EUROGRAPHICS 2008 / G. Drettakis and R. Scopigno
(Guest Editors)

Volume 27 (2008), Number 2

Render2MPEG: A Perception-based Framework Towards Integrating Rendering and Video Compression

Robert Herzog[1], Shinichi Kinuwaki[1], Karol Myszkowski[1], Hans-Peter Seidel[1]

[1]MPI Informatik, Computer Graphics Group, Saarbrücken, Germany

Abstract

Currently 3D animation rendering and video compression are completely independent processes even if rendered frames are streamed on-the-fly within a client-server platform. In such scenario, which may involve time-varying transmission bandwidths and different display characteristics at the client side, dynamic adjustment of the rendering quality to such requirements can lead to a better use of server resources. In this work, we present a framework where the renderer and MPEG codec are coupled through a straightforward interface that provides precise motion vectors from the rendering side to the codec and perceptual error thresholds for each pixel in the opposite direction. The perceptual error thresholds take into account bandwidth-dependent quantization errors resulting from the lossy compression as well as image content-dependent luminance and spatial contrast masking. The availability of the discrete cosine transform (DCT) coefficients at the codec side enables to use advanced models of the human visual system (HVS) in the perceptual error threshold derivation without incurring any significant cost. Those error thresholds are then used to control the rendering quality and make it well aligned with the compressed stream quality. In our prototype system we use the lightcuts technique developed by Walter et al., which we enhance to handle dynamic image sequences, and an MPEG-2 implementation. Our results clearly demonstrate many advantages of coupling the rendering with video compression in terms of faster rendering. Furthermore, temporally coherent rendering leads to a reduction of temporal artifacts.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism - Rendering, Global Illumination, MPEG, Compression

1. Introduction

A server-based platform with the client access through the Internet becomes a very attractive approach to access and interact with 3D graphics. To reduce the dependence on restricted computational capabilities on the client side and to simplify handling the diversity of client devices, the server can render images and stream them to the client side using a standard video compression technique. This way the client does not require any support for rendering and a standard web browser may be sufficient for video playback and backward interaction with a 3D model on the server. This is in particular important for applications that require huge 3D data, which cannot be handled by thin clients such as smart phones, PDAs, or even laptops. In such a scenario the 3D data is stored in one reliable place (possibly with guaranteed full-time access) and does not have to be transmitted to the client, which improves interaction and simplifies control over confidential data. This also ensures that in applications involving collaborative work all users always deal with the same, fully updated 3D models.

Such client-server graphics platforms are available on the market, e.g., RealityServer developed by the mental images, Inc. company. The range of possible applications for such systems can be such diverse as: remote access to 3D data (e.g., for maintenance and repair purposes), industrial and architectural design, 3D navigation and tourism, interactive online (mobile) entertainment, and others. Such a client-server architecture can also be attractive in medical applications requiring 3D visualization and data access at any time and location.

To make such solutions practical and reduce the required bandwidth for video streaming, the MPEG and Motion JPEG encoding standards are used. However, in existing solutions the compression step is completely independent from the preceding rendering step. In this work, we demonstrate that by closer coupling of these two steps the computation redundancy can be reduced and the rendering quality can be well matched to the video quality, so the benefits of such coupling can be mutual.

The most obvious benefits come from the use of motion compensation vectors which, at the rendering stage, are inexpensive to derive for every pixel with very high accuracy [MB95]. On the compression side, this eliminates costly search for motion compensation vectors for pixel blocks [WKC94], which is one of the major bottlenecks in video encoding. Also, erroneous motion vectors due to approximate search algorithms are eliminated and variable block-size motion compensation [Bov05, Chapter 6.5] does not introduce any significant computational overhead. On the rendering side, pixel-level motion compensation enables pixel shading re-computation for consistent scene sample points tracked for subsequent frames, which reduces temporal aliasing in particular for textured regions. Motion compensation can be used for proper simulation of camera shutter speed and resulting motion blur [HDMS03] as well as filtering in temporal domain. All these techniques suppress temporal rendering artifacts, which otherwise cannot be distinguished from image features by the encoder and are reproduced in video at expense of extra bandwidth [BG00].

In this work our focus is on the rendering accuracy control. It is current practice that frames are often rendered with many details that are later discarded due to lossy video compression. This means that the rendering quality can often be reduced without affecting the video quality, which may lead to faster rendering that is important in interactive applications. In MPEG compression the video quality is controlled through varying the quantization of discrete cosine transform (DCT) coefficients (used for a pixel block representation), which effectively leads to the information loss. A good match between the compressed and rendered image quality can be achieved by imposing a stopping condition on rendering, so that further computation cannot contribute to visible pixel changes due to the quantization error. This can be facilitated by using the DCT pixel block representation both at the rendering and compression stages [BM95]. Also, such a DCT representation enables an inexpensive incorporation of a perceptual image quality metric to rendering, which additionally can adapt the quantization error to the image content by modeling important characteristics of the human visual system (HVS) such as contrast sensitivity, luminance and contrast masking [FPSG97, BM98, WPG02].

In this paper we propose a framework combining realistic rendering and MPEG video compression. Our rendering is based on the instant global illumination algorithm [Kel97, WKB*02] combined with the lightcut data structure [WFA*05, WABG06]. We extend these techniques to take advantage of temporal coherence between subsequent frames. The rendering quality is guided by a DCT-based quality metric, which maintains the rendering errors below the visibility level imposed by the quantization errors. At the same time the quantization errors are adapted to local image content to make the compression errors uniformly perceivable across the image space. Our prototype system shows many advantages of combining rendering and compression into a unified framework such as faster rendering and reduction of temporal artifacts.

In the following section we discuss previous work, which is relevant for video compression and rendering integration. We briefly summarize our system in Section 3 and then outline the extensions of the lightcut and instant radiosity techniques to fit them into our Render2MPEG framework. In Section 5 we introduce a perceptual model, which we use to steer the global illumination computation. In Section 6 we present results obtained using our techniques. Finally, we conclude this work and suggest directions of future research.

2. Previous Work

Computer graphics techniques have been used to compute motion vectors for synthetic scenes that are then used for the MPEG compression [WKC94]. The authors showed that when using their projective-texture based algorithm for motion vectors computation, better compression ratios can be achieved than for standard MPEG solutions [Bov05, Chapter 6.1], in particular for scenes which are rich in textures. Also, image-based rendering introduced by McMillan and Bishop [MB95] directly leads to the motion vector computation for each image pixel. Depth information from the previous frame is required to warp that frame and find new pixel positions in the following frame. In our system such image warping can be performed efficiently at the rendering stage, and the resulting motion vectors can directly be applied to video encoding.

Compression techniques have been also used for streaming 3D data in a client-server architecture. Cohen-Or et al. [COMF99] showed that by rendering mesh elements with projective textures on the client, and streaming only the residual error between warped texture views for subsequent frames from the server, the bandwidth-demanding transmission of full resolution textures can be avoided. However, the sequence of visible geometry and textures for streaming and rendering on the client side have been prepared off-line. Levoy [Lev95] proposes a client-server architecture in which the residual error between high- and low-quality renderings performed at the server is compressed and streamed to the client. The client performs only the low-quality rendering, adds decompressed residual error, and displays the composite. In our solution, we assume that only video decoding is performed on the client side, and we are more interested in the impact of compression for steering rendering on the server side.

Bolin and Meyer [BM95] developed a ray tracer, which projects pixel samples directly onto a DCT basis function for an 8×8 pixel block. The technique was developed for static images. As a result of such rendering a JPEG-like image representation is directly obtained. Since samples are generated sparsely for each block, a costly least squares procedure is required to approximate all DCT coefficients that best interpolate the sampled data. When samples are progressively added additional frequency terms in the DCT block representation are also added. In our approach, we avoid such costly operations by using packet ray tracing [WKB*02] that processes all 16×16 pixels in the macroblock at once with an overhead equivalent to the cost of 30 rays computed independently.

More advanced visual models have been used to steer the costly global illumination computation [BM98, RPG99, YPG01]. In all these techniques the goal was to continue the computation until rendering inaccuracies do not affect anymore the visual quality of images. Our goal is quite different, because we accept visible quality degradation under the condition that similar degradation is introduced by the following lossy video compression. Thus, we intend to adaptively control the rendering quality to make it synchronized with the available bandwidth (effectively the quantization error), which is important in Internet applications where the quality of services (QoS) is not guaranteed. We achieve this goal using a DCT-based visual model [Wat93, ZDL00], whose predictions are well aligned with typical artifacts in image and video compression. A similar model has been successfully used by Walter et al. [WPG02] for off-line estimation of visual masking by textures. In their approach, both the scene lighting and geometry are not considered and thus do not contribute to the masking prediction, which may result in too conservative (precise) rendering. Also, they do not consider the quantization error, which in our application leads to major improvements of rendering efficiency.

All discussed perception-based rendering techniques struggle with proper estimation of scene lighting as required to account for local luminance adaptation and visual masking. This is a typical “chicken-and-egg” problem, where the lighting knowledge is required to take the full advantage of the visual model, which is actually supposed to steer the lighting computation. In our approach, by taking into account the frame-to-frame coherence in lighting, we obtain a good prediction of the lighting distribution in the subsequent frame, whose computation is steered by a visual model.

3. Algorithm Overview

In the following we briefly describe the whole algorithm, which is sketched in Fig. 1. Our renderer is based on the lightcuts algorithm [WFA*05, WABG06] as lightcuts is about one to two orders of magnitude more efficient than equivalent final gathering approach based on photon mapping [Chr99, Jen01, KGPB05]. This is due to a smaller number of final gather rays that need to be evaluated per pixel (ca. 50-400) and because lightcuts, based on instant radiosity, does not require computationally intensive density estimation queries in the photonmap. Nonetheless, per pixel computation is still too costly for efficient rendering in particular for higher resolution images. Therefore, we exploit the spatial coherence of the indirect illumination. In contrast to the interpolation of pixel radiance as proposed in the *reconstruction cuts* [WFA*05], we favor irradiance interpolation of the incident lighting in object space [WRC88, KGPB05]. This decouples the interpolation from the surface material and prevents interpolation errors for high-frequency textures and surface boundaries. Since irradiance caching works only robustly for smooth lighting, we separate the high-frequency lighting components. A common way is to compute direct and indirect light separately since direct lights mostly influence the whole image whereas the weaker indirect lights often have only local impact on the rendered image. Therefore, we compute for all pixels direct lighting as well as dynamic indirect lighting using standard instant radiosity techniques [Kel97] with shadow mapping. We generate parabolic shadow maps on the GPU and use them for per pixel accurate lighting computations.

Summarizing, our algorithm works as follows: First, a uniform grid is updated for all dynamic objects, which is used as raytracing acceleration data structure. Next, photons are traced through the scene splatting their energy to a fixed set of virtual point lights (VPLs), which we refer to as anchor lights. In the case of the first frame we construct a light tree over the anchor lights as described in Section 4.1. Otherwise all internal nodes in the tree are updated as described in Section 4.1.1. Then we shoot all primary (eye) rays for the current frame using packet raytracing [WKB*02] for 16×16 rays corresponding to one MPEG macro-block and find correspondences in the previous frame by backward reprojection. This way we can fill our error-threshold map with the previously computed visibility thresholds as explained in Section 5.3, which are then used to control the accuracy of the global illumination computation for the current frame. After computing direct and dynamic indirect lighting for each pixel using shadow maps, we compute the indirect lighting based on irradiance caching. Note that our error-threshold map is intended to steer the lightcuts computation on a per pixel basis. However, when using irradiance caching a cache sample influences a large neighborhood of pixels, which are likely to have a different rendering error-threshold. As a remedy, we filter the error threshold for a cache by averaging the tolerable rendering errors over the cache's footprint in image space. For weighting the error thresholds we use a Gaussian filter kernel. Finally, we compute the new error threshold map as described in Section 5.2, which is then used for the next frame.

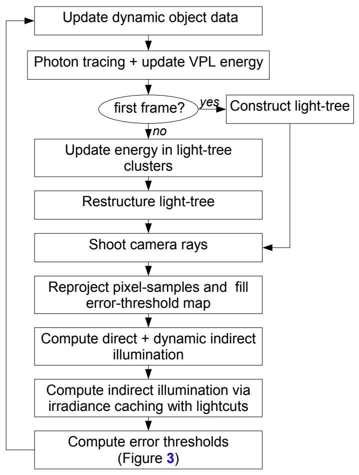

Update dynamic object data
Photon tracing + update VPL energy
first frame?
yes
Construct light-tree
no
Update energy in light-tree clusters
Restructure light-tree
Shoot camera rays
Reproject pixel-samples and fill error-threshold map
Compute direct + dynamic indirect illumination
Compute indirect illumination via irradiance caching with lightcuts
Compute error thresholds (Figure 3)

Figure 1: *Flow in the rendering pipeline for computing one frame.*

4. Temporally Coherent Lighting Computation

Although temporally coherent global illumination is not the main focus of this work, for efficient video coding and motion compensation on MPEG side, we favor spatially and temporally coherent illumination in contrast to unbiased solutions [RPG99, BM98, BM95], which trade off bias for high-frequency noise. Note that noise is well preserved in the DCT-based MPEG compression and leads to undesirable bandwidth expansion [BG00]. We chose a hierarchical version of instant radiosity [Kel97] based on lightcuts [WFA*05] because lightcuts is a pixel-adaptive, scalable global-illumination algorithm and perhaps more importantly, it allows to control the computation by a perceptual rendering error. However, the original lightcuts algorithm has been developed for still images [WFA*05]. In the temporal domain only short time intervals have been considered to model the effect of motion blur [WABG06]. In order to support longer animation sequences with fully dynamic scenes, we extend lightcuts into the temporal domain. This imposes several constraints to the algorithm in order to suppress temporal noise:

- A constant number of virtual point light sources (VPLs) in the scene are considered for the whole animation sequence.
- VPLs are sampled uniformly across the scene surfaces and their positions are fixed (similar to the anchor [SKDM05] and gather [HPB06] samples). We call those VPLs anchor lights.
- Only VPL intensities are updated from frame to frame using coherent photon splatting.
- The light hierarchy, which is built on top of the VPLs as in [WFA*05], is not rebuilt but only updated from frame to frame.

During computation of the pixel radiance only a small fraction of anchor lights is evaluated per pixel, which are chosen adaptively by computing a cut in the light hierarchy. While this lightcut is computed as in [WFA*05], the pixel error thresholds that determine the cut size adapt to the HVS and the quantization error imposed by MPEG.

4.1. Construction of a Light Hierarchy

As in the original lightcuts algorithm [WFA*05], the light hierarchy is constructed only once using a costly greedy bottom-up construction to cluster any two virtual point lights (VPLs) at a time that minimize a cost function (e.g. volume of associated bounding box and bounding cone weighted by their summed energy). See [WFA*05] for more details. An alternative approach using a recursive top-down construction of the point-light hierarchy is much more efficient and easier to implement but decreases the rendering performance by ca. 3% - 15% due to larger lightcuts arising from less precise error bounds. For the dynamic case, we need to be able to reconstruct and update the hierarchy efficiently. This is feasible since we assume lighting and scene geometry is temporally coherent such that only little changes in the hierarchy have to be made per frame. Besides, it is desirable to keep the lighting temporally coherent since this reduces flickering, which increases also the efficiency of the MPEG compression. As in [WFA*05] a cluster node in the light hierarchy has two children and shares the geometric properties with one child node, the representative child. The representative child is always chosen stochastically with the probability proportional to the relative light source intensity. This is important because it ensures that the induced rendering errors of individual clusters are uncorrelated and do not accumulate (see Fig. 6b). Because VPLs on dynamic objects violate our assumptions about temporally coherent lighting, they are not inserted into the light hierarchy but are handled separately as in standard instant radiosity [WKB*02, Kel97].

4.1.1. Updating the Light Hierarchy

At first, the intensity of all anchor lights at the leaf level of the hierarchy is updated by photon density estimation. In order to maintain temporal coherence, the raytraced photon paths are regenerated for successive frames using the same Halton number sequence when performing the random walk, i.e. in the case of only static objects no lighting changes will appear. Even in the dynamic case the energy of most anchor lights changes smoothly in time and abrupt lighting changes are rather of local nature. Therefore updates in the light tree structure and intensity affect mostly the lower levels in the tree and propagate slowly up the hierarchy, which further suppresses temporal noise in the radiance computation. Although changes in the light hierarchy are minor between successive frames, they may accumulate in the long run and eventually require a reconstruction of the entire hierarchy. Since higher levels in the light tree keep the sum of the intensities of their sub-trees, intensity changes in the anchor lights at the leaf level are simply propagated up to the cluster nodes of the hierarchy.

In the original lightcuts approach, the tree is built such that the VPL with highest energy is most likely to be at the top of the hierarchy, which cannot be ensured if we keep the tree static. In case of occlusion, e.g. a dynamic object moves in front of a cluster's representative anchor light, the corresponding node should descent the hierarchy according to the tree construction metric [WFA*05]. To model this behavior, we swap the two children of a node in the light tree if the intensity ratio of representative child to non-representative child is smaller than a threshold ($C_I = 0.7$), thus changing the representative/non-representative relationship of parent and child nodes. Only pointers to the representative child and anchor light need to be updated in a bottom-up manner. Therefore the overall tree structure is kept static and updating the tree becomes very efficient since the tree continuously reorganizes itself over time.

5. Rendering Accuracy Control

Aligning the rendering errors with the compression errors as imposed by the bandwidth of a given video streaming channel is an important goal of our Render2MPEG framework. A simple strategy is to keep the rendering errors below the quantization error imposed by the only lossy operation in video encoding: the quantization of DCT coefficients. Such a quantization error is determined for each DCT basis function used to represent pixel values in 8×8 pixel blocks. For a given video-stream bandwidth controlled by the value of the q_{scale} parameter, the quantization error amounts to the product of $q_{scale} \cdot q_{ij}$, where q_{ij} are predefined coefficients in the 8×8 quantization matrix Q. In our current implementation we always use the default quantization matrix Q for intra-frame coded blocks as specified in the MPEG-2 standard (ISO/IEC 13818) [MPE] and we let the user set q_{scale}.

While the matrix Q takes into account the contrast sensitivity function (CSF) attributed to the HVS, it does not adapt the quantization error to the frame's local content. It is well known that the quantization errors are less visible in cluttered image regions due to visual masking, which decreases the sensitivity for image details [BM95, FPSG97]. Also, the visual sensitivity in terms of detection of absolute luminance thresholds reduces in bright image regions due to luminance masking [Dal93]. We model these effects locally for each block, and we incorporate elevated sensitivity thresholds due to masking into the quantization error $q_{scale} \cdot q_{ij}$. This enables even more aggressive rendering. Note that while q_{scale} is a convenient parameter to control the compression bandwidth, it correlates poorly with the optimum visual quality that can be achieved at a given bit rate [Wat93].

This section is organized as follows. In Section 5.1 we present the derivation of quantization error with incorporated masking effects. In Section 5.2 we describe our approach to transform the resulting quantization error from the frequency domain into the spatial domain as required by our renderer. Obviously, the most reliable estimate of quantization errors can be achieved when the image content is fully known. For this reason we apply the visual model to the previously computed frames in order to predict the quantization error to be used in the current frame rendering. We discuss various strategies of transferring the quantization error from frame to frame in Section 5.3.

5.1. Luminance and Contrast Masking Model

The visual model used by us to predict luminance and spatial masking is inspired by research in image and video compression [Wat93, ZDL00]. The luminance masking model as proposed in [Wat93] requires luminance values, which are dependent on the particular display characteristics such as the luminance range and gamma correction. Since the masking model needs to consider observed luminance of pixels on a particular display device, we have to consider the γ of such display (e.g. $\gamma = 2.0$) before applying the luminance masking model to our DCT coefficients. Since the luminance masking model proposed in [Wat93] is a simple power function, the display gamma is easily accounted for by multiplying γ with the luminance masking exponent (0.649) [Wat93].

$$t_{ij}^k = q_{ij} \cdot \left(\frac{c_{00}^k}{\bar{c}_{00}} \right)^{\gamma \cdot 0.649}, \qquad (1)$$

where c_{00}^k denotes the DC coefficient of block k and $\bar{c}_{00}$ the average DC coefficient of all blocks. Intuitively, this luminance masking model increases the threshold of tolerable quantization errors q_{ij} for brighter image regions and decreases in darker regions as predicted by the Weber's law.

Then, we estimate the tolerable elevation of quantization error m_{ij}^k due to contrast masking [Wat93] as:

$$m_{ij}^k = t_{ij}^k \cdot \max\left(1, \left| \frac{c_{ij}^k}{t_{ij}^k} \right|^{0.7} \right), \qquad (2)$$

where c_{ij}^k denotes the ij-th DCT coefficient of block k. Following [Wat93] we ignore contrast masking for the DC coefficient c_{00}^k, i.e., we assume that $m_{00}^k = t_{00}^k$. Intuitively, this contrast masking model increases the threshold of a tolerable quantization error for regions with high contrast patterns of a spatial frequency represented by the given coefficient c_{ij}^k.

The localized quantization error m_{ij}^k can be used to control rendering accuracy for each block in order to make the distribution of the perceptual error uniform across all blocks of the image. Note that at the compression stage we could also use the localized quantization error m_{ij}^k to modify the matrix Q for each block as proposed in [RW96], but at present optimizing video bandwidth is less important than speeding up rendering.

5.2. Maximum Tolerable Error in Rendering

From the rendering perspective the localized quantization error m_{ij}^k derived in Eq. (2) expresses the maximum tolerable rendering error as imposed by the quantization matrix and the local masking effects. This error must be then re-scaled by q_{scale} to mirror the user-imposed compression bandwidth. Since each block k is quantized by dividing all its coefficients c_{ij}^k by $m_{ij}^k \cdot q_{scale}$ and rounding to the nearest integer, the maximum possible quantization error is $\frac{1}{2} \cdot m_{ij}^k \cdot q_{scale}$. By exploiting the linearity property of the DCT transform and preserving the polarity of DCT coefficients using the sign(c_{ij}^k) function, we can conservatively construct the worst case distorted frame with DCT coefficients $\hat{c}_{ij}^k$ by adding the maximum possible quantization error to the original frame:

$$\hat{c}_{ij}^k = \begin{cases} \frac{1}{2}\text{sign}(c_{ij}^k) \cdot q_{scale} \cdot \min\limits_{ij}(q_{ij}) & \text{if } c_{ij}^k < \frac{1}{2} q_{ij} \cdot q_{scale} \\ c_{ij}^k + \frac{1}{2}\text{sign}(c_{ij}^k) \cdot q_{scale} \cdot m_{ij}^k & \text{otherwise,} \end{cases} \qquad (3)$$

which after compression should be visually equivalent to the original frame (c_{ij}^k) when both are compressed with the same q_{scale} value.

When computing $\hat{c}_{ij}^k$ for the coefficients c_{ij}^k that do not vanish as the result of lossy compression (case 2 in Eq. (3)), the quantization error including the masking effects are considered. However, the signal represented by the remaining c_{ij}^k coefficients (case 1 in Eq. (3)) is removed from the compressed image (i.e. $\tilde{c}_{ij}^k = 0$) and therefore contrast masking for the corresponding frequencies as predicted by Eq. (2) is not valid. In this case, even considering only the quantization error resulting purely from the MPEG compression (i.e. $\hat{c}^k_{ij} := \frac{1}{2} q_{ij} \cdot q_{scale}$) may lead to overestimated errors and reduced rendering quality for higher q_{scale} values. Since the largest quantization errors q_{ij} are assigned to high frequency coefficients (which usually vanish as the result of lossy compression), they tend to dominate in the tolerable rendering error estimate. Because this error estimate is finally used as upper bound for the rendering error in the spatial domain (see Fig. 2), its spatial frequency selectivity inherent for the DCT domain is lost. Consequently, such high quantization errors may also contribute to excessive tolerance for errors in the lower frequency signals (higher eye sensitivity), which leads to visible image distortions. Note that low-frequency quantization errors may also generate high-frequency rendering errors, which is less critical since quantization errors increase for higher frequencies. One solution is to leave those coefficients that are invisible after quantization unaltered [WPG02]. However, such approach does not scale well with higher q_{scale} values where most frequencies are removed from the signal. Therefore, we set those coefficients to the minimum quantization for AC coefficients ($\min(q_{ij}) = 16$), which is adaptively tuned to the current video bandwidth by the q_{scale} multiplier. Although this approach may still be too conservative, it produces good results in terms of rendering efficiency and robustness.

After performing the inverse DCT on $\hat{c}^k_{ij}$ and inverse tone-mapping to get the distorted luminance $\hat{Y}_{xy}$ for every pixel (x,y), we compute the maximum tolerable pixel errors e_{xy}:

$$e_{xy} = \max(0.02 \cdot Y_{xy}, |\hat{Y}_{xy} - Y_{xy}|). \quad (4)$$

as the absolute difference between the luminance Y_{xy} of the original undistorted pixel and $\hat{Y}_{xy}$. In order to avoid too small error thresholds, we clamp the result at a "perceptually conservative" lower bound of 2% of the pixel's luminance Y_{xy} [WFA*05].

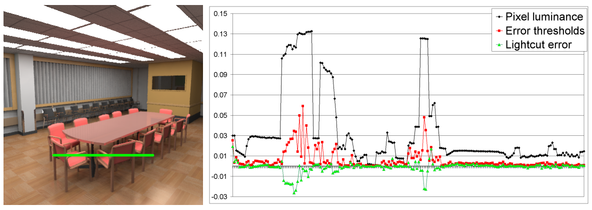


Figure 2: *Pixel luminance (black), error thresholds (red), and rendering errors (green) along a scanline in the left image (green line).*

The error e_{xy} can change from frame to frame as a function of image content, but also can be affected by variable network bandwidth. In the following section we discuss temporal aspects of handling e_{xy}.

5.3. Temporal Handling of Tolerable Rendering Error

In this section we discuss the problem of re-using the maximum tolerable error e_{xy} from the previous frames to steer the current frame computation. Another important issue is temporal coherence of the error between subsequent frames, which is necessary to reduce video flickering. Such temporal coherence is improved by blending the tolerable error between previous $(t-1)$ and current frame t, such that the blending weights fall-off exponentially for older frames. The final rendering error-thresholds that are then stored in the threshold map are computed as:

$$\Delta Y_{xy}(t) = \begin{cases} e_{xy} & \text{if } (x,y) \not\mapsto (x',y') \\ (1-w)e_{xy} + w\Delta Y_{x'y'}(t-1) & \text{otherwise,} \end{cases} \quad (5)$$

where the mapping $(x,y) \mapsto (x',y')$ to the corresponding error threshold in the previous frame is obtained through back-projecting the current frame's pixel sample (effectively the camera and object motion compensation is performed). When occluded/disoccluded/non-existing region in frame $(t-1)$ is identified by the backprojection (case $(x,y) \not\mapsto (x',y')$) then we simply use the error e_{xy} estimated for frame t. The blending weight w is a trade-off between temporal coherence and error propagation. When using a larger blending weight our error thresholds can become less accurate for successive frames while lighting errors are kept coherent and vice versa. We set $w = 0.5$ which, according to our tests, does not seem to bias the error in our threshold map for future frames.

The resulting error thresholds $\Delta Y_{xy}(t+1)$ are then used for the future frame $(t+1)$ to decide upon the stopping condition in the lightcut computation: if the maximum upper error bound of all clusters in the pixel's current lightcut is below our error threshold or the lightcut size exceeds a maximum of 1,000 clusters, we stop refining the lightcut. The lightcut is computed as in [WFA*05] using a priority queue. A lightcut sample for one pixel with 3% error threshold is shown in Fig. 6b. Error thresholds and actual lightcut rendering-errors for a scanline are visualized in Fig. 2. Note that rendering errors (green curve) alternate around zero but are coherent for neighboring pixels with similar error threshold.

The entire flow of the threshold computation is shown in Fig. 3.

We applied the processing flow as shown in Fig. 3 to two basic strategies of the error handling with respect to the previous frames:

1. Quantization-*dependent* masking computation *without* motion compensation.
2. Quantization-*independent* masking computation *with* motion compensation.

The first case means applying the masking model in Eq. (1) and Eq. (2) to the quantized DCT coefficients $\hat{c}^k_{ij}$, thus also considering quantization errors in the masking prediction, which results in more relaxed rendering errors. This is because quantization errors in the final image such as blocking artifacts are also included in the contrast masking. This approach is only valid for scenes with slow camera and object motions where block boundaries from the previous frame are aligned with blocks in the current frame.

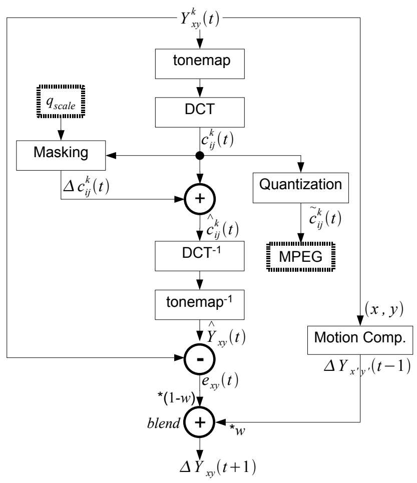


Figure 3: *The computation flow for deriving error thresholds $\Delta Y_{xy}(t)$ for all pixels (x,y) in the frame computed at time t. $\Delta Y_{xy}(t+1)$ is used to steer rendering of the subsequent frame at time $(t+1)$.*

The second approach is more conservative when computing the error thresholds at the expense of having smaller rendering errors in particular for higher MPEG compression settings, where masking due to quantization dominates. In this case the DCT coefficients c^k_{ij} are directly used to predict masking as it is shown in Eq. (1) and Eq. (2). Motion compensation helps in aligning masking with image details and even significant motion of camera and rigid objects can be successfully handled. The same motion compensation procedure as between frames $(t-1)$ and (t) is now applied to frames (t) and $(t+1)$. For pixels that do not find a feasible sample in the four nearest neighbor pixels of the previous frame, a maximum relative rendering error of 2% is assumed. A pixel sample is assumed to be feasible if its pixel depth and surface normal are similar.

6. Results

We have tested our framework on 4 scenes with different lighting, frequency content, and complexity. For visualization and validation purposes the results of the CONFERENCE scene, shown in Fig. 4, are computed at a per pixel basis in order to compare with the original lightcuts algorithm [WFA*05] that uses a maximum luminance error per pixel of 2%, which we have used as the reference in our evaluations. Since the original lightcut algorithm has been designed for high-quality rendering with a large number of point light sources (VPLs), we compare our error metric also using a larger number (150,000) of VPLs in the CONFERENCE scene. The other scenes, CORNELL BOX, SPONZA ATRIUM, SIBENIK CATHEDRAL, were generated in a dynamic scenario and the videos are provided at [Web08]. The relevant statistics for the animated scenes shown in Fig. 5 are given in Tables 1. To speedup rendering, the irradiance caching algorithm [WRC88, KGPB05] has been used. Note that for efficiency reasons the MPEG-2 encoder [MPE] uses simple integer arithmetic in the quantization. Therefore, the provided

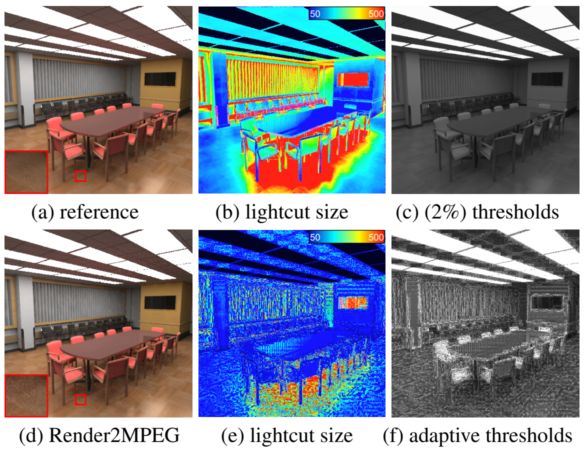


Figure 4: *Comparing original lightcuts sampling using 2% error metric (first row) with our MPEG-driven perceptual error metric (second row) where the rendering is adapted to compression level with $q_{scale}=16$. (MPEG-encoded frames are shown in Fig. 8.) The color-encoded number of considered lights per pixel is given in the second column with an average of 281 lights for the reference and 152 using our method, respectively. The error threshold maps (scaled by 32 and displayed with $\gamma=2.6$) are shown in the third column. The average error threshold is 26% in our case.*

q_{scale} values in the results, which are given as input to the encoder, should be divided by 16 to corresponds to the actual quantization errors used in Section 5.2.

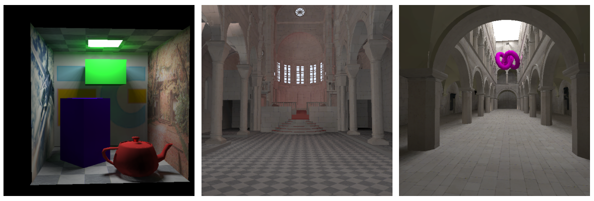


Figure 5: *Animated test scenes (left to right): textured* CORNELL BOX, *indirectly lit* SIBENIK, SPONZA ATRIUM. *Videos are provided at [Web08].*

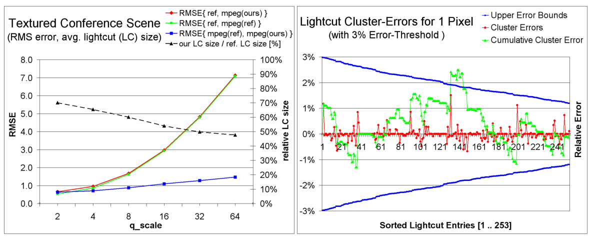
Our numerical results were computed on a single PC equipped with an Athlon 64 2.4 GHz with 4 GByte of memory and a GeForce 6800 based graphics card. The statistics in Table 1 show (from left to right): scene settings including number of direct and indirect anchor lights (VPLs) and the number of photon splats to update the indirect VPL intensities, the q_{scale} quantization multiplier given by the MPEG encoder, the average relative error-threshold per frame (i.e. divided by pixel luminance), the average number of swapped cluster nodes in the light tree per frame, the average lightcut size with respect to the total number of anchor lights (VPLs), the average computation times in seconds per frame for photon tracing and energy splatting to anchor lights (T_{light}), for computing the shadow maps for direct and dynamic indirect light on the GPU (T_{gpu}), for primary (eye) ray casting (T_{eye}), for the lighting computation including lightcut evaluation and irradiance caching (T_{cut}), the average total rendering time in seconds per frame, and the speedup of the indirect lighting computation relative to the reference solution, which uses a constant error threshold of 2% of the pixels luminance value (see Fig. 4c for an example). Our error-threshold computation including a discrete cosine transforms of all blocks and motion compensation takes 0.2 sec per frame for all three animations. The memory utilization is less than 50 MBytes for the tested scenes. Since the SIBENIK scene is shown in a walk-through animation, lighting changes and updates of the light tree are not necessary. The only cost is the initial lighting computation and the light tree construction. The main bottleneck so far is the random photon walk (T_{light}), which uses the same data structure (uniform grid) for Monte Carlo ray tracing that is actually optimized for packet-ray tracing of primary eye rays (T_{eye}).

In Fig. 6a we show per pixel statistics of rendering/compression error and relative lightcut (LC) size for the textured CONFERENCE scene shown in Fig. 4 for various compression settings. The lightcut size (black dashed curve) is given relative to the reference using 281 clusters on average. It directly mirrors the saving in computation time for this particular scene, which range from approximately 30% to 60% with increasing q_{scale}. Fig. 6b shows the upper bounds and real lightcut errors for one pixel with a cut size of 253 and 3% error threshold. Note that the computed upper error bounds (blue) are always greater than the real cluster errors (red) but not necessarily greater than the sum over the individual cluster errors (green), whose distribution is presented in Fig. 7a. Fig. 7a shows that for the vast majority of pixels the rendering errors introduced by the lightcut computation are smaller than their error thresholds (green points). Nevertheless, there are a few outliers occasionally (red points). Fortunately, outliers are mainly pixels that have a relatively small error-threshold as shown in Fig. 7b, which is often too conservative anyway (see Eq. (4)). These observations hold also for the other scenes we have tested.

In Fig. 8 we demonstrate how our rendering adapts to varying bit rates by means of the parameter q_{scale} provided by MPEG. In the first row the absolute error thresholds to be used for the next frame are shown. Below are given the numerical values for the average relative error-threshold $\frac{\Delta Y_{xy}}{Y_{xy}}$ in percent (i.e. absolute luminance threshold divided by pixel luminance). The second row shows our compressed rendering results using the thresholds above and their peak signal-to-noise ratio (PSNR), which are visually equivalent to the compressed reference images. We observed that even though the compressed images of reference and our solution differ slightly (blue curve in Fig. 6a), their RMS error (RMSE) with respect to the uncompressed reference is very similar (see red and green curves in Fig. 6a) because the quantized pixel values fluctuate around the estimated reference value. The third row shows the residual (the rendering error) of the reference frame and our rendered frame before compression for visualization purposes scaled by a factor of 32. The PSNR and the average lightcut size per pixel are given below the images. One can observe that the rendering error does not scale in the same way as the compression error does. This is because the enforced error thresholds are upper bounds for the rendering error introduced by the lightcuts algorithm, which is usually much smaller for most pixels (see Fig. 7a).

Scene+Setting times in sec	q_{scale}	$\frac{\Delta Y_{xy}}{Y_{xy}}$	LC-tree changes	LC size	T_{light}	T_{gpu}	T_{eye}	T_{LC}	Time/ frame	Speedup only LC
CORNELLBOX	2	7.9%	1.1%	8.0%	2.5	0.8	0.1	3.0	6.6	×1.5
2000 VPLs	48	14.4%	1.1%	2.7%	2.5	0.8	0.1	1.6	5.2	×3.1
$3{\cdot}10^5$ photons	-	2.0%	1.1%	10.9%	2.5	0.8	0.1	4.0	7.5	×1.0
SPONZA	2	5.7%	2.4%	5.1%	14.2	1.1	0.5	19.0	35.0	×1.9
8900 VPLs	48	12.9%	2.4%	0.9%	14.2	1.1	0.5	9.1	25.1	×5.2
10^6 photons	-	2.0%	2.4%	6.8%	14.2	1.1	0.5	34.5	50.4	×1.0
SIBENIK	2	7.0%	0.0%	5.5%	(40.0)	0.0	0.7	6.5	7.4	×1.7
9100 VPLs	48	15.4%	0.0%	1.6%	(40.0)	0.0	0.7	4.4	5.3	×3.3
10^6 photons	-	2.0%	0.0%	7.4%	(40.0)	0.0	0.7	12.4	13.2	×1.0

Table 1: *Statistics and computation times (in seconds) for the rendering phases of our algorithm for the 3 animated test scenes shown in Fig. 5. Image resolution is* 512×512. *The shown speedup is only for the lightcut (LC) computation without irradiance caching.*



(a) (b)

Figure 6: *(a): Root mean square errors (RMSE) and relative lightcut size for varying q_{scale} in the* CONFERENCE *scene shown in Fig. 8. (b): Lightcut cluster-errors of a pixel in the image of the conference scene with an error threshold of 3%. The graph visualizes the sorted lightcut (size 253) with descending upper error bounds. The cumulative sum (green curve) of the individual cluster errors (red curve) shows the actual rendering error, which is close to 0 for this particular pixel (see point 253 of the green curve). Since the individual cluster errors can be considered as independent and identically-distributed random variables with finite variance, the actual rendering errors are approximately normal distributed (see Fig. 7a).*

7. Discussion and Future Work

This work serves as a proof of concept and there are a few limitations of our system. The choice of MPEG-2 [MPE] instead of more suitable for streaming MPEG-4 was driven by such factors as simplicity and availability of the source code. The choice of lightcuts imposes a few restrictions on the lighting computation such as clamping of close VPLs and missing caustics illumination. Besides, the separation of dynamic and static objects and the irradiance caching limits the generality of the lightcuts algorithm. The developed error thresholds for rendering are too conservative with respect to the MPEG compression error. We expect therefore that further investigations about the correlation between the rendering error of lightcuts and the quantization error in MPEG might enable even more aggressive rendering.

Even though our rendering is too slow for interactive streaming applications, we demonstrated that such properties as temporal coherence, low noise level, and local (pixel or

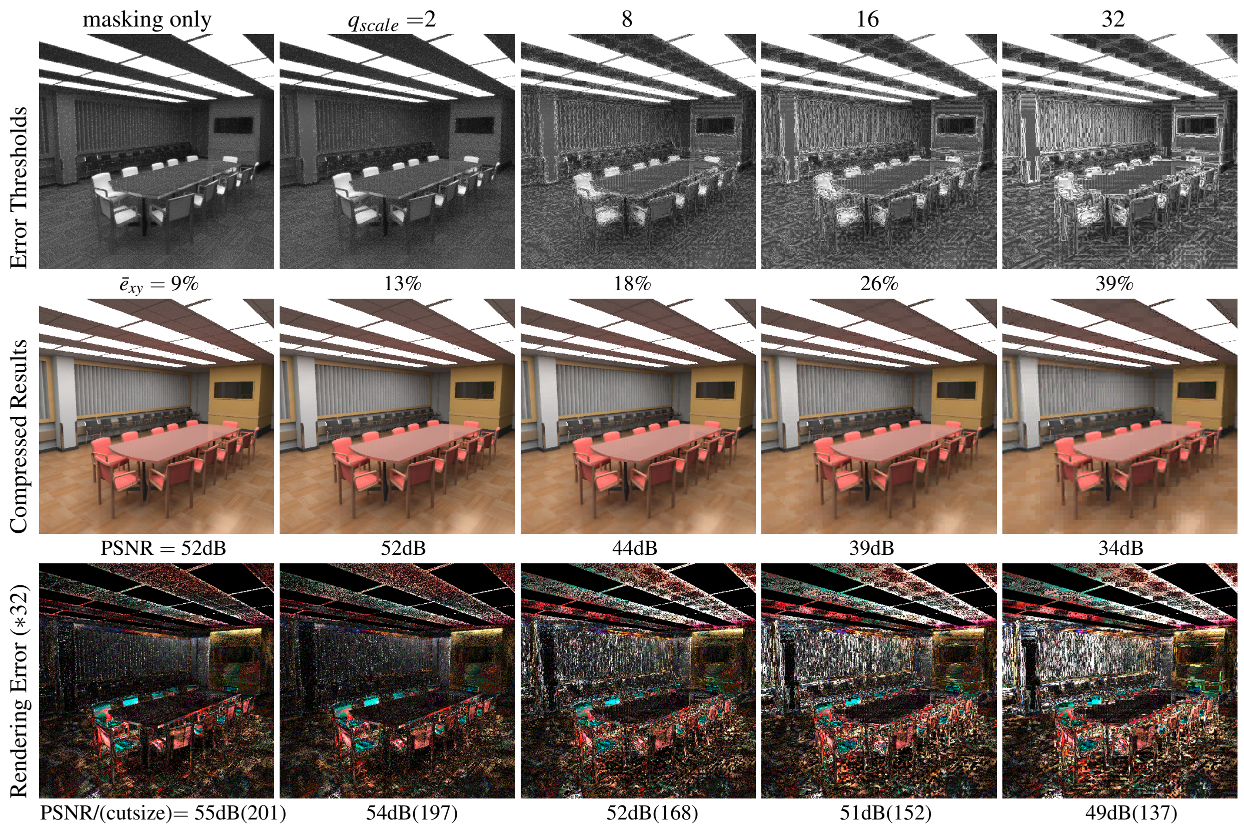
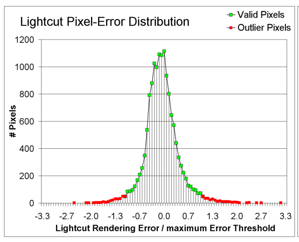
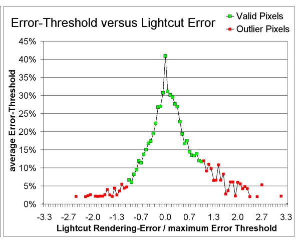


Figure 8: *Absolute rendering error thresholds (first row) for corresponding q_{scale} value (columns) driving our global illumination rendering and compression. The compressed images for varying q_{scale} with their peak signal-to-noise ratio (PSNR) are shown in the second row. While our compressed images stay visually equivalent to the compressed reference images, the rendering errors with respect to the reference solution increase steadily for higher q_{scale}. The absolute rendering error (difference between our rendered images and reference images) are shown in the third row (magnified by factor 32). The values in brackets correspond to the average resulting lightcut size.*

irradiance cache level) accuracy control are desirable in Render2MPEG applications. By mapping our lightcut algorithm to a multi-processor architecture as in [WKB∗02] interactive performance should be feasible due to similarities between the instant global illumination and lightcut techniques.





(a) (b)

Figure 7: *(a) Histogram of the actual lightcut pixel-errors in the image of the conference scene (q_{scale} = 32). The x-axis represents the ratio of actual rendering error and our maximum tolerated error-thresholds for that pixel. The green points represent the valid pixels for which the rendering error is below our imposed error thresholds. Red points in the graph indicate outliers which have a higher actual error than tolerated. A plot of the lightcut-error for a single pixel is shown in Fig. 6b. (b) The average error threshold (y-axis) for the corresponding rendering error ratio (same domain as in chart (a)). Intuitively, the higher the pixel's error-threshold, the more likely is that the rendering error will be smaller than its threshold.*

By closely coupling rendering errors with q_{scale} controlling the bandwidth of compressed streams, we can easily support two different encoding modes: CBR (constant bit rate, which leads to variable quality) and VBR (variable bit rate, which enables constant quality). Our error metric naturally adapts the rendering quality to the required quantization level no matter what encoding mode is used, although delayed by one frame.

Since our metric operates on tone mapped pixels, implicitly it takes into account tone mapping characteristics, which ideally should be adjusted to each display device and surrounding lighting conditions at the client side.

In the current implementation we decided to estimate the rendering error thresholds (refer to Section 5.2) for each frame. As future work we leave experimentation with computing the error map just for I-frames and propagating it along motion compensation vectors for P and B frames. It can be expected that more relaxed error estimates could be considered for P- and B-frames, but this also requires further studies.

8. Conclusions

In this work we investigated the problem of simultaneous control of accuracy for rendered and compressed video frames. We demonstrated that by taking into account MPEG's quantization mechanisms and basic HVS characteristics in deriving the perceptual error thresholds, we could significantly improve the rendering performance by relaxing rendering errors, while obtaining video streams with frames surprisingly similar to the compressed high-quality reference frames. By exploiting temporal coherence in rendered frames we could acquire information required by our HVS model and successfully predict the tolerable error map for frames to be rendered, which is a notorious problem for many of existing perception-based rendering solutions. Our results clearly show that stronger integration of rendering and compression software is desirable to avoid redundant computation by existing frame-by-frame standalone renderers. In this context, algorithms for temporally coherent global illumination computation become important and our extension of the lightcut algorithm aimed in this direction.

References

[BG00] BORDER P., GUILLOTEL P.: Perceptually adapted MPEG video encoding. In *IS&T/SPIE Conf on Human Vision and Electronic Imaging V* (2000), Proceeding of SPIE, volume 3959, pp. 168–175.

[BM95] BOLIN M. R., MEYER G. W.: A frequency based ray tracer. In *Proceedings of SIGGRAPH* (1995), pp. 409–418.

[BM98] BOLIN M. R., MEYER G. W.: A perceptually based adaptive sampling algorithm. In *Proceedings of SIGGRAPH* (1998), pp. 299–310.

[Bov05] BOVIK A. (Ed.): *Handbook of Image and Video Processing*. Elsvier, Academic Press, 2nd ed., 2005.

[Chr99] CHRISTENSEN P. H.: Faster Photon Map Global Illumination. In *Journal of Graphics Tools* (1999), vol. 4, pp. 1–10.

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

[Dal93] DALY S.: The Visible Differences Predictor: An algorithm for the assessment of image fidelity. In *Digital Images and Human Vision* (1993).

[FPSG97] FERWERDA J. A., PATTANAIK S. N., SHIRLEY P. S., GREENBERG D. P.: A model of visual masking for computer graphics. In *Proceedings of SIGGRAPH* (1997), pp. 143–152.

[HDMS03] HAVRAN V., DAMEZ C., MYSZKOWSKI K., SEIDEL H.-P.: An efficient spatio-temporal architecture for animation rendering. In *Eurographics Symposium on Rendering* (2003), pp. 106–117.

[HPB06] HAŠAN M., PELLACINI F., BALA K.: Direct-to-indirect transfer for cinematic relighting. *ACM Transactions on Graphics 25*, 3 (July 2006), 1089–1097.

[Jen01] JENSEN H. W.: *Realistic Image Synthesis Using Photon Mapping*. AK, Peters, 2001.

[Kel97] KELLER A.: Instant radiosity. In *Proceedings of SIGGRAPH* (1997), pp. 49–56.

[KGPB05] KŘIVÁNEK J., GAUTRON P., PATTANAIK S., BOUATOUCH K.: Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics 11*, 5 (2005).

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

[MB95] MCMILLAN L., BISHOP G.: Plenoptic modeling: An image-based rendering system. In *Proceedings of SIGGRAPH* (1995), pp. 39–46.

[MPE] MPEG-2: Free mpeg-2 encoder software. http://www.mpeg.org/MPEG/video/.

[RPG99] RAMASUBRAMANIAN M., PATTANAIK S. N., GREENBERG D. P.: A perceptually based physical error metric for realistic image synthesis. In *Proceedings of SIGGRAPH* (1999), pp. 73–82.

[RW96] ROSENHOLTZ R., WATSON A. B.: Perceptual adaptive JPEG coding. In *IEEE International Conference on Image Processing* (1996), pp. 901–904.

[SKDM05] SMYK M., KINUWAKI S., DURIKOVIC R., MYSZKOWSKI K.: Temporally coherent irradiance caching for high quality animation rendering. *Proceedings of Eurographics 2005 24*, 3 (2005), 401–412.

[WABG06] WALTER B., ARBREE A., BALA K., GREENBERG D. P.: Multidimensional lightcuts. *ACM Transactions on Graphics 25*, 3 (2006), 1081–1088.

[Wat93] WATSON A.: DCT quantization matrices visually optimized for individual images. In *Human Vision, VisualProcessing, and Digital Display IV* (1993), SPIE, volume 1913-14, pp. 202–216.

[Web08] WEBSITE: Render2mpeg project, 2008. http://www.mpi-inf.mpg.de/resources/anim/EG08/.

[WFA*05] WALTER B., FERNANDEZ S., ARBREE A., BALA K., DONIKIAN M., GREENBERG D.: Lightcuts: A scalable approach to illumination. In *Proceedings of SIGGRAPH* (2005), pp. 1098 – 1107.

[WKB*02] WALD I., KOLLIG T., BENTHIN C., KELLER A., SLUSALLEK P.: Interactive global illumination using fast ray tracing. In *Eurographics Workshop on Rendering* (2002), pp. 15–24.

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

[WPG02] WALTER B., PATTANAIK S. N., GREENBERG D. P.: Using perceptual texture masking for efficient image synthesis. *Computer Graphics Forum 21*, 3 (2002), 393–399.

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. In *Proceedings of SIGGRAPH* (1988), pp. 85–92.

[YPG01] YEE H., PATTANAIK S., GREENBERG D.: Spatiotemporal Sensitivity and Visual Attention for Efficient Rendering of Dynamic Environments. *ACM Transactions on Graphics 20*, 1 (January 2001), 39–65.

[ZDL00] ZENG W., DALY S., LEI S.: Visual optimization tools in JPEG 2000. In *IEEE Intern. Conf. on Image Processing* (2000), pp. 37–40.