is up to a factor of 30 larger than that of the other methods, see Table 1. The TR method produces results that are of less quality when compared to those obtained via BF, but is much better than BFS, although it uses the same number of iterations per frame except for the first frame. The TRG method has similar results as TR, and uses the same number of iterations with the same computation time as BFS. TRG has a lower quality than TR for the first few frames of the vid-

eo, but after only a few frames it equalizes. For **V** this happens at frame f_4, see Figure 3. At frame f_5 TRG has even a higher quality than TR, supposedly due to the random nature of simulated annealing.
For **V** every compressed frame requires 8,830 bytes, leading to a total of 1,792,506 bytes for the whole video.

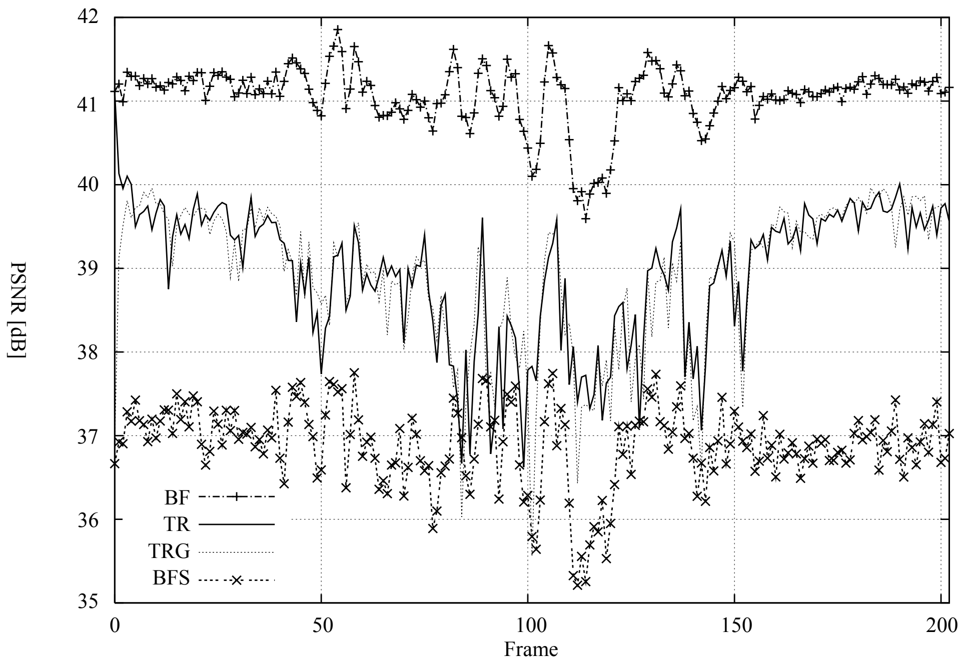

Figure 2: Approximation quality for **V** of the four described approaches.

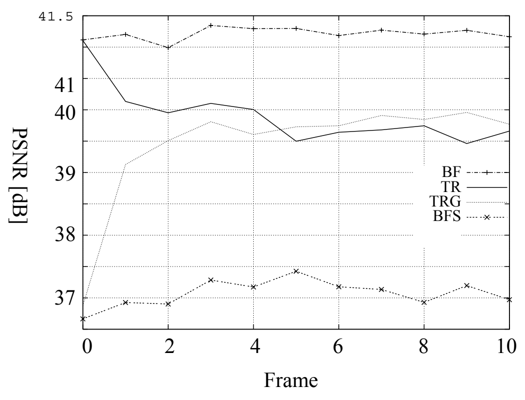

Figure 3: First eleven frames of Figure 2.

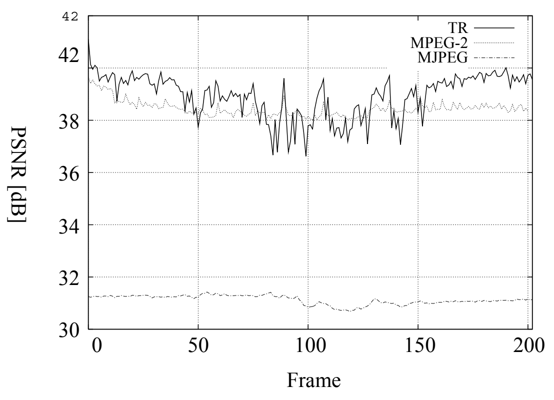

Figure 4: Comparison of TR, MJPEG and MPEG-2 for **V**.

To compare our video compression methods with state-of-the-art algorithms, we compressed **V** using MJPEG and MPEG-2 compression, using the software `mencoder` with the `lavc codec`. We adjusted the compression parameters to produce a file of about the same size. For **V** we used the parameter `vqmin=15` to produce an MJPEG video file of size 1,820,354 bytes. For the MPEG-2 video compression we had to use a rate of 25 frames per second, since frame rates of 17 are not supported, but this does not affect the approximation quality.
Figure 4 shows a comparison of the approximation quality frame by frame of the TR method described in this paper to the results of the MJPEG and MPEG-2 videos. Table 1 shows the numerical results. For these videos, TR was superior to MJPEG, and it was even slightly better than MPEG-2, although MPEG-2 utilizes time coherence to compress the output even more, which is not yet the case for TR.

Table 1: Results of compression of **V**.

	BF	BFS	TR	TRG	MJPEG	MPEG-2
Iterations	61M	2,030k	2,320k	2,030k	——	——
Time	11h38’	22’14”	25’40”	22’20”	——	——
PSNR [dB]	41.1	36.9	38.9	38.8	30.6	37.9

6. CONCLUSIONS AND FUTURE WORK

Our results show that the triangulation reuse technique presented in this paper reduces the computation time for compressing video clips using data-dependent triangulations.
To further reduce the computation time one could estimate the optical flow to predict an even better initial triangulation, or one could try to parallelize the calculations of the GSA: Several local modification operations could be performed in parallel, as long as their areas of modification do not interfere.
To further improve the approximation quality or the compression ratio one could utilize the time coherence of the successive frames. For this purpose, only that part of the triangulation that changes from one frame to the next could be transmitted. Furthermore, the color at the vertex positions could be estimated from the approximation of the last frame. Only the difference between estimated and actual color has to be compressed using arithmetic coding. Another approach is to use volume elements (e.g., tetrahedra) to approximate a volumetric data set, where two dimensions are the pixel coordinates of a frame, and the third dimension represents time.

ACKNOWLEDGEMENT

This work was supported by DFG IRTG 1131, “Visualization of Large and Unstructured Data Sets,” University of Kaiserslautern, and NSF contract ACI 9624034 (CAREER Award) and a large ITR grant, University of California, Davis. We thank the members of the Visualization and Computer Graphics Research Group at IDAV. In particular we express our special thanks to Oliver Kreylos, for his support at the beginning of this project, and Klaus Denker, for his help on the implementation of the software.

REFERENCES

L. Demaret, N. Dyn, and A. Iske (2006). Image compression by linear splines over adaptive triangulations. *Signal Process.*, 86(7):1604–1616.
N. Dyn, D. Levin, and S. Rippa (1990). Data dependent triangulations for piecewise linear interpolations. *IMA Journal of Numerical Analysis*, 10(1):137–154.
M. Garland and P. Heckbert (1995). *Fast polygonal approximation of terrains and height fields*. Technical report, CS Department, Carnegie Mellon University.
H. Hoppe (1996). Progressive meshes. In *SIGGRAPH ’96*, pages 99–108, 1996.
O. Kreylos and B. Hamann (2001). On simulated annealing and the construction of linear spline approximations for scattered data. *IEEE TVCG*, 7(1):17–31.
B. Lehner, G. Umlauf, and B. Hamann (2007). Image compression using data-dependent triangulations. In G. Bebis, editor, *Int. Symposium on Visualization and Computer Graphics 2007*, pages 351–362.
W. B. Pennebaker and J. L. Mitchell (1992). *JPEG: Still Image Data Compression Standard*. Springer.
J. Rossignac (1999). Edgebreaker: Connectivity compression for triangle meshes. *IEEE TVCG*, 5:47–61.
L. L. Schumaker (1993). Computing optimal triangulations using simulated annealing. *Computer Aided Geometric Design*, 10(3-4):329–345.
J. Watkinson (2004). *The MPEG Handbook, Second Edition*. Focal Press.
G. Wyszecki and W. Stiles (1982). *Color Science: Concepts and Methods, Quantitative Data and Formulae*. John Wiley & Sons.