EUROGRAPHICS 2011 / M. Chen and O. Deussen
(Guest Editors)

Volume 30 (*2011*), *Number 2*

Scalable Remote Rendering with Depth and Motion-flow Augmented Streaming

Dawid Pająk[1,2] Robert Herzog[2] Elmar Eisemann[3] Karol Myszkowski[2] Hans-Peter Seidel[2]

[1]West Pomeranian University of Technology, Szczecin, Poland
[2]MPI Informatik, Saarbrücken, Germany
[3]Telecom ParisTech, Paris, France

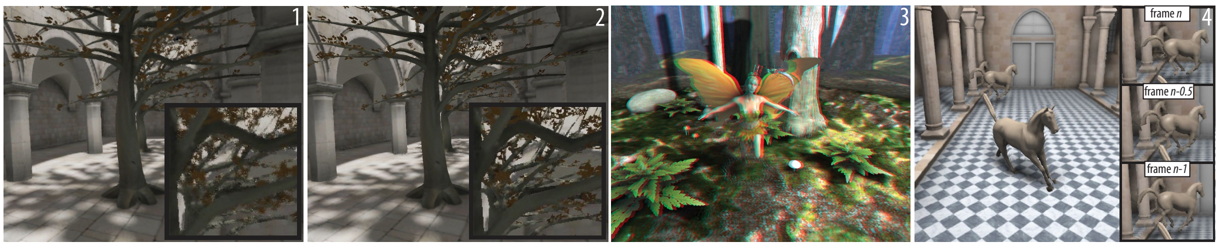

Figure 1: *Remote rendering allows navigating in complex scenes even on weak client hardware. But not only final images are of interest on the client side, auxiliary information like depth or motion become increasingly attractive in this context for various purposes. Examples include spatio-temporal upsampling (1, 2), 3D stereo rendering (3), or frame extrapolation (4). Standard encoders (H.264 in image 1) are currently not always well-adapted to such streams and our contribution is a novel method to efficiently encode and decode augmented video streams with high-quality (compare insets in image 1 and 2).*

Abstract

In this paper, we focus on efficient compression and streaming of frames rendered from a dynamic 3D model. Remote rendering and on-the-fly streaming become increasingly attractive for interactive applications. Data is kept confidential and only images are sent to the client. Even if the client's hardware resources are modest, the user can interact with state-of-the-art rendering applications executed on the server. Our solution focuses on augmented video information, e.g., by depth, which is key to increase robustness with respect to data loss, image reconstruction, and is an important feature for stereo vision and other client-side applications. Two major challenges arise in such a setup. First, the server workload has to be controlled to support many clients, second the data transfer needs to be efficient. Consequently, our contributions are twofold. First, we reduce the server-based computations by making use of sparse sampling and temporal consistency to avoid expensive pixel evaluations. Second, our data-transfer solution takes limited bandwidths into account, is robust to information loss, and compression and decompression are efficient enough to support real-time interaction. Our key insight is to tailor our method explicitly for rendered 3D content and shift some computations on client GPUs, to better balance the server/client workload. Our framework is progressive, scalable, and allows us to stream augmented high-resolution (e.g., HD-ready) frames with small bandwidth on standard hardware.

1. Introduction

Server-client platforms that enable remote 3D graphics interaction are of high importance in the age of mobility and constant hardware change. By relegating rendering to powerful servers and streaming of the resulting frames, the requirements on the client side can be reduced to standard web browsing and video playing abilities.

Server-client graphics platforms have many more advantages, e.g., potentially huge data sets remain on the server which simplifies maintenance, improves security, and enables consistent updates in collaborative work scenarios.

Consequently, numerous applications can benefit from this setup. Examples include medical 3D visualization, industrial and architectural design, at the spot repair of complex devices (e.g., ships and aircrafts), 3D navigation and tourism, and in particular online entertainment and gaming. In the latter case, players subscribe to an instant access of a large variety of games. Server-side execution makes client-hardware upgrades, installation and software updates mostly unnecessary. These advantages are attractive for customers, as well as game developers that benefit from reduced marketing / distribution costs and piracy risks. Not surprisingly, a number of companies such as OnLive, OTOY, Gaikai are committed to provide technology for game streaming.

The key problems of such an approach are the server workload that limits scalability with respect to the number of clients and bandwidth. Particularly, standard compression schemes are often expensive and introduce a long latency. HD-ready-resolution frames need approximately 3 Megabits per second for good quality encoding using traditional streaming with H.264. However, usage of MPEG B-frames introduces delay which is unacceptable for real-time applications, like games. Low-delay streaming requires more data transfer (e.g., OnLive recommends 5 Megabits or more for HD-ready), but effects such as 3D stereo, or high frame rates, would quickly lead to an excessive bandwidth.

Our insight is that transfer costs can be reduced by augmenting the stream with supplementary information, such as depth or motion. These *attribute buffers* are very cheap to compute on the server, and allow many applications when transferred to the client, including warped 3D stereo, spatio-temporal upsampling (super-resolution), and frame extrapolation. All of these benefit from the client-side execution — 3D (acceptable disparity depends on eye distance and device size), resolution and framerate can be optimized for the client's equipment or preferences. The only problem is that such applications require high precision attribute buffers which makes standard compression schemes difficult to apply. Our approach addresses this issue by linking rendering and compression, while existing streaming systems (e.g., Reality Server, OnLive) benefit little from the fact that a 3D model is underlying the rendering.

While we follow the (usually valid) assumption that the client machine is much less powerful than the server, a pure video-streaming scenario makes almost no use of the client's computational capabilities. Instead we want to exploit this resource. To transfer high-precision auxiliary data (e.g., depth) to the client, our solution uses an efficient edge encoding and a new fast diffusion process. Consequently, on the server, costly raytracing or per-pixel shaders are only applied to low resolution frames, that are streamed to the client where the corresponding high-resolution frames are reconstructed, hereby lowering server workload and bandwidth.

Precisely, our contributions are:

- **Compression** – an efficient scheme not relying on future frames and a novel inter frame prediction;
- **Precision** – high-quality discontinuity edges to enable spatio-temporal upsampling;
- **Decompression** – a novel GPU-adopted diffusion scheme and decoding;
- **Transfer** – a solution to adapt bandwidth.

We illustrate the advantages of our motion-augmented video streaming with several client-side applications (e.g., stereo rendering, temporal super-resolution).

2. Previous Work

Only few successful attempts exist that couple 3D rendering, efficient data compression and streaming. Cohen-Or et al. [COMF99] and Levoy [Lev95] enhance client rendering by streaming residuals (difference between high-quality-server and low-quality-client frames). The server workload increases because residuals require both, the client and server-side rendering. Also, the client needs to receive a 3D scene description. In the limit, one could stream only graphics API calls [NDS*08], but powerful clients are needed and server-side rendering is impossible.

We execute scene-related costly computations on the server. The client's cost depends only on the image resolution. To reduce server costs, we build upon amortized rendering techniques [SaLY*08, YSL08, HEMS10]. Usually on a single machine, we employ them in a streaming context and let the client reconstruct the image.

We can also exploit knowledge concerning the final compressed video. In the Render2MPEG framework [HKMS08], details that would be removed during compression are not rendered. Similarly, we can use a low bandwidth or limited client display capabilities to reduce the server load.

One major contribution of our work is a novel depth encoding. In particular, free viewpoint video (FVV) and 3D television [KAF*07] (3DTV) require the so-called multiview video (MVV) representations, which often involve the depth data to warp the original video stream to nearby virtual camera positions. Because video codecs such as H.264 are optimized for image statistics and human perception, depth compression requires specialized solutions [MMS*09]. Many approaches smooth depth values in order to increase compression performance [Feh04] at the cost of crucial precision. Lossless depth compression via a run-length like encoding [JWB04], or by integrating an underlying 3D model [EWG99] might not easily meet bandwidth constraints. Consequently, we aim at high accuracy, but allow a tradeoff between quality and compression to meet bandwidth limitations.

Precision is most crucial at depth discontinuities [MMS*09]. Edge-preserving filtering [PJO*09] changes depth values, but lets the encoder focus more on discontinuities. Also, region-of-interest specifications and depth-value redistribution can improve quality [KbCTS01]. Nonetheless, depth discontinuities are only handled implicitly in the encoding step which can lead to significant divergence [MMS*09].

Merkle et al. [MMS*09] introduced a different depth-optimized encoding for adaptive pixel blocks (possibly separated by a single linear edge) and assign a constant or linear depth approximation (if present for both edge sides). Fitting the edges and constructing the representation is costly, but leads to a quality leap. We avoid explicit optimizations and show that we achieve higher quality with higher encoding efficiency. We also introduce a temporal prediction which is very effective for animated sequences.

The importance of depth discontinuities is also illustrated by approaches that perform joint color / depth compression [MD08] because of the often strong correlation. In similar spirit, we will rely on depth information to perform spatio-temporal upsampling to reduce the server workload, but instead of a semi-automatic solution [MD08], we need an automatic and sufficiently efficient online solution.

Image encoding with H.264 is computationally expensive. Half the time is used on neighborhood matching and motion estimation [CBPZ04] that can be directly recovered from 3D [FE09] and enable higher compression [WKC94]. In our solution, we rely on motion vectors to enable spatio-temporal upsampling in dynamic scenes. Their high redundancy [MF98] allows for an efficient encoding.

Our solution benefits from the fact that image information is strongly correlated with discontinuities [Eld99]. Edges can be beneficial for compression [GWW*08], but the construction and reconstruction steps are usually costly. The locality of our approach makes it efficient and ready for an online rendering context.

3. Augmented Streaming Framework – An Overview

Our algorithm consists of several components, which are depicted in Figure 2. On the server side, we render a jittered low-res. version H_t^{low} of the current frame H_t. Avoiding the production of a full-res. frame reduces the server workload and saves bandwidth to be used for our additional streaming data. The image H_t^{low} is encoded using a state-of-the-art H.264 video encoder and sent to the client who constructs a high-res. version H_t based on the previously reconstructed high-res. frame H_{t-1}. Key for this upsampling are high-res. depth information of the current frame D_t and the motion flow M_t between the current and previous frame. Both are cheap to compute on the server [SaLY*08].

To efficiently transfer depth D_t and motion M_t to the client, we rely on a customized compression scheme. We detect discontinuities that define edges on the server (Sec. 3.1). The edges and their depth and motion values

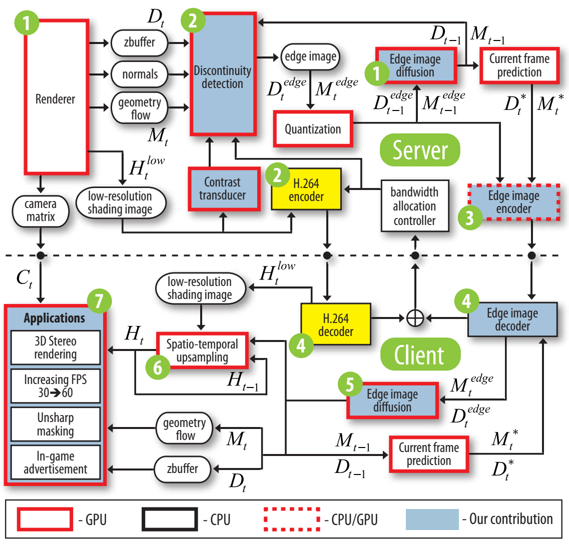

Figure 2: *Outline of the proposed streaming architecture. Green markers indicate the computation order within a frame cycle.*

(D_t^{edge}, M_t^{edge}) are encoded rapidly using previous information (D_{t-1}, M_{t-1}), then sent to the client. The client reconstructs a high-res. depth D_t and motion flow image M_t by diffusing the sparse edge information (Sec. 3.2), hereby exploiting smoothness of the signal between discontinuities. The client computes the final image H_t using the low-res. image H_t^{low}, the depth, and motion flow (Sec. 3.3).

Our compression scheme outperforms state-of-the-art encoders (Sec. 5) and enables a high-quality spatio-temporal reconstruction. We propose a bandwidth control based on a perceptual metric (Sec. 3.4) and provide implementation details of our low-level encoding process (Sec. 3.5). The relatively accurate depth D_t and motion M_t data, enable many applications on the client side that were difficult to achieve with previous encoding strategies (Sec. 4).

3.1. Server-Side Preparation

It is very cheap to produce high-res. *attribute buffers* (depth, surface normal, motion, and texture) which is typically done for deferred shading. The main server cost stems from producing the final shaded image, especially when using ray tracing. Using a lower resolution image H_t^{low} leads to a performance gain. We then stream H_t^{low} via H.264.

A H.264 codec is not optimal for high-res. depth D_t and motion M_t whose precision is crucial for the client-side upsampling that transforms H_t^{low} into a high-res. high-quality output H_t. Also other applications (Sec.3.3) would suffer. In contrast to *standard* image information, depth or motion have little visual meaning and traditional video-coding (e.g., MPEG) is perceptually-tuned. Instead, discontinuities in the attribute buffers should be preserved, while precision can be lowered for the remaining image. E.g., frame extrapolation with a low-quality motion flow will be most perceivable at contrast-discontinuities or region boundaries [SLW*08]. We will show that D_t and M_t can be compressed and streamed efficiently by relying on a customized edge-based compression and client-side reconstruction via diffusion. The result closely resembles the original on edges and smoothly interpolates between them.

Edges are detected from all attribute buffers. One could use a gradient-based edge detector, but it often fails in practice. E.g., a flat surfaces seen at a grazing angle is detected as discontinuity. Instead, we rely on a Laplace operator. For performance reasons, we detect edges from depth and motion-flow buffers only. A weighted sum of responses from all buffers yields pixel significance. Predefined threshold selects significant pixels, the so-called *edge samples* (in Sec. 3.4, we show how to improve upon this ad-hoc threshold). To avoid missing low frequency changes, we also uniformly add one sample every 32×32 pixels.

We do not define curves explicitly and only select edge samples, but a Laplacian detects edge structures (Fig. 4) well. Further, it always extracts two-pixel-wide edges. Consequently, we keep two values, one for each side along the discontinuity. Only in this setting, a diffusion process can reproduce the discontinuity. Besides depth and motion, we store the sign of the Laplacian detector (zero-crossing of 2nd derivative) because — almost always — neighboring pixels with the same sign also lie on the same surface. This information will be used to improve the compression efficiency.

3.2. Client-Side Reconstruction via Diffusion

The client receives a low-res. shading image, and edge-based representation including values for depth and motion. In this section, we propose a fast diffusion scheme used to reconstruct high-res. depth and motion information.

The basic idea of our method is a push-pull mechanism [GGSC96]. The goal of the *push step* (Fig. 3, grey) is to fill holes by reducing resolution. We create a pyramid-like image representation by successive downsampling:

$$D(x,y) = \begin{cases} I(2x,2y) & ,w(2x,2y) > 0 \\ \frac{\sum_{x',y' \in N_{2x,2y}} w(x',y') I(x',y')}{\sum_{x',y' \in N_{2x,2y}} w(x',y')} & ,w(2x,2y) = 0 \end{cases} \quad (1)$$

where $N_{x,y} = \{(x-1,y-1),(x-1,y+1),(x+1,y+1),(x+1,y-1)\}$ stands for a local neighborhood, I is the edge-based image and w contains binary weights which are set to 1.0 for edge samples, and 0.0 for empty pixels. This operation extends the edge samples by one pixel in each level and holes are quickly filled. Due to the uniform edge samples (one sample per 32×32 pixels), five levels of the pyramidal downsampling are sufficient to entirely fill the top image in the pyramid.

After the push step, we use a *pull* operation (Fig. 3, orange) to propagate the filled hole back to the high-res. image. In a top-down manner, coarse-level samples are "pulled" back to their corresponding position in the finer level. We do not overwrite existing fine-level values as these were defined during the push step and are thus based on a more-precise higher-resolution image. The push step kept even pixel values, consequently, during the pull step, we only transfer values from these even pixels (red squares). The remaining pixels of the current pyramid level are filled in local diffusion steps (we apply a diagonal (cyan) then axis-aligned diffusion which performed best in our tests).

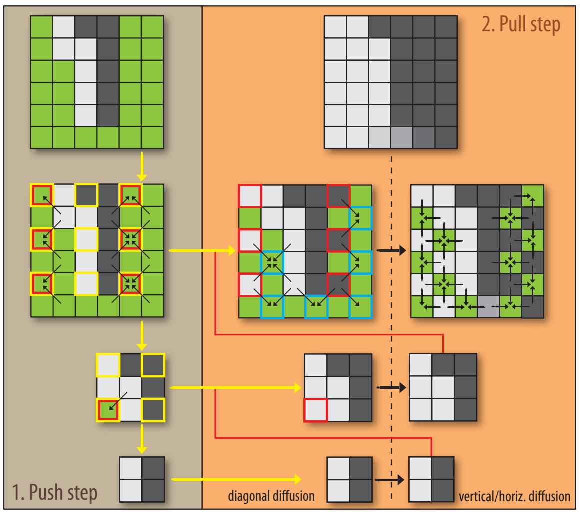

Figure 3: *Our diffusion scheme. Samples stored on both sides of an edge are spread over the image in two stages. 1) Push phase: downsample image (yellow rectangles). Try to fill empty (green) samples by averaging diagonal neighbours. 2) Pull phase: coarse level images fill in holes in finer scales. First, transfer copied values (previously yellow, now red rectangles). Fill empty pixels (blue rectangles) by averaging diagonal, then adjacent samples.*

To further improve the smoothness of the output surface, similar to multi-grid methods, one can apply a pre or post-smoothing step on each level in the *pull* stage. A single Jacobi iteration reduces local diffusion errors, but also slightly increases the average approximation error. Compared to standard push-pull, our diffusion has reduced anisotropy (due to *quincunx lattice* sampling scheme) and respects edges. Additionally, we process screen-space linear input and therefore diffuse in a perspective correct way.

3.3. Client-Side Spatio-Temporal Upsampling

Having all data available on the client, the final high-res. shaded image needs to be constructed. One could simply scale the low-res. shaded image H_t^{low} to the screen resolution, but then the quality would be poor. Instead, one can significantly improve quality when performing a spatio-temporal reconstruction (following [HEMS10]).

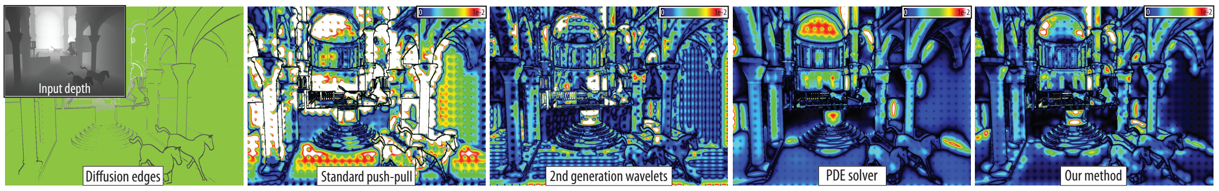

Figure 4: *Depth buffer edge-diffusion error visualization. Input edge image (left) has been processed with selected "inpainting" algorithms. Our method provides quality comparable to PDE solvers, yet it is much less computationally intensive. Also, it has the smallest relative error of approximation, which is particularly important for our problem. Measured PSNR: push-pull: 36.45dB, 2nd generation wavelets: 48.47dB, PDE: 52.42dB, our method: 52.49dB.*

To reconstruct the final image, the client relies on the reconstructed attribute buffers of the current frame: depth D_t and motion flow M_t. Using M_t, pixels in the current frame can be projected back into the previous high-res. frame H_{t-1} which is still present on the client. As in [HEMS10], a bilateral weighting scheme (screen position, depth, and distance to the samples in H_t^{low}) attributes weights to the pixels in the temporal and spatial neighborhood in H_{t-1} and H_t^{low}. The weighted sum defines the current frame H_t.

Disocclusions (pixels in H_t whose corresponding sample was either outside the viewing frustum or occluded in H_{t-1}) need to be considered in the weighting scheme. In this case, no reliable information exists in H_{t-1} and non-zero weights are only attributed to pixels in the current frame H_t^{low}.

The complete spatio-temporal upsampling is more complex than described, e.g., illumination gradients are used to treat dynamic lighting. The interested reader is referred to [HEMS10]. In particular, the server creates jittered frames that ensure a convergence to a high-quality rendering for static elements. The major difference is that we do not use normals to reduce bandwidth. We counteract this restriction with a twice bigger spatial reconstruction kernel for higher accuracy. This larger support is possible because of our streaming context: the client avoids the scene rendering and offers many unused resources.

3.4. Scalability

In an online rendering context, it is important to adapt the bandwidth dynamically. One could adapt the resolution of the rendered video stream effectively, hereby changing the upsampling ratio, or reduce the frame rate by making use of temporal upsampling (see Fig. 1) on the client. While the previous measures have a strong impact on the bandwidth, but also quality, one can also control the bandwidth/quality tradeoff on a finer level in the edge encoder.

When many edges arise from the discontinuity detection, our encoding becomes sub-optimal. Fortunately, not all edges are needed. "Edge importance" strongly relates to the perceptual contrast in image H_t. The more cluttered an image region becomes, the less precision is needed due to perceptual masking effects. Hence, we would like to weight geometric edges according to "visual importance". However, since this computation is executed on the server, we want to avoid high costs, therefore we focus on the available low-res. rendering H_t^{low}. Consequently, masking thresholds are computed via the GPU only for coarse scale edges on low-frequency bands of a Gaussian pyramid.

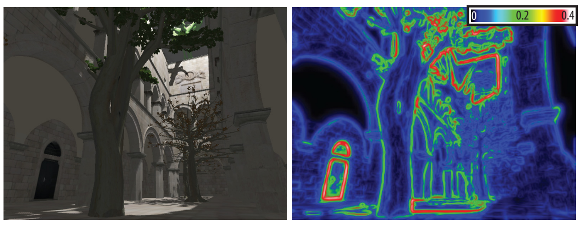

Figure 5: *A complex frame and the visual significance of its edges (dark blue=imperceivable, red=clearly visible). To meet bandwidth constraints, we favor important geometric edges. Note the small edge importance in the tree, which is dominated by visual masking.*

Precisely, we convert the low-dynamic-range image H_t^{low} to linear luminance (we ignore isoluminant chroma edges) and build a Gaussian pyramid. Using a Sobel filter, we compute the magnitude $c_l(x)$ of the gradients in each level. These gradients are analyzed for visual masking with a contrast transducer borrowed from JPEG2K [ZDL00]:

$$R_l(x) = \frac{|c_l(x)|^{0.6}}{1.0 + \sum_{i \in \mathcal{N}_8(x)} |c_l(i)|^{0.2}}, \tag{2}$$

where $\mathcal{N}_8(x)$ is the eight-pixel neighborhood around pixel x in each pyramid level. The apparent contrast map $R(x)$ is the sum of the band responses $R_l(x)$ (Fig. 5). The final edge importance that guides the edge-sample selection, is computed by multiplying $R(x)$ with the weights derived from the attribute buffers. Further, we blend the edge weights with the motion-compensated edge weights of the previous frame to enforce temporal coherence and reduce edge flickering.

By exploiting the new edge weighting, we can achieve a *constant bit rate control* (CBR) by increasing/decreasing the number of edge pixels. To this extent, we vary the edge thresholds by a rate control factor, which exponentially converges to the desired target bit rate. Furthermore, although not needed in our tests, one can increase/decrease the edge-data quantization (see Sec. 3.5).

3.5. Stream Compression

We showed how the edge-based representation allows us to transfer high-quality high-res. images, but, for this transfer to be efficient, we need to encode the data properly. This section explains our efficient compression technique. We pursue two goals: a fast encoding and a good compression.

We encode two elements, first, a binary image defining the existence of edge samples (1=edge sample, 0=empty); second, the edge-sample values that are stored in these pixels. The binary image compresses well and defines a topological layout. Edge-sample values (such as depth, motion-flow or edge laplacian "sign") are then stored in scanline order and the binary image allows us to attribute the values to their original image locations.

Binary Image encoding One of the most effective schemes for lossless binary image encoding is JBIG2 [TUITU99]. It employs adaptive binary arithmetic coding [WNC87] to reduce the size of the resulting bit-stream. Unlike Huffman coding, arithmetic coders can output symbols occupying less than 1 bit on average. Such high compression ratios are possible by modeling probabilities (symbol frequency tables) based on the spatial neighborhood of the encoded characters. The adaptivity of the scheme ensures that more frequent symbols will require less bits to encode. This *higher-order* modelling is especially suitable for compression of images showing regularly structured elements (e.g., fonts, simple shapes). Usually the spatial neighborhood consists of previously encoded pixels (assuming scanline traversal) – the values beyond the encoding position will not be available to the client when performing the decompression. For binary edge images, this restriction resulted in lower efficiency. Further, JBIG2 was designed for still image compression, whereas we want to exploit inter-frame coherence and temporal prediction. Therefore, we propose a different approach.

To improve the intra-frame compression, we split the binary image into four interleaved sets of pixels. During the first pass, we predict values using only previous pixels, but from the second pass on, pixels extracted from previous runs are available. For any compression position, we can thus rely on pixels that lie in the "compression future" (Fig. 6, pass 1–3). Intuitively, we can better predict where edges lie in the image. For edge-only images, samples from different passes show a high spatial correlation (Fig. 6, pass 3). The impact on the compression efficiency is significant. In practice, the first pass uses approx. 50% of the file size, while all three following passes are squeezed in the remaining 50%. In contrast to a standard JBIG2 encoding, our intra-frame scheme generates 25% smaller frames on average.

To exploit inter-frame coherence, previous-frame samples are added to the current sample encoding context by warping the previous frame into the current view. Any warping strategy would work, but we use the fast solution we also employ for stereo rendering (Section 4). We can then access values

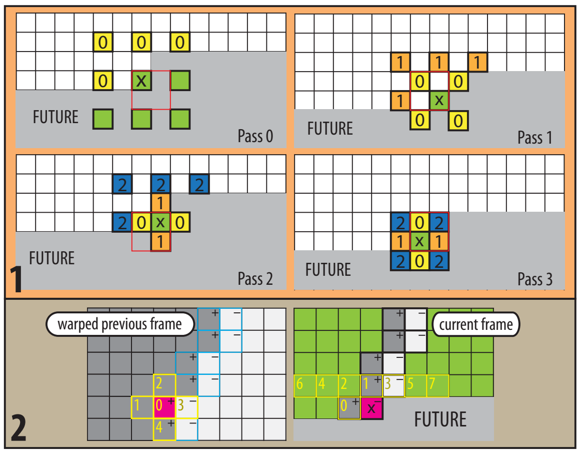

Figure 6: *Compression prediction schemes. 1) Spatio-temporal contexts for edge-image compression. To compute probabilities for sample 'x', a 9-neighbor spatio-temporal context is used, choosing from current pass, previous pass (numbered pixels) or the motion-compensated previous frame (green pixels). Combining values from these various origins keeps the prediction rates high. 2) Spatio-temporal value prediction. We select two potential predictors for current sample (purple cell in current frame): one from current and the other originating from warped previous frame. Candidates are chosen from a fixed neighborhood set (yellow cells). The first neighbor that matches the current sample's sign is selected as the "best" predictor.*

of the previous frame during en- and decoding (green pixels in Fig. 6). Depending on the quality of the warped frame, this context extension increases compression ratios between 1.2 and 7 (on average 2.5).

Edge-Sample Values To encode the edge-sample values (depth, motion vectors), we make use of the binary image. Basically, we group all edge-sample values in scanline order and compress this number stream. To reduce data entropy, we first apply a spatio-temporal predictor. The resulting residual stream is entropy coded before being sent.

Precisely, during the prediction step (Fig. 6, 2), we find a value that best matches the currently encoded sample. We assume that such a value will exists in a close spatial neighborhood and most likely will belong to the same geometric surface (same sign of the Laplacian operator). The residual is then computed by subtracting the match from the current pixel value. As depth discontinuities are highly correlated with motion discontinuities the best predictor for depth values will also be a good candidate for motion prediction.

In order to choose the best predictor we offer ourselves two choices, the first sample with the same Laplacian sign on the edge in the current or in the previous frame. In each step, we keep track of the used strategy in a table containing all possible 3×3-binary edge configurations. The strategy to choose a new sample is based on the corresponding entry, if current frames proved more successful, we apply this strategy, otherwise we use the previous frame. Afterwards we verify whether this choice was effectively the better one. We cannot do this before because we need a deterministic scheme that allows us to execute the same steps on the client for decompression. Based on the verification, we update our table entry which will affect the decision for the next similar context. In this way we probabilistically train a good predictor for each given context.

To reduce the range of the residual values, one can perform additional quantization which results in increased compression ratios and only slightly reduced accuracy. In our implementation, a quantization factor of 2 generates frames with approx. 90% of zero residual coefficients.

In order to capture low-frequency value changes, we added an additional edge-sample value every 32×32 pixels. These regular samples define a low resolution image that can be compressed separately. This way, we better preserve the edge-like nature in the remaining image which is exploited by our encoding algorithm. In practice, the low resolution image is encoded using the Paeth predictor [Pae91] followed by entropy coding.

Motion Flow Quantization For static objects, motion vectors are only a consequence of camera movement, which is known by the client. Hence, we encode only the non-camera movement of dynamic object pixels. We further quantize 2D motion vectors non-linearly by applying a power function (0.5), which favors slower motion, as these movements are most crucial for the upsampling process. Only fast moving dynamic objects might exhibit some small artifacts which are usually invisible because our detail perception on moving objects is much lower than for static elements. Consequently, the motion flow encoding proved less critical. Its spatial and temporal coherence leads to a good compression.

4. Applications

Besides the discussed spatio-temporal upsampling of lower resolution frames, the augmented depth/motion information allows for various other applications on the client-side. We will briefly introduce and discuss a few important ones.

3D Stereo Rendering Knowing the camera and depth is enough to compute 3D stereo via image-based warping. We use the technique proposed in [DRE*10], where a grid is overlayed with the depth buffer. The grid's vertices are snapped to the pixel corners between foreground and background depth-buffer pixels to reduce artifacts. We then transform this grid to simulate a left and right eye view. Disoccluded pixels can optionally be blurred to reduce the visibility of artifacts. The procedure runs entirely on the GPU and takes less than 2 *msec* (nVidia GTX 280) for the grid warping with a resolution of 320×180 vertices which is sufficient for an HD-ready resolution. In Fig. 1, image 3, the detailed leaves and the fly in the foreground are nicely warped, which shows the importance of precise depth edges.

Temporal Upsampling We can also interpolate or extrapolate frames in time (Fig. 1). Such an approach has been proposed in [DER*10]. The current frame is warped to the future by extrapolating the motion vectors and using the depth for occlusion. Of course, non-linear motion and motion discontinuities can result in overshoots. Such scheme is most sensible for high frame rates above 30 Hz to reduce LCD-display hold-type blur [DER*10]. Nevertheless, it is useful if frames are dropped during transmission.

Advertisement and Highlighting The depth buffer can be used for many interesting effects to highlight elements, e.g., Unsharp masking [LCD06]. Although such an operation could be employed on the server side, for sports events, where many clients share the same data stream, high-lighting can be done locally, thus allowing an observer to track their favorite player or adjust display settings (e.g., screen-space ambient occlusion is part of many display drivers, but not everybody appreciates and activates the mode). Another example is online advertisement. The supplementary streams make it possible to seamlessly integrate new content that can directly target the interacting user.

5. Results

We developed a proof-of-concept OpenGL prototype, computing all steps of our streaming system, however simulated without network component. We run the full application (Server and Client) on a desktop system equipped with an Intel Core 2 Quad Processor Q9550 and a NVIDIA Geforce GTX 280 graphics card and, separately, the client-side operates on a notebook computer with an NVIDIA Geforce 9400M GPU and an Intel Core 2 Duo 2.2Ghz CPU, a lower-end card that is comparable to modern mobile GPUs. We tested our approach on various challenging 3D scenes, of which we analyzed three in detail. The SIBENIK scene is fully dynamic with quickly changing lighting, but it has relatively simple geometry and depth (plus motion) compresses well using our scheme. The SPONZA scene consists of mostly static diffuse geometry, for which spatial upsampling is strongly beneficial assuming that depth is encoded precisely enough. However, it represents a worst-case scenario for depth compression because of the detailed tree model. Finally, the FAIRY scene is excessively textured and exhibits high-tessellated dynamic objects and complex motion trajectories.

In Table 1, we analyze the performance of our method and unfold the timings for each step of our algorithm. As illustrated in Figure 2, most of these steps map to GPU pixel shaders, only the H.264 video and final edge encoding are CPU based. Edge image encoding performs neighborhood matching on the GPU, writing the result in a texture that is read back to main memory, where we apply CPU arithmetic coding. Despite this CPU involvement, we already achieve real-time performance on mainstream hardware for server and client. Our client-side timings are achieved in milliseconds. Even on a weak client (notebook with low-end GPU), our decoding and upsampling at resolution 800×600 reaches realtime frame rates of 25–30 fps (15.5 ms for diffusion and upsampling on GPU, 25 ms for stream decoding on CPU). Moreover, the compression of motion and depth edges is relatively simple to perform for the server. Our scheme is suitable for low-cost client hardware and ready for today's portable technology. The server workload scales well for more clients by various means, particulary the upsampling factor enables strong speedups (see also supplementary material).

The video encoding of (low-res.) frames is performed on the highly optimized x264 CPU encoder. However, encoding performance depends on profile settings from which only "ultrafast" to "veryfast" settings achieved real-time rates on our hardware for HD-ready resolutions. Since we do not tolerate frame delay we can only use I- and P-frames (i.e., disable bidirectional motion-compensated B-frames) which, in combination with the two mentioned settings, yields poor quality for moderate bit rates (2 Mbit/sec). However, when encoding low-res. frames (as needed by our solution), "medium" or even "slow" quality settings achieve real-time performance.

Scene	Resolution	Server							Client		
		T_{ren}	T_{h264}	T_{ren}^{low}	T_{pre}	T_{enc}	*Fps*	*Speedup*	T_{dec}	T_{dif}	T_{up}
SIBENIK	SVGA	57	32.3	8.6	3.8	26.6	25.6	×2.3	19.0	1.3	1.2
	HD 720p	106	46.0	14.5	6.2	26.8	21.1	×3.1	20.3	1.5	2.3
	HD 1080p	251	73.7	29.1	13.9	47.1	11.1	×3.6	38.4	3.4	4.8
SPONZA	SVGA	63	32.4	9.1	3.9	29.7	23.4	×2.3	23.4	1.5	1.2
	HD 720p	120	45.0	15.1	6.9	35.2	17.5	×2.9	28.9	2.1	2.2
	HD 1080p	271	76.4	30.7	14.4	57.0	9.8	×3.4	48.0	3.9	4.7
FAIRY	SVGA	32	31.0	7.2	3.8	24.2	28.4	×1.8	16.0	1.3	1.1
	HD 720p	51	44.4	10.6	7.3	32.3	19.9	×1.9	24.6	2.6	2.1
	HD 1080p	107	72.2	16.7	15.7	54.6	11.5	×2.1	46.7	2.8	4.7

Table 1: *Server and Client timings (in msec) for different target resolutions:* 800×600 *(SVGA),* 1280×720 *(HD 720p),* 1920×1080 *(HD 1080p). Rendering costs are dominated by frame size and all frames are* 4× *upsampled. For comparison the costs for full-res. rendering (T_{ren}) and video-encoding with H.264 (T_{h264}) are provided together with the savings by our method in column (Speedup). Our server-side timings comprise: low-res. rendering (deferred shading) (T_{ren}^{low}), the time for previous frame depth/motion warping plus edge diffusion (simulating client steps) plus all other GPU steps involved in the pipeline (T_{pre}) and, finally, the arithmetic and H.264 low-res. encoding on the CPU (T_{enc}). On the client-side we perform: low-res. H.264 stream and edge decoding including depth/motion frame prediction (T_{dec}), and edge diffusion (T_{dif}) followed by upsampling the low-res. video frame (T_{up}).*

The resulting compression ratio and the quality is analyzed in Table 2. We compare the efficiency of our method

Scene	Bandwidth		H.264		Our method	
	shading $[Mbit/sec]$	depth $[Mbit/sec]$	**A** $[dB]$	**B** $[dB]$	**A** $[dB]$	**B** $[dB]$
SIBENIK	2.0	1.0	35.1	49.6	35.7	49.5
SPONZA	2.0	2.5	34.2	51.2	35.8	54.2
FAIRY	2.0	2.0	30.6	52.5	33.5	64.2

Table 2: *Quality comparison with H.264 codec. For a given bandwidth constraint we compare the quality (PSNR measured on final output images) of our solution in respect to H.264 based depth+motion encoding.* ***A**)* *Spatio-temporal upsampling done on the client compared to ground truth high-res. image.* ***B**)* *Depth buffer reconstruction on the client side.*

for 3D stereo vision and spatio-temporal upsampling for various configurations. Regarding the absolute compression error (PSNR) of depth images, we are not always better than depth encoding with H.264. However, our compression scheme better preserves depth and motion vectors along edges, which is crucial for our applications. Hereby, we avoid visible distortions while in other homogeneous regions our introduced error is less perceptible. This property also translates in an improved frame reconstruction as illustrated in Fig. 1, Table 2 and the accompanying video.

Generally, our method provides a high-quality depth and motion approximation well suited for upsampling and warping frames (see Fig. 1). Our client-side spatio-temporal bilateral upsampling (Sec. 3.3) ensures robust upsampled frames even for a larger depth/motion-residual quantization (e.g., $q = 3$) since the edge topology is encoded lossless and therefore avoids leakage between discontinuous image regions. In our results we encoded depth with 10 bits (or 8 bits when comparing against H.264) and 2D motion using 2 times 8 bits whereas for H.264-encoded depth we were limited to 8 bits only. H.264 offers a special *High* 4 : 4 : 4 *Predictive* mode particularly suited for depth, but featuring lossless encoding makes it impractical in our context.

Our compression exceeds the quality of the state-of-the-art platelets approach [MMS*09] (Fig. 8). For a fair comparison, we only relied on an intra-frame context and did not make use of previous frames for this comparison. In particular, we manage to maintain complex boundaries, which are only linearly approximated for platelets. A disadvantage of our compression is that the bitrate is harder to control than for platelets. Nevertheless, our solution is practical and the adaptive scheme steers the bitrate convincingly to reach an acceptably low deviation (Fig. 7). Such a property is important for online contexts.

Our diffusion solution compares favorably to many existing methods. Simple push-pull solutions fill holes using nearest samples, often involving a Gaussian pyramid. However, averaging across edges leads to leakage which might be unacceptable. For our solution, leakage occurs only where it is least perceivable (even for 3D warping) and has little impact on the upsampling process. PDE-based diffusion [FFLS08, OBW*08] provides good quality and smooth results, but is rather costly and difficult to control the number

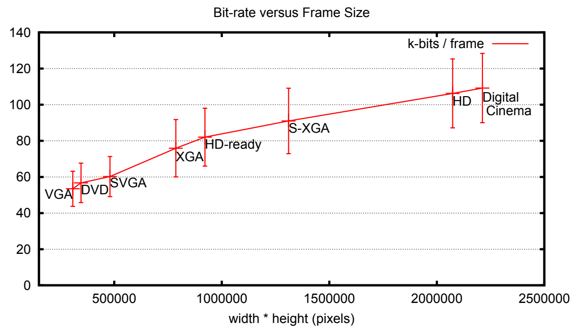

Figure 7: *Plot of mean and standard dev. (over 400 frames) of the bit rate with respect to frame resolution for constant edge detection thresholds of the animated (camera, lighting and objects) Sibenik scene.*

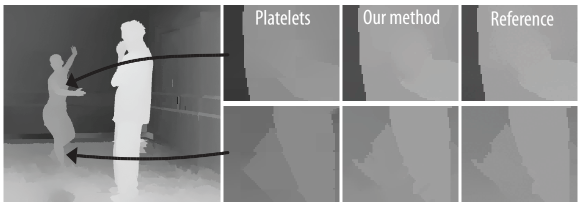

Figure 8: *Depth reconstruction quality: For the same bandwidth (0.1 bits/pixel), our method achieves significantly better reconstruction quality (PSNR of 48.1 dB) than platelet encoding (PSNR of 43.9) and captures smooth regions and discontinuities. (Data courtesy of [MMS*09])*

of iterations until convergence. Further, such a global diffusion is not needed and is disturbed by the uniform edge samples. Recent work by Fattal [Fat09] introduces second generation wavelets for edge-stopping smoothing and diffusion, that could be used in our context. Nonetheless, our approach is simpler and faster, with directly controlled execution time. Furthermore, our method achieves the highest-quality reconstruction when applied to our streamed representation, as it is inherently well suited for the additional sparse uniform sampling. Fig. 4 compares the various approaches.

6. Limitations and Future Work

Our streaming solution inherits limitations from upsampling methods. For volumetric effects, mirror reflections, refractions or multi-sample rendering (e.g. antialiasing, transparency) the computed geometric flow will not correspond to the actual optical flow, thus, forcing temporal reprojection to fail. Deriving correct motion flow and depth in such a scenario is challenging. However, as shown in the accompanying video, our scheme handles these situations partially by automatically falling back to pure spatial upsampling, more advanced detection methods or handling remain future work.

The H.264 video encoder uses the x264 library [x26], which is limited to 4:2:0 chroma subsampling that stores color data downsampled by a factor of 2 in both dimensions. This sometimes leads to undesired color bleeding in colorful scenes in case that the spatial upsampling window is larger than 2. However, we believe that 4:2:2 or even better 4:4:4 chroma subsampling would solve this issue.

Currently, we use a sparse regular sampling to account for low-frequency information. Although this proved sufficient for our purposes, an interesting alternative could be to rely on multi-scale edge detection. Further, our statistical edge encoding could be improved with a global optimization, but then we would lose efficiency.

Lossless edge-topology encoding is difficult to combine with constant bandwidth usage. For some particular scenarios the perceptual edge weighting scheme did not converge to the target rate quickly enough. Faster bandwidth control could be achieved by scaling the resolution and/or framerate dynamically, as in the H.264 SVC standard.

Performance-wise the proposed solution assumes server-side pixel computations to be moderately expensive. In case of simple shaders (Table 1, FAIRY scene) the system is not able to efficiently amortize the cost of additional computations with lower resolution rendering. However, similar to commercial solutions, video compression could be performed in hardware and further code optimizations (vectorization/parallelization) remain possible. In fact, our method could easily benefit from specialized hardware. Similar to H.264, the final (and the most expensive) encoding step employs a standard arithmetic encoder (i.e. CABAC) which makes us believe that a full-quality hardware realization is possible using existing components.

7. Conclusion

We presented an efficient solution for 3D-rendered-content streaming. Our method achieves similar compression as high-end encoders. In contrast to competition, our approach provides the client with additional information, such as depth, at no additional transmission cost. Our contribution is also the careful selection of such data: inexpensive to compute on the GPU server (much cheaper than full-resolution frames), but easy-to-compress without dependence on future frames. Moreover, the full resolution depth maps enable 3D stereo viewing and the motion-flow maps can be used to boost the frame rate (important to reduce hold-type blur for LCD displays) or recover corrupted frames. Another interesting application is personalized rendering for advertisement or enhancement purposes. Our scheme is naturally compatible with existing game engines. It relies on specialized rendering procedures, but the required changes are minimal and easy to realize in game engines (for games, deferred shading is common and directly provides the needed data). This feature makes our approach attractive for many streaming contexts. Moreover, we are not limited to rasterization.

Our framework can be beneficial for raytracing, where remote rendering is even more appropriate since computations require high-performance computers, or even cluster.

References

[CBPZ04] CHENG L., BHUSHAN A., PAJAROLA R., ZARKI M. E.: Real-time 3D graphics streaming using MPEG-4. In *Proc. IEEE/ACM Workshop on Broadb. Wireless Serv. and Appl.* (2004).

[COMF99] COHEN-OR D., MANN Y., FLEISHMAN S.: Deep compression for streaming texture intensive animations. In *Proceedings of SIGGRAPH* (1999), pp. 261–268.

[DER*10] DIDYK P., EISEMANN E., RITSCHEL T., MYSZKOWSKI K., SEIDEL H.-P.: Perceptually-motivated real-time temporal upsampling of 3D content for high-refresh-rate displays. *Comp. Graph. Forum (Proc. of Eurographics) 29*, 2 (2010), 713–722.

[DRE*10] DIDYK P., RITSCHEL T., EISEMANN E., MYSZKOWSKI K., SEIDEL H.-P.: Adaptive image-space stereo view synthesis. In *Proc. of VMV* (2010).

[Eld99] ELDER J. H.: Are edges incomplete? *International Journal of Comp. Vision 34*, 2-3 (aug 1999), 97–122.

[EWG99] EISERT P., WIEGAND T., GIROD B.: Rate-distortion-efficient video compression using a 3D head model. In *Intern. Conf. on Image Proc.* (1999), pp. 217–221.

[Fat09] FATTAL R.: Edge-avoiding wavelets and their applications. *ACM Trans. on Graph. 28*, 3 (July 2009), 22:1–22:10.

[FE09] FECHTELER P., EISERT P.: Depth map enhanced macroblock partitioning for H.264 video coding of computer graphics content. In *Intern. Conf. on Image Proc.* (2009), pp. 3441–3444.

[Feh04] FEHN C.: Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV. In *Stereoscopic Displays and Virtual Reality Systems XI* (2004), vol. 5291, SPIE, pp. 93–104.

[FFLS08] FARBMAN Z., FATTAL R., LISCHINSKI D., SZELISKI R.: Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Trans. on Graph. 27*, 3 (2008), 67:1–67:10.

[GGSC96] GORTLER S. J., GRZESZCZUK R., SZELISKI R., COHEN M. F.: The lumigraph. In *Proc. of SIGGRAPH* (1996), ACM, pp. 43–54.

[GWW*08] GALIC I., WEICKERT J., WELK M., BRUHN A., BELYAEV A. G., SEIDEL H.-P.: Image compression with anisotropic diffusion. *Journal of Math. Imaging and Vision 31*, 2-3 (2008), 255–269.

[HEMS10] HERZOG R., EISEMANN E., MYSZKOWSKI K., SEIDEL H.-P.: Spatio-temporal upsampling on the GPU. In *Proc. of I3D* (2010), ACM, pp. 91–98.

[HKMS08] HERZOG R., KINUWAKI S., MYSZKOWSKI K., SEIDEL H.-P.: Render2MPEG: A perception-based framework towards integrating rendering and video compression. *Comp. Graph. Forum (Proc. of Eurographics) 27*, 2 (2008), 183–192.

[ITU99] ITU-T: Information technology – coded representation of picture and audio information – lossy/lossless coding of bi-level images, itu-t recommendation t.82 – iso/iec international standard 14492 final committee draft.

[JWB04] JIANG Z., WONG T.-T., BAO H.: Depth image sequences compression using programmable graphics hardware. TR-2004-01, 2004.

[KAF*07] KAUFF P., ATZPADIN N., FEHN C., MÜLLER M., SCHREER O., SMOLIC A., TANGER R.: Depth map creation and image-based rendering for advanced 3dtv services providing interoperability and scalability. *Signal Processing: Image Communication 22*, 2 (2007), 217 – 234. Special issue on 3D video and television.

[KbCTS01] KRISHNAMURTHY R., BING CHAI B., TAO H., SETHURAMAN S.: Compression and transmission of depth maps for image-based rendering. In *Intern. Conf. on Image Proc.* (2001), pp. 828–831.

[LCD06] LUFT T., COLDITZ C., DEUSSEN O.: Image enhancement by unsharp masking the depth buffer. *ACM Trans. on Graph. (Proc. of SIGGRAPH) 25*, 3 (2006), 1206–1213.

[Lev95] LEVOY M.: Polygon-assisted JPEG and MPEG compression of synthetic images. In *Proceedings of ACM SIGGRAPH* (1995), pp. 21–28.

[MD08] MAITRE M., DO M.: Joint encoding of the depth image based representation using shape-adaptive wavelets. In *Intern. Conf. on Image Proc.* (2008), pp. 1768–1771.

[MF98] MURAD A., FUJA T.: Exploiting the residual redundancy in motion estimation vectors to improve the quality of compressed video transmitted over noisy channels. In *Intern. Conf. on Image Proc.* (1998), pp. 497–501.

[MMS*09] MERKLE P., MORVAN Y., SMOLIC A., FARIN D., MÜLLER K., DE WITH P. H. N., WIEGAND T.: The effects of multiview depth video compression on multiview rendering. *Signal Processing: Image Communcation 24*, 1-2 (2009), 73–88.

[NDS*08] NAVE I., DAVID H., SHANI A., LAIKARI A., EISERT P., FECHTELER P.: Games@Large graphics streaming architecture. In *Symp. on Consumer Electronics (ISCE)* (2008).

[OBW*08] ORZAN A., BOUSSEAU A., WINNEMÖLLER H., BARLA P., THOLLOT J., SALESIN D.: Diffusion curves: A vector representation for smooth-shaded images. In *ACM Trans. on Graph. (Proc. of SIGGRAPH)* (2008), vol. 27, pp. 92:1–92:8.

[Pae91] PAETH A. W.: Image file compression made easy. In *Graphic GEMS II* (1991).

[PJO*09] PARK Y. K., JUNG K., OH Y., LEE S., KIM J. K., LEE G., LEE H., YUN K., HUR N., KIM J.: Depth-image-based rendering for 3DTV service over T-DMB. *Signal Processing: Image Communication 24*, 1-2 (2009), 122–136.

[SaLY*08] SITTHI-AMORN P., LAWRENCE J., YANG L., SANDER P. V., NEHAB D.: An improved shading cache for modern GPUs. In *Proc. of Graphics Hardware* (2008), pp. 95–101.

[SLW*08] STICH T., LINZ C., WALLRAVEN C., CUNNINGHAM D., MAGNOR M.: Perception-motivated Interpolation of Image Sequences. In *Proc. APGV* (2008), pp. 97–106.

[WKC94] WALLACH D. S., KUNAPALLI S., COHEN M. F.: Accelerated MPEG compression of dynamic polygonal scenes. In *Proceedings of SIGGRAPH* (1994), pp. 193–197.

[WNC87] WITTEN I. H., NEAL R. M., CLEARY J. G.: Arithmetic coding for data compression. *Commun. ACM 30*, 6 (1987), 520–540.

[x26] Free software library and application for H.264 video coding. http://www.videolan.org/developers/x264.html.

[YSL08] YANG L., SANDER P. V., LAWRENCE J.: Geometry-aware framebuffer level of detail. *Comp. Graph. Forum (Proc. of EGSR) 27*, 4 (2008), 1183–1188.

[ZDL00] ZENG W., DALY S., LEI S.: Visual optimization tools in JPEG 2000. In *IEEE Intern. Conf. on Image Proc.* (2000).