Augmented Compression for Server-Side Rendering

Fabian Giesen, Ruwen Schnabel and Reinhard Klein

University of Bonn
Computer Graphics Group
Email: {giesen,schnabel,rk}@cs.uni-bonn.de

Abstract

In this work we recall attention to problems that arise in a client-server setting with server-side rendering and propose a practical method for accelerated high-quality render-stream compression on the server. Server-side rendering is gaining importance for three main reasons: On the one hand great differences in quality and performance of client's systems (ranging from PDAs to high-end workstations) complicate application development, on the other hand 3D content providers refrain from transmitting costly 3D data to clients and last but not least no adequate and widely accepted standardized 3D interchange format exists. A major challenge is the high server workload per client. To address one factor of the server load, we describe an augmented compression of server-side renderings that produces standard video streams but exploits the additional information available through image warping for motion estimation.

1 Introduction

Over the course of the last decade, 3D hardware has become cheap and commonplace; an average new PC has better graphics capabilities than dedicated workstations had 10 years ago. As a result, applications that make extensive use of 3D graphics are becoming increasingly widespread and popular.It also means that standards have risen considerably: even relatively cheap hardware is able to render scenes with millions of visible triangles and hundreds of megabytes of texture data at interactive rates. At the same time, 3D display of one sort or another has appeared even in relatively weak embedded and mobile devices; an example are car navigation systems, which by now typically show a (relatively crude) 3D rendition of the area surrounding the car.

But while 3D rendering is starting to become a commodity, there is a huge variation in the available levels of performance and quality. On PCs, high-end graphics cards not only provide a far larger featureset than integrated graphics chips, they are also between 1 or 2 *orders of magnitude* faster. For embedded and mobile devices, the differences are even bigger, ranging from graphics chips that only provide a framebuffer with no hardware-accelerated rendering at all through hardware support for 2D vector graphics to fully-fledged 3D chipsets roughly on par with high-end PC rendering hardware around 2001.

This creates a big problem for application developers: the only way to achieve consistent quality and performance over a wide range of different target machines is to either have separately tuned datasets and renderers for different configurations, which is very expensive to develop, or to aim for the lowest common denominator, which means that the added capabilities of newer hardware don't get used at all.

Another problem for client-side rendering is the need to distribute the actual 3D content to clients. Especially if the acquisition or creation of content is a costly process or the content contains vital business or technical information, owners agree, if at all, only very reluctantly to its distribution. For instance, with systems such as *Google Earth*, the potential userbase is everyone with access to the Internet. This is a problem for providers of GIS (geographical information system) datasets: this data is quite costly to obtain, and making it available to virtually everyone free of charge is not always in their best interest.

Finally, just as development of 3D hardware does not stand still, neither does rendering and geometry/material scanning technology, with the result that data formats go in and out of fashion every few years, always being replaced by a representation that is more suitable for the current state of the art. All attempts at interchange formats are at best limited and at worst obsolete by the time their specification is finished. This is a big problem when developing an application that is supposed to have a lifetime of at least a few years, since it complicates the choice of delivery format considerably or necessitates proprietary data formats.

In a networked environment, an obvious approach is to try and offload some of the work to the server. Martin [13] describes three primary ways of performing 3D rendering in a client-server environment:

Client-side methods, where the server only supplies a description of the 3D scene (model/geometry data, textures etc.) to the clients, which have to perform the rendering themselves. This is the approach taken by the majority of current networked 3D applications. It is relatively simple on the server side and a single server can easily process a large number of clients. However, as explained above, rendering on the client causes significant problems for implementors and content providers.

Server-side methods let the server perform all the rendering. The finished image is then transmitted to the client, usually in compressed form. Of course, the maximum number of concurrent clients per server is usually several orders of magnitude lower than for client-side methods but on the plus side compatibility issues on the client mostly disappear, and the same quality is available to all clients no matter how powerful their hardware is. Also, it is easy to switch scene representations or renderers: all it takes is installing a new application on the server(s).

Hybrid-side methods perform rendering on both the server and client sides and combine the images on the client. The problem is that such schemes inherit weaknesses from both client- and server-based methods: rendering overhead is incurred on both sides, but scene data still needs to be transmitted to clients, and compatibility problems may still arise.

In this work we want to draw attention to server-side rendering that, while not a panacea, certainly has serious practical advantages for a wide range of applications. There are two main issues that need to be considered for a practical server-side rendering system:

1. Each server needs to render frames for several clients, an expensive operation. If possible, one would like to reduce the total workload by reducing the amount of per-client work by exploiting temporal coherency between the frames produced for a single client and similarities between the viewpoints of different clients.
2. After rendering, the server also needs to compress the outgoing data for each of the clients. Again, this is quite expensive; however, since the images are rendered by the server, there is quite precise information about what changed from one frame to the next; current video coders do not make use of such additional information, which might incur a cost in both bitrate, and thus network bandwidth, as well as performance.

Here we will only consider the compression aspects of server-side rendering and our main contribution is as follows: We describe a practical method that is able to improve compression ratio and speed for current video standards: warping is used to estimate motion vectors for conventional motion compensation-based video codecs, the advantage of such codecs being that they are standardized already, and that decoder software and hardware is available virtually everywhere.

2 Previous work

McMillan's dissertation [14] laid the necessary groundwork in image warping for various camera models. Marks PhD thesis [12]describes techniques more suitable in a real-time environment. The general idea is to use warping to speed up a conventional (rasterization or ray tracing-based) renderer by only generating a few images per second and using warping to interpolate between them.

2.1 Client-server rendering

We are not the first to suggest using warping-based methods in a client-server rendering environment. Warping a reference depth image to a new viewpoint typically produces a quite dense image, apart from holes that appear due to occlusion or exposure errors. This can be exploited on the server side by only rendering those areas where no information is available, and for compression by only transmitting new data pixels where such errors occur. In [18], Yoon and Neumann describe their IBRAC system (Image-Based Rendering Acceleration and

Compression) which is based on this basic idea.

Since the algorithm is purely warping-based and has no means of replacing colors or depth values for pixels that are deemed correct after warping, surfaces that are not perfectly diffuse obviously pose a problem. Even worse are invisible occluder errors, since there is by definition no information contained in the reference image that suggests their presence.

Less ambitious and also less problematic is a client-server system briefly described in chapter 6 of Marks aforementioned PhD thesis [12]. The client transfers the current camera position to the server, which periodically renders a new view and transmits it back to the client. Warping is used for two purposes: to compensate for network latency, and to have the client render at a higher frame rate than server-side rendering speed and network bandwidth allows. However, the system is again purely warping-based and thus subject to occlusion and exposure errors. No compression is used for the communication between client and server.

Hudson and Mark [8] update this system to use 3 reference image sets with different centers of projection. Together with an algorithm to select "good" reference viewpoints, this notably reduces occlusion and exposure errors in typical walkthrough scenarios. The lack of compression makes the system impractical for use over the Internet, though.

Chang and Ger [6] describe a similar system with significantly lower CPU usage, using PDAs as clients. They only use single planar reference images (which makes the system susceptible to invisible occluder errors) but do support layered depth images [15] to reduce occlusion errors.

Another client-server system is described by Thomas et al. [16]. Aiming at urban walkthroughs, they try to optimize the placement of reference cameras so that a complete scene can be described exclusively using a relatively small number of reference images. The server doesn't need to constantly render new camera views; rather, new views are only generated when the user moves in an area not well covered by the currently active reference views. This reduces network bandwidth requirements significantly; reference views are additionally compressed using `zlib`. However, to make use of the multiple reference images, the clients need to perform several warping operations per frame (typically 2 to 4). Moreover, while the camera selection algorithm makes it unlikely that any significant errors are introduced during the warping process, it does so only after severely constraining the scene geometry and viewer.

2.2 Compression

Apart from IBRAC, the systems mentioned in the previous section are all designed to work over a local network with relatively high transfer rates and low latencies; as a result, they do not invest much effort on compression, since it would have little to no practical benefit. For usage over the Internet, available bandwidth from the server to the client is more constrained (and a cost factor!), which makes compression inevitable.

Aliaga et al. [2] present a method designed for architectural walkthroughs of real-world buildings, where floor plans (and thus a coarse description of the underlying geometry) are available and photographs are fairly easy to obtain; warping is then used mainly as prediction between similar images, to reduce the amount of data that has to be stored. However, the algorithm is not applicable to interactive applications, where the images to be compressed are not known beforehand.

A coding algorithm for *depth* images is presented by Duan and Li [7], who discuss compression of layered depth images. They reported that compression of LDIs is as high as $17:1$ with minimal visual distortion. However, in practice a bad choice of reconstruction filters in the renderer (or optimizations that sacrifice visual quality for speed) can amplify small errors significantly.

2.3 Video coding

Most state of the art video codecs [9][10][3] are quite similar in their basic structure and support two basic modes: intra and inter coding. Intra coded frames are self-contained (like an image file), while inter frames use data from other (previously coded) frames to exploit temporal coherency between frames. The intra part of any video codec is just a still image coder; inter coding and the additional redundancies it can exploit are the reason why current video formats are able to achieve significantly lower bitrate at the same quality, compared to coding each frame individually.

All these codecs perform the inter prediction step using different variants of *motion compensation*: the destination image is partitioned into rectangular blocks of pixels. For each block, a single 2D vector (m_x, m_y) is stored, which is an offset relative to the block's position that specifies where the source data for the respective block is located in the reference frame. Conventional (block) motion compensation just copies the respective pixels over, and is used by MPEG-4 variants.

The problem of finding these motion vectors on the encoder side is called *motion estimation*. Formally, it is an instance of the optical flow problem in computer vision [11][4][5], but there are no regularity constraints on the motion vectors, and the objective function is coding cost instead of a direct image similarity metric. As a result, codec implementations typically don't use direct optical flow algorithms; specialized methods, usually based on explicit search with early termination heuristics, are more common. This motion estimation process is quite costly (even with optimized algorithms), and typically a significant fraction of the total video encoding time is spent on it.

The image obtained from the motion compensation process is an approximation of the target frame; this approximation is subtracted from the actual image data, yielding the *residual image*. The residual image is decorrelated using a wavelet transform or similar techniques. The transform yields *transform coefficients* which are then quantized, the main lossy step in video coding. Since they are the result of a decorrelation process, those coefficients can be quantized individually; this is done using uniform quantizers with or without dead zone.

3 Augmented compression

Network bandwidth requirements for a server-side rendering application are necessarily relatively high: even with modest resolutions (e.g. 320×240 pixels) and low frame rates (20 frames/second), video data needs over 200 kbit/second to display high-motion scenes with acceptable visual quality (and virtually any camera movement results in a lot of motion, because moving the camera changes the whole frame). Furthermore, state of the art codecs have to be used, because older ones perform significantly worse (in terms of resulting distortion) for low-bitrate applications.

Also, rendering several views per frame on the server side (one for every client) is expensive in terms of CPU/GPU time; having to additionally encode several compressed streams at the same time adds to the computational load, and video encoding has high cost all by itself already.

Here we propose an augmented compression algorithm that is designed to reduce the server load. The key idea is to use the additional information available from the rendering process to improve the time-consuming motion estimation stage of compression. The additional information available on the server is comprised of the external and internal camera parameters as well as per-pixel depth values. While it might be promising to use image warping as a replacement for the conventional motion compensation in such a setting, this is currently not supported by video standards or decoding hardware, although we do expect that to change in a number of years. Thus, for the time being, we restrict ourselves to available standards and show that even in such a constrained environment improvements can be gained from the depth information. This way our algorithm can make use of established conventional video codecs, which are well-tuned and have widespread hardware and software support; in particular, even some otherwise very weak devices such as MP3 players can play back these videos because they have hardware decoders.

3.1 Warping-based motion estimation

Our main idea is to improve video compression quality and, possibly, speed by using warping to determine motion vectors, instead of performing conventional motion estimation. The video format used for comparison is H.264 [10], because it is both an ISO and ITU standard and quite popular in applications.

H.264 allows having more than one motion vector per macroblock: the 16×16 pixel blocks can be subdivided further, down to sixteen blocks of 4×4 pixels each if necessary. The resulting subblocks are called *partitions*. The optimal partitioning for each macroblock is determined during encoding. Instead of performing the default motion search procedure, the midpoint of each partition is warped to obtain the corresponding point in the reference image.[1] The difference between the position of the midpoint in the destination image and the po-

[1]There might not be such a point, if the 3D position of the destination point is in front of the reference camera's near plane; in that case, the fallback solution is to perform the regular motion search procedure on that partition.

sition in the reference image is used as the motion vector; since warping typically results in fractional coordinates anyway, this motion vector can be determined with subpixel precision.

The per-partition cost of this procedure is quite low: one evaluation of the warping equation (11 multiplies, 9 additions, 1 division), two floating point to integer conversions, and two integer subtractions to turn the reference frame coordinates into motion vectors relative to the destination block midpoint.

Using the obtained motion vectors directly is possible, but it still makes sense to give the encoder a little more freedom: using a motion vector that is off by some subpixels may reduce blurring, use less bits with no significant quality difference, or be otherwise beneficial. Thus it still makes sense to try a few motion vectors in the direct neighborhood of the calculated motion vector, and pick the best one.

To do this efficiently, a very simple trick is used: care is taken to make sure that rounding errors made when converting the fractional motion vectors to integers are one-sided. This is done by always rounding downwards. The obtained motion vector (m_x, m_y) will thus tend to be slightly too small in both components. The code then tries all four members of the set $\{(m_x+i, m_y+j) \mid i,j \in \{0,1\}\}$ as candidate motion vectors, and the best one is used. This requires less than half as many tests as the more obvious procedure of rounding motion vectors towards the closest integer and then trying candidate motion vectors of the form $\{(m_x+i, m_y+j) \mid i,j \in \{-1,0,1\}\}$, but results in very similar quality.

It is still possible to perform regular subpixel refinement (like one would do with motion vectors obtained from a direct search procedure) on the resulting motion vectors; just as with normal motion estimation, this increases quality by allowing the codec to make a better rate/distortion tradeoff, at the expense of increased runtime. But even without subpixel refinement, the obtained motion vectors have good quality.

3.2 Implementation

We implemented the method as described above by modifying `x264` [1], an open-source H.264 codec library that also performs quite well in H.264 encoder comparisons.

Warping has two main components: calculation of the warping matrix and evaluation of the warping equation. The computation of the warping matrix only has to be performed once per frame, or more precisely, once for every pair of reference and destination frame.

The warping equation is evaluated to determine the motion vector for a given partition (which is described by the x, y coordinates of its top-left pixel and its width and height) by warping the midpoint, as described above.If the warping process yields a motion vector, its cost and that of other "close" motion vectors are computed. The cost function is a weighted sum of the number of bits required to encode a motion vector and a vector norm of the difference between the reference block and the block to be coded. H.264 predicts the motion vector for each partition from the motion vectors in adjacent partitions that have already been coded; this predicted motion vector and the null motion vector are cheaper to encode than regular motion vectors, so their cost is also evaluated. The overall best motion vector is used.

If the warping process was not successful, normal motion search is performed as a fallback solution; this only occurs very rarely in practice however. Finally, additional subpixel refinement of the obtained motion vectors can be performed, if desired. This improves quality but comes at an extra expense in CPU time.

4 Results

The modified version of `x264` was tested on several different sequences (see Fig. 1 and accompanying videos) and with different encoder parameters (target bitrate, number of reference frames per target frame, and accuracy of subpixel motion estimation) to evaluate the efficiency of the proposed warping-based motion estimation process.

The test procedure is as follows: First, the input sequence is read once in full, to make sure it is in the filesystem cache so that I/O bandwidth does not affect the results. Then, the encoder application is run with the specified settings, encoding from the input file. After completion, several statistics about the encoded sequence are output, including objective quality metrics and the speed of encoding as measured in encoded frames per second. For performance measurements the encoder writes to the

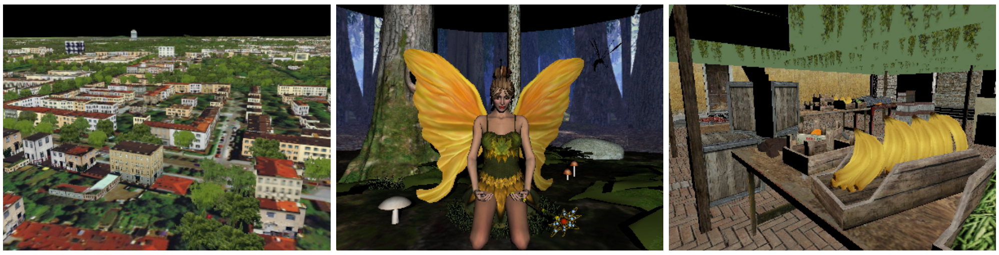
(a) Terrain (b) Fairy (c) Italy

Figure 1: Screenshots taken from the three different test sequences used in the evaluation in sec. 4

null device and, in order to account for random variations due to background processes, each run is repeated three times, and the median speed is used. The quality of the encoded sequence is measured both using the PSNR of the luminance channel and the structural similarity index (SSIM) between the luminance images. The latter is introduced in [17] and tries to take perceptual effects of the human visual system into account to produce an objective quality metric that has a stronger correlation to perceived similarity than the mean square error (MSE) and derived metrics such as the PSNR do. SSIM indices range between 0 and 1, where 0 would indicate that the two images are completely uncorrelated, while 1 means that they are identical.

All tests are run on a notebook with an Intel Core2Duo T7500 processor (clocked at 2.2 GHz) and 3 GB of RAM. The encoder always makes use of the dual-core processor and all partition types. We disabled B-frames, since using them actually decreased overall quality in these tests.[2]

"Terrain" sequence The first test sequence, "terrain", is a flight over a (completely diffuse) 3D dataset of Munich. The geometry for individual houses is quite simple (mostly extruded 2D paths), but since there is often a large number of houses visible at the same time, the overall amount of geometric detail visible in a typical frame is relatively high. The camera motion is a mixture of user interaction and computer-generated smooth flights between different points of interest.

Results are shown in table 1. "Bitrate" is the target bitrate passed to the encoder, "Ref" is the number of reference frames to use (higher numbers improve quality but cost extra CPU time), "WarpME" indicates whether warping-based motion estimation was used or not and "SubME level" selects the quality of subpixel motion estimation: 1 disables it altogether, 2 performs a few subpixel refinement iterations, and successive levels add a higher number (and better accuracy) of refinement steps, up to 5 which is the default. Levels 6 and 7 perform full rate-distortion optimization instead of minimizing the heuristic cost function; this improves quality but significantly increases CPU usage and is probably impractical for real-time encoding. The columns "SSIM-Y", "PSNR-Y" and "Frames/s" report the results obtained with the given parameter set.

A first surprise is that turning on warping-based motion estimation, all other parameters being equal, does not improve speed. Further experimentation revealed that the difference is caused by evaluating the warping equation: This cost is nearly constant and paid for every partition, whereas the normal motion estimation search patterns use predicted motion vectors and early-outs to minimize average-case runtime. The actual time spent computing the cost for candidate motion vectors is very similar in both cases, but the warping-based method has higher overhead because the warping equation needs to be evaluated.

However, warping-based motion estimation *does* produce a notable improvement in PSNR in *all* tests, and in SSIM for all but the 300 kbit/s tests. In fact, for all tests, enabling WarpME yields better results (in terms of PSNR) than those produced without warping and using the *next higher* listed level of subpixel refinement. SSIM results are not quite as spectacular, but enabling warping still produces significant improvements with bitrates of 500 and 1000 kbit/s: while not surpassing the SSIM indices

[2]They generally improve quality when two-pass encoding can be used.

Settings				Results		
Bitrate (kbit/s)	Ref	WarpME	SubME level	SSIM-Y	PSNR-Y (dB)	Frames/s
300	1	no	1	0.7949808	26.140	133.34
300	1	yes	1	0.7918640	26.480	131.01
300	1	no	2	0.8017598	26.307	112.31
300	1	yes	2	0.7955168	26.589	111.13
300	1	no	5	0.8116736	26.586	78.73
300	1	yes	5	0.8098858	26.877	79.33
300	3	no	1	0.7926901	26.103	125.71
300	3	yes	1	0.7921847	26.485	119.71
300	3	no	2	0.8003088	26.308	103.90
300	3	yes	2	0.7942310	26.597	100.91
300	3	no	5	0.8111303	26.592	69.39
300	3	yes	5	0.8099403	26.898	68.85
500	1	no	1	0.8570027	27.950	126.00
500	1	yes	1	0.8629337	28.630	123.35
500	1	no	2	0.8638736	28.196	104.71
500	1	yes	2	0.8671046	28.783	101.88
500	1	no	5	0.8708137	28.466	70.60
500	1	yes	5	0.8728295	28.982	70.13
500	3	no	1	0.8553014	27.914	119.17
500	3	yes	1	0.8630835	28.655	111.13
500	3	no	2	0.8636627	28.218	95.59
500	3	yes	2	0.8671956	28.816	90.03
500	3	no	5	0.8714704	28.512	62.85
500	3	yes	5	0.8736568	29.044	61.24
1000	1	no	1	0.9221869	30.991	115.77
1000	1	yes	1	0.9269663	31.948	109.73
1000	1	no	2	0.9272521	31.292	92.73
1000	1	yes	2	0.9300071	32.137	88.09
1000	1	no	5	0.9308854	31.571	61.11
1000	1	yes	5	0.9323950	32.299	60.47
1000	3	no	1	0.9213025	30.956	108.83
1000	3	yes	1	0.9277248	32.011	100.33
1000	3	no	2	0.9277198	31.341	86.78
1000	3	yes	2	0.9310868	32.219	80.54
1000	3	no	5	0.9318547	31.658	55.56
1000	3	yes	5	0.9334796	32.395	54.70

Table 1: Encoding results for the “terrain” sequence.

obtained using the “next higher level” of subpixel refinement, they are nevertheless quite close.

These results are very consistent over the quite large range of different parameters given, indicating that warping results in either notably improved quality for very little extra CPU time, or matches a given target quality with substantially lower CPU cost.

“Fairy” sequence The second test sequence, “fairy” is a camera flight through the Utah Fairy Forest scene (a standard test scene for real-time raytracing, available at `http://www.sci.utah.edu/~wald/animrep/`) and shows a model of a fairy in front of a forest backdrop (which includes modeled mushrooms, grass, and trees). There are large variations in the size of geometric features: for example, individual grass blades are represented as geometry. The camera plays back a motion along a spline that was created in a 3D modeling application.

Corresponding results are shown in table 2. Only the results when using one reference frame are reported in the following; in all of the sequences, and both with and without warping, using multiple reference frames results in better quality at the cost of slightly higher encoding time, and the behavior of warping-based motion estimation was very similar between one-reference-frame and multiple-reference-frame tests.

Settings				Results		
Bitrate (kbit/s)	Ref	WarpME	SubME level	SSIM-Y	PSNR-Y (dB)	Frames/s
300	1	no	1	0.8544471	31.999	146.94
300	1	yes	1	0.8543651	31.967	147.32
300	1	no	2	0.8593798	32.159	126.32
300	1	yes	2	0.8589095	32.119	127.15
300	1	no	5	0.8695813	32.459	89.44
300	1	yes	5	0.8686421	32.387	90.57
500	1	no	1	0.8924078	33.747	136.82
500	1	yes	1	0.8927671	33.723	135.22
500	1	no	2	0.8974561	33.973	114.52
500	1	yes	2	0.8974329	33.935	114.97
500	1	no	5	0.9040100	34.256	79.45
500	1	yes	5	0.9029872	34.184	80.11
1000	1	no	1	0.9377240	36.744	120.76
1000	1	yes	1	0.9378214	36.714	120.50
1000	1	no	2	0.9419348	37.056	98.47
1000	1	yes	2	0.9414331	36.991	97.96
1000	1	no	5	0.9457670	37.369	66.44
1000	1	yes	5	0.9449789	37.291	66.36

Table 2: Encoding results for the “fairy” sequence.

The numbers themselves are quite different than those obtained using the “terrain” sequence. Here, warping is actually slightly faster in a large number of cases, but produces slightly worse results in general, though the difference is small: always lower than 0.075 dB for the PSNR ratings—contrast with the consistent improvement of over 0.25 dB for *all* tests run on the terrain dataset, with warping producing a gain exceeding 0.9dB several times for the higher bitrates. The SSIM indices are, similarly, quite close. So while the warping-based motion estimation yields no improvement for this scene, it does not make the results significantly worse, either. In general, the camera motions in this test sequence are quite smooth and slow, which benefits conventional motion estimation, since a search procedure is likely to find good motion vectors quickly.

“Interactive fairy” sequence To test whether this indeed makes a difference, the same scene was rendered using a different camera motion, this time recorded from an interactive session. As a result, motions are jerkier in general, and include short burst of very high-motion frames whenever the viewer “looks around”, turning the camera rapidly

Settings				Results		
Bitrate (kbit/s)	Ref	WarpME	SubME level	SSIM-Y	PSNR-Y (dB)	Frames/s
300	1	no	1	0.8940561	34.118	144.35
300	1	yes	1	0.8976926	34.656	141.18
300	1	no	2	0.8983479	34.358	127.15
300	1	yes	2	0.9011825	34.860	123.34
300	1	no	5	0.9061646	34.734	90.42
300	1	yes	5	0.9133149	35.303	89.16
500	1	no	1	0.9310123	36.642	133.95
500	1	yes	1	0.9349554	37.420	130.61
500	1	no	2	0.9352065	36.969	115.65
500	1	yes	2	0.9385026	37.715	112.50
500	1	no	5	0.9401657	37.339	80.00
500	1	yes	5	0.9448482	38.109	79.23
1000	1	no	1	0.9685249	40.839	119.25
1000	1	yes	1	0.9703983	41.906	115.21
1000	1	no	2	0.9713022	41.251	101.78
1000	1	yes	2	0.9723293	42.198	96.32
1000	1	no	5	0.9731321	41.572	68.17
1000	1	yes	5	0.9744583	42.550	67.68

Table 3: Encoding results for the "interactive fairy" sequence.

Settings				Results		
Bitrate (kbit/s)	Ref	WarpME	SubME level	SSIM-Y	PSNR-Y (dB)	Frames/s
300	1	no	1	0.6353905	25.557	131.20
300	1	yes	1	0.6679018	26.709	129.44
300	1	no	2	0.6397960	25.668	113.39
300	1	yes	2	0.6722945	26.828	112.71
300	1	no	5	0.6493255	25.838	81.71
300	1	yes	5	0.6833704	27.024	81.24
500	1	no	1	0.7036136	26.687	120.24
500	1	yes	1	0.7314405	27.981	119.74
500	1	no	2	0.7087490	26.836	102.31
500	1	yes	2	0.7372080	28.129	100.52
500	1	no	5	0.7161530	26.995	72.09
500	1	yes	5	0.7444381	28.296	72.18
1000	1	no	1	0.7982175	28.812	106.67
1000	1	yes	1	0.8173899	30.186	105.68
1000	1	no	2	0.8038153	28.988	88.75
1000	1	yes	2	0.8229522	30.386	87.41
1000	1	no	5	0.8115458	29.204	60.63
1000	1	yes	5	0.8285797	30.566	60.00

Table 4: Encoding results for the "cs_italy" sequence.

in the process. Rendering a video with the new camera path resulted in the "interactive fairy" sequence. Results are shown in table 3.

Here, observations are similar to what was already described for the "terrain" sequence: warping-based motion estimation delivers a notable gain in both PSNR and SSIM for all tests, requiring very modest amounts of extra CPU time to do so—far less than the cost of better motion estimation methods. This seems to confirm the conjecture that warping improves on conventional motion estimation mainly by determining fast motions accurately; for slow motions (in the single-pixel or subpixel range), a search-based procedure has a good chance to find local rate-distortion minima, while warping always results in a motion vector close to the "correct" one, which may not be optimal in rate-distortion terms.

"CS_Italy" sequence To confirm this theory, another interactive test sequence is tested. This fourth and last test sequence, "cs_italy", uses the map of the same name from the game Counter-Strike as its 3D scene. Due to limitations of the raytracer used, the included diffuse light maps were not used—resulting in lower visual quality, but not making a substantial difference for video coding, since lighting is still completely diffuse and the lightmaps only contribute quite low-frequency information. The scene has some amount of variation in geometric scale—including houses, a marketplace, with the merchandize of individual stands represented as 3D geometry—but less so than the "fairy" scene does. The camera path was obtained from an interactive session by taking a 30-second long segment from the middle.

Results for this scene are shown in table 4. Here, warping significantly improves quality in all cases, both as measured by PSNR (with an increase exceeding 1 dB in all cases) and the structural similarity index. In particular, the results obtained using warping without subpixel refinement are significantly better than those obtained without warping and level 5 subpixel refinement, even though the former runs faster by a factor of 1.68 on average.

A similar comparison is done for the interactive sequences in table 5. It shows the SSIM index and speed obtained using warping without subpixel refinement; this is compared with the level of subpixel refinement that achieves the closest match in SSIM when warping-based motion estimation is disabled. "Speedup" is the ratio between the two encoding frame rates. As can be clearly seen, warping provides a notable speedup for the interactive scenes in all but one case.[3] The bitrate used was appended to the sequence names.

5 Conclusion

The results reported in this work show that the modified `x264` encoder using warping-based motion es-

[3]The increase in PSNR due to warping is way more pronounced than the increase in SSIM; a comparison based on PSNR values yields far higher speedup values, but since SSIM has higher correlation with perceived image quality, it was used instead.

		Without warping		Warping (SubME=1)		
Sequence	SubME	SSIM-Y	Frames/s	SSIM-Y	Frames/s	Speedup
Terrain–300	1	0.7949808	133.34	0.7918640	131.01	0.9825
Terrain–500	2	0.8638736	112.31	0.8629337	123.35	1.0983
Int. Fairy–300	2	0.8983479	127.15	0.8976926	141.18	1.1103
Int. Fairy–500	2	0.9352065	115.65	0.9349554	130.61	1.1294
CS_Italy–300	6	0.6541142	67.84	0.6679018	129.44	1.9080
CS_Italy–500	6	0.7191762	58.36	0.7314405	119.74	2.0517

Table 5: Time spent to reach a given SSIM index with and without warping.

timation provides significantly increased quality for sequences interactively rendered from user input, which is the typical use case in a client-server setting using server-side rendering. The performance is still comparable to that of conventional motion estimation for scenes with a relatively low amount of motion. The increased quality achieved by our method is attractive to the user, but the main advantage in practice is probably that the same level of quality can be delivered to the user at lower bit rates. This notable increase in quality or, equivalently, compression performance is achieved at very low cost in terms of CPU time, and can be accomplished using comparatively simple modifications to the video encoder. Our results suggest that submitting "hint" motion vectors to a video encoder could be a promising approach in general, if the application has such motion information available. In the future we will further explore the use of warping for video compression, possibly by designing a novel video codec that allows transmission of depth information. This would enable client-side warping and novel motion compensation means. Server-side rendering also merits further research in areas such as load-balancing, where similarities between different views can be exploited, effects of network latency and collaborative rendering.

References

[1] L. Aimar, L. Merrit, et al. x264 - a free h264/avc encoder. Available at `http://www.videolan.org/developers/x264.html` (accessed May 11, 2008), November 2007. SVN Version 680.

[2] D. Aliaga, P. Rosen, V. Popescu, and I. Carlbom. Image warping for compressing and spatially organizing a dense collection of images. *Signal Processing: Image Communication*, 21(9):755–769, October 2006.

[3] BBC. *Dirac Specification, Version 2.2.0*, April 2008. Available at `http://dirac.sourceforge.net/DiracSpec2.2.0.pdf` (accessed Apr. 23, 2008).

[4] M. Black and P. Anandan. Robust dynamic motion estimation over time. In *CVPR*, pages 296–302, 1991.

[5] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *ECCV*, volume IV, pages 25–36, 2004.

[6] C. Chang and S. Ger. Enhancing 3D Graphics on Mobile Devices by Image-Based Rendering. In *PCM '02: Proceedings of the Third IEEE Pacific Rim Conference on Multimedia*, pages 1105–1111, 2002.

[7] J. Duan and J. Li. Compression of the layered depth image. *IEEE Transactions on Image Processing*, 12(3):365–372, March 2003.

[8] T. Hudson and W. Mark. Multiple Image Warping for Remote Display of Rendered Images. Technical Report TR99-024, University of North Carolina at Chapel Hill, 1999.

[9] ISO/IEC. *Information technology – Coding of audio-visual objects – Part 2: Visual*, 2001. Reference number ISO/IEC 14496-2:2001(E) (referred to as "MPEG-4 part 2").

[10] ITU-T. *Recommendation H.264: Advanced video coding for generic audiovisual services*, March 2005. Also an ISO standard as ISO/IEC 14496-10:2005 (referred to as "H.264").

[11] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI81 Vol. 2*, pages 674–679, 1981.

[12] W. Mark. *Post-Rendering 3D Image Warping: Visibility, Reconstruction, and Performance for Depth-Image Warping*. PhD thesis, University of North Carolina at Chapel Hill,

Chapel Hill, NC, USA, 1999.

[13] I. Martin. Adaptive Rendering of 3D Models over Networks Using Multiple Modalities. Technical Report RC21722, IBM T.J. Watson Research Center, April 2000.

[14] L. McMillan, Jr. *An image-based approach to three-dimensional computer graphics*. PhD thesis, University of North Carolina at Chapel Hill, 1997.

[15] J. Shade, S. Gortler, L. He, and R. Szeliski. Layered depth images. In *Proceedings of SIGGRAPH*, pages 231–242, 1998.

[16] G. Thomas, G. Point, and K. Bouatouch. A Client-Server Approach to Image-Based Rendering on Mobile Terminals. Technical Report RR-5447, INRIA, INRIA Futurs, Bordeaux, January 2005.

[17] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004.

[18] I. Yoon and U. Neumann. Web-Based Remote Rendering with IBRAC (Image-Based Rendering Acceleration and Compression). *Computer Graphics Forum*, 19(3):321–330, 2000.