Signal Processing: *Image Communication* 24 (2009) 73–88

Contents lists available at ScienceDirect

Signal Processing: *Image Communication*

journal homepage: www.elsevier.com/locate/image

The effects of multiview depth video compression on multiview rendering

P. Merkle[a,*], Y. Morvan[b], A. Smolic[a], D. Farin[b,1], K. Müller[a], P.H.N. de With[b], T. Wiegand[a,c]

[a] *Fraunhofer Institute for Telecommunications Heinrich-Hertz-Institut, Image Processing Department, Einsteinufer 37, 10587 Berlin, Germany*
[b] *Eindhoven University of Technology, Signal Processing Systems, PO Box 513, 5600 MB Eindhoven, The Netherlands*
[c] *Image Communication Chair, Technical University of Berlin, Einsteinufer 17, 10587 Berlin, Germany*

ARTICLE INFO

Article history:
Received 7 October 2008
Accepted 19 October 2008

Keywords:
3D television
Multiview video
Depth compression
View synthesis rendering

ABSTRACT

This article investigates the interaction between different techniques for depth compression and view synthesis rendering with multiview video plus scene depth data. Two different approaches for depth coding are compared, namely H.264/MVC, using temporal and inter-view reference images for efficient prediction, and the novel platelet-based coding algorithm, characterized by being adapted to the special characteristics of depth-images. Since depth-images are a 2D representation of the 3D scene geometry, depth-image errors lead to geometry distortions. Therefore, the influence of geometry distortions resulting from coding artifacts is evaluated for both coding approaches in two different ways. First, the variation of 3D surface meshes is analyzed using the Hausdorff distance and second, the distortion is evaluated for 2D view synthesis rendering, where color and depth information are used together to render virtual intermediate camera views of the scene. The results show that—although its rate-distortion (R-D) performance is worse—platelet-based depth coding outperforms H.264, due to improved sharp edge preservation. Therefore, depth coding needs to be evaluated with respect to geometry distortions.

1. Introduction

Multiview video (MVV) representations enable new applications such as free viewpoint video (FVV) and 3D television (3DTV) [17]. The main characteristic of 3DTV is to offer a 3D depth impression of the observed scenery. FVV on the other hand is characterized by providing the user the ability to interactively select an arbitrary viewpoint in the video scene as known from computer graphics. Both technologies do not exclude each other, but rather can be combined into one system. A common characteristic of such technologies is that they use MVV data, where a real world scene is recorded by multiple synchronized cameras.

A popular format for 3DTV uses a conventional monoscopic color video and an associated per pixel depth-image. For this, MPEG specified a standard for efficient compression and transmission [10,11]. In the context of MVV, this format is combined with MVV to form a multiview video+depth (MVD) format, consisting of multiple color videos with associated depth data. Since MVD representations cause a vast amount of data to be stored or transmitted to the user, efficient compression techniques are essential for realizing such applications. Previous work in this field presented various solutions for multiview video coding (MVC), mostly based on H.264 with combined temporal and inter-view prediction, as

* Corresponding author.
E-mail addresses: merkle@hhi.de (P. Merkle), y.morvan@tue.nl (Y. Morvan), smolic@hhi.de (A. Smolic), kmueller@hhi.de (K. Müller), P.H.N.de.With@tue.nl (P.H.N. de With), wiegand@hhi.de (T. Wiegand).

[1] Dirk Farin was with Eindhoven University of Technology, Eindhoven, The Netherlands, when this work was carried out. He is now with Robert Bosch GmbH, Hildesheim, Germany.

0923-5965/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.image.2008.10.010

well as different approaches for depth-image coding, like transform- or wavelet-based depth compression. As a first step towards standardization of technologies for 3DTV and FVV applications, a new standard addressing algorithms for MVV data compression—MVC—is currently developed by the Joint Video Team (JVT) of VCEG and MPEG, which is scheduled to be finalized in 2008. As a second step, MPEG started an *ad hoc* group explicitly on 3D video recently, focusing on FVV and 3DTV systems from a normative point of view, including representation, generation, processing, coding and rendering of MVD data.

Depth-images are a 2D representation of the 3D scene surface and for 3DTV and FVV applications the depth information of MVD data is used for rendering virtual intermediate views of the scene, while for the original camera views only the color information from MVV is sufficient. Therefore, we evaluate the effect of geometry distortions caused by depth-image compression for both 3D scene geometry and rendering quality of MVD data. In this article, we propose a novel coding algorithm that is adapted to special characteristics of depth-images [15], as well as a view synthesis rendering method for evaluating the impact of coding artifacts produced by different coding approaches [13].

This article is organized as follows. Section 2 is about multiview depth compression. Beside H.264-based depth coding, the specific characteristics of depth-images are analyzed and the novel platelet-based depth coding algorithm is introduced. Section 3 is about the 3D scene representation of depth-images and the evaluation of geometry distortions. In Section 4, view synthesis rendering for MVD data is explained, especially its application to evaluate the effects of depth compression. Section 5 presents the results of coding, mesh distance and rendering experiments.

2. Multiview depth video compression

The MVD format consists of several camera sequences of color texture images and associated per sample depth-images or depth maps as illustrated in Fig. 1. Depth-images are a 2D representation of the 3D surface of the scene. Practically, the depth range is restricted to a range between two extremes z_{near} and z_{far}, indicating the minimum and maximum distance, respectively, of the corresponding 3D point from the camera. By quantizing the values in this range, the depth-image in Fig. 1 is specified, resulting in a grayscale image.

A sequence of such depth-images can be converted into a video signal and compressed by any state-of-the-art video codec. Since depth-images represent the scene surface, their characteristics differ from texture images. Encoding depth-images with video codecs that are highly optimized to the statistical properties and human perception of color or texture video sequences, might still be efficient, but result in disturbing artifacts. This requires novel algorithms for depth compression to be developed, that are adapted to these special characteristics. Therefore, after introducing multiview depth video coding based on H.264 in Section 2.1, we evaluate the characteristics of depth video in Section 2.2, which led to the novel platelet-based depth-image coding algorithm, presented in Section 2.3.

2.1. H.264-based depth coding

The most efficient algorithm to date for single-view video compression is H.264/AVC [12,22]. Hence, it typically serves as a starting point for modern MVC, where a 3D scene is captured by a number of cameras. Since some of the cameras share common content, a coding gain can be achieved with multiview coding in comparison to single-view coding, when exploiting the statistical dependencies between the camera views in addition to temporal statistical dependencies within each sequence. Although multi-camera settings in practice range from simple 1D linear and arched arrangement to complex spatial position distribution patterns, the underlying multiview coding structure can still be mapped onto that basic 2D temporal/inter-view array. This approach was followed by multiview compression proposals that finally led to a standardization project for an amendment to H.264/AVC for MVC.

An H.264/MVC coder basically consists of multiple parallelized single-view coders. Therefore, they both use similar temporal coding structures, were a sequence of successive pictures is coded as intra (I), predictive (P) or bipredictive (B) pictures. For I pictures, the content is only predicted from the current picture itself, while P and B picture content is also predicted from other temporal reference pictures. One approach for further improving

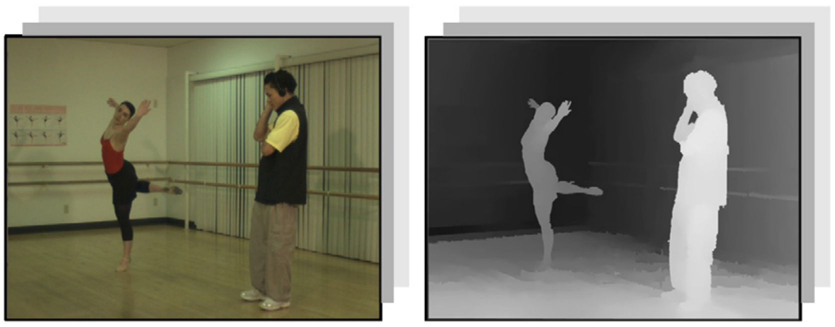


Fig. 1. Example for the MVD format with texture images (left) and corresponding depth-images (right). A typical depth-image frequently contains regions of linear depth changes bounded by sharp discontinuities.

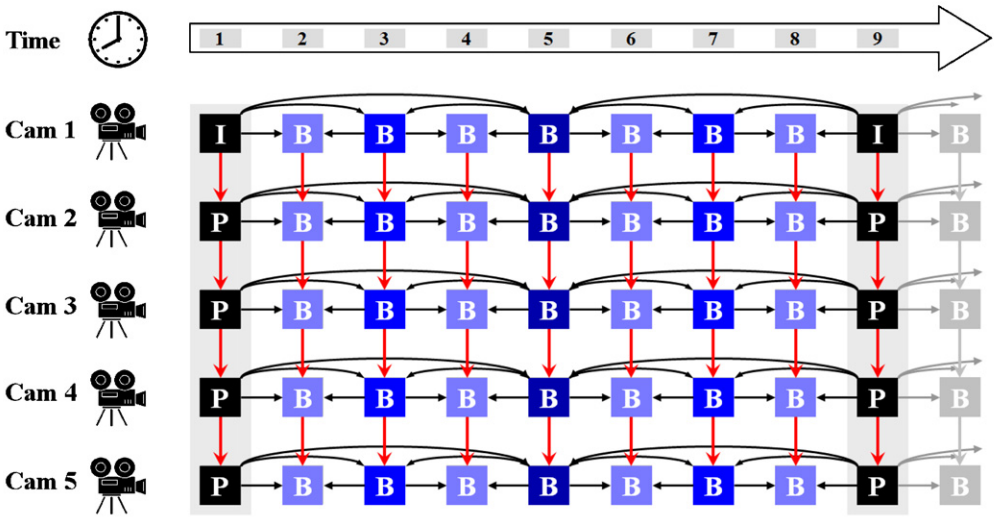


Fig. 2. H.264/MVC coding structure with inter-view prediction (red arrows) and temporal prediction (black arrows) using hierarchical B pictures for 5 cameras and a GOP size of 8 pictures.

coding efficiency is the use of hierarchical B pictures [18], where B pictures can be references for other B pictures and thus creating a B picture hierarchy, as shown for temporal prediction with three hierarchical levels in Fig. 2. Here, the distance between so called key pictures (black) is 8 so that we refer to a group of pictures (GOP) size of 8.

For MVC, the single-view concepts are extended, so that a current picture in the coding process can have temporal as well as inter-view reference pictures for motion-compensated prediction [14]. An example for an H.264/MVC coding structure with five linearly arranged cameras and GOP size of 8 is shown in Fig. 2. This coding structure illustrates how the advantages of hierarchical B pictures are combined with inter-view prediction, without any changes regarding the temporal prediction structure. For view "*Cam* 1", the prediction structure is identical to single-view coding and for the remaining views, inter-view reference pictures are additionally used for prediction (red arrows).

When using H.264 video coding techniques for depth-image compression, multiview sequences of depth-images of an MVD data set are converted into YUV 4:0:0 format video signals. The resulting multiview depth video sequences can then be encoded and decoded like conventional color (or texture) videos, applying either H.264/AVC for simulcast coding or H.264/MVC for multiview coding of a multiview depth video data set.

2.2. *Rate-distortion analysis of piecewise-smooth depth-images*

In this section, considering the smooth properties of depth-images, we propose to approximate, i.e., model, depth-images by piecewise smooth functions. First, we discuss the rate-distortion functions (or rate-distortion behavior) of two different compression algorithms to encode piecewise smooth functions: a transform-based encoder and a tree segmentation-based encoder. Second, comparing both rate-distortion functions, we show that a tree segmentation-based encoder yields a more attractive rate-distortion (R-D) behavior (when compared to a transform-based encoder). As a result, a tree segmentation-based coding algorithm provides a suitable method to encode piecewise smooth depth-images.

The characteristics of depth-images differ from ordinary color images. Since a depth map explicitly captures the 3D structure of a scene, large parts of typical depth-images depict object surfaces. As a result, the input depth-image contains large areas of smoothly changing grey levels. In contrast, at the object boundaries, the depth map exhibits abrupt changes, i.e., sharp edges. Therefore, following these observations, we propose to model depth-images by piecewise smooth functions. The problem of approximating and compressing piecewise smooth signals was first addressed using a wavelet transform. In [1], it was show that the R-D behavior of such a transform-based encoder can be written as

$$D(R) \sim d_1 R^{-2\alpha} + d_3 \sqrt{R}\, 2^{-d_4\sqrt{R}} \tag{1}$$

where R and $D(R)$ correspond to the rate and the rate-distortion functions, respectively. The first term models the R-D behavior of the smooth pieces in the image. This R-D behavior cannot be substantially improved. However, the R-D behavior at the signal-discontinuity (the second term in the previous equation) can be further improved.

More specifically, the R-D behavior can be improved by using a tree segmentation-based coding algorithm. A tree segmentation-based coding algorithm recursively decomposes the input image into blocks of variable sizes and approximates each block by a selected modeling function. In [20], it was mathematically shown that the R-D behavior of a tree segmentation-based coding algorithm decays as

$$D(R) \sim c_4 2^{-c_5 R} \tag{2}$$

Comparing both previously introduced rate-distortion functions, it can be inferred that the R-D function of a tree segmentation-based coding algorithm features a faster decaying R-D behavior (when compared to the R-D function of a transform-based encoder). Note that this result holds for piecewise smooth images. Therefore, a coding gain can be obtained by encoding piecewise-smooth depth-images by using a tree segmentation-based coding algorithm. As a first step toward tree segmentation-based *coding* algorithms, several tree segmentation-based *reconstruction* algorithms have been proposed. These algorithms can be classified by the modeling function used to approximate each image block in the image. One modeling function, the "Wedgelet" [6] function, is defined as two piecewise-constant functions separated by a straight line. This concept was originally introduced as a mean of detecting and reconstructing edges from noisy images, and was later extended to piecewise-linear functions, called "Platelet" [23] functions. In the following section, we focus on the compression of piecewise-smooth depth-images using the previously introduced Platelet functions.

2.3. Platelet-based depth coding

Considering our compression framework, we adopt the "Wedgelet" and "Platelet" signal decomposition method. In this way, we follow the idea developed in [20] to encode texture images using piecewise polynomials as modeling functions. The proposed algorithm models smooth regions of the depth-images using piecewise-linear functions separated by straight lines. To define the area of support for each modeling function, we employ a quadtree decomposition that divides the image into blocks of variable size, each block being approximated by one modeling function containing one or two surfaces. The quadtree decomposition and the selection of modeling function are then optimized such that a global rate-distortion trade-off is realized.

2.3.1. Depth-image modeling

In this section, we present a novel approach for depth-image coding using the piecewise-linear functions previously mentioned. The idea followed is to approximate the image content using modeling functions. In our framework, we use two classes of modeling functions: a class of piecewise-constant functions and a class of piecewise-linear functions. For example, regions of constant depth (e.g. flat unslanted surfaces) show smooth regions in the depth-image and can be approximated by a piecewise-constant function. Secondly, planar surfaces of the scene, like the ground plane and walls, appear as regions of gradually changing grey levels in the depth-image. Hence, such a planar region can be approximated by a single linear function. To specify the 2D-support of the modeling functions in the image, we employ a quadtree decomposition that hierarchically divides the image into blocks, i.e., nodes of different size. In some cases, the depth-image within one block can be approximated with one modeling function. If no suitable approximation can be determined for the block, it is subdivided into four smaller blocks. To prevent that many small blocks are required along a diagonal discontinuity, we divide the block into two regions separated by a freely placed, slanted line. Each of these two regions is coded with an independent function. Consequently, the coding algorithm chooses between four modeling functions for each leaf in the quadtree:

- *Modeling function* $\hat{f}_1$: Approximate the block content with a constant function.
- *Modeling function* $\hat{f}_2$: Approximate the block content with a linear function.
- *Modeling function* $\hat{f}_3$: Subdivide the block into two regions A and B separated by a straight line and approximate each region with a constant function (a wedgelet function).

$$\hat{f}_3(x,y) = \begin{cases} \hat{f}_{3A}(x,y) = \gamma_{0A} & (x,y) \in A \\ \hat{f}_{3B}(x,y) = \gamma_{0B} & (x,y) \in B \end{cases}$$

- *Modeling function* $\hat{f}_4$: Subdivide the block into two regions A and B separated by a straight line and approximate each region with a linear function (a platelet function).

$$\hat{f}_4(x,y) = \begin{cases} \hat{f}_{4A}(x,y) = \theta_{0A} + \theta_{1A}x + \theta_{2A}y & (x,y) \in A \\ \hat{f}_{4B}(x,y) = \theta_{0B} + \theta_{1B}x + \theta_{2B}y & (x,y) \in B \end{cases}$$

2.3.1.1. Estimation of model coefficients. The objective of this processing step is to provide the model coefficients that minimize the approximation error between the original depth signal in a block and the corresponding approximation.

For $\hat{f}_1$, only one coefficient α_0 has to be computed. Practically, the coefficient α_0, that minimizes the error between f and $\hat{f}_1$, simply corresponds to the mean value of the original data. Secondly, it was indicated earlier that we use a linear function $\hat{f}_2$ to approximate blocks that contain a gradient. In order to determine the three coefficients β_0, β_1 and β_2 of the linear function $\hat{f}_2(x,y) = \beta_0+\beta_1 x+\beta_2 y$, a least-squares optimization is used. For the wedgelet $\hat{f}_3$ and platelet $\hat{f}_4$ functions, we have to determine not only the model coefficients, but also the separating line. For this reason, the coefficient estimation is initialized by testing every possible line that divides the block into two areas. This step provides a candidate subdivision line and two candidate regions A and B. Subsequently, the wedgelet and platelet coefficients are computed over the candidate regions using the average pixel value and a least-squares minimization, respectively.

The decision for each modeling function is based on a rate-distortion decision criterion that is described in the following section.

2.3.2. R-D optimized bit-allocation

In this section, we aim at providing details about the bit-allocation strategy that optimizes the coding in a rate-distortion sense. Considering our lossy encoder/decoder framework, our aim is to optimize the compression of a given image to satisfy a R-D constraint. In our case, there are three parameters that influence this trade-off: (1) the selection of modeling functions, (2) the quadtree decomposition and (3) the quantization step-size of the modeling-function coefficients. Thus, the problem is to adjust each of the previous parameters such that the objective R-D constraint is satisfied. The three aspects will be individually addressed below.

To optimize these three parameters in an R-D sense, the adopted approach is to define a cost function that combines both rate R_i and distortion D_i of the image i. Typically, the Lagrangian cost function

$$J(R_i) = D_i(R_i) + \lambda R_i \tag{3}$$

is used, where R_i and D_i represent the rate and distortion of the image, respectively, and λ is a weighting factor that controls the R-D trade-off. Using the above Lagrangian cost function principle, the algorithm successively performs three independent parameters optimizations: (1) an independent selection of modeling functions; (2) a quadtree structure optimization and (3) the quantizer step-size selection. Let us address all three aspects now.

2.3.2.1. Modeling function selection. First, we assume that an optimal quadtree segmentation and quantizer step-size is provided. Since the rate and distortion are additive functions over all blocks, an independent optimization can be performed within the blocks. Therefore, for each block, the algorithm selects the modeling function that leads to the minimum coding cost. More formally, for each block, the algorithm selects the best modeling function $\tilde{f}$ in an R-D sense according to

$$\tilde{f} = \underset{\hat{f}_j \in \{\hat{f}_1 \hat{f}_2 \hat{f}_3 \hat{f}_4\}}{\operatorname{argmin}} (D_m(\hat{f}_j) + \lambda R_m(\hat{f}_j)), \tag{4}$$

where $R_m(\hat{f}_j)$ and $D_m(\hat{f}_j)$ represent the rate and distortion resulting from using one modeling function $\hat{f}_j$, respectively.

2.3.2.2. Quadtree decomposition. To obtain an optimal quadtree decomposition of the image, a well-known approach is to perform a so-called *bottom-up* tree-pruning technique [5]. The algorithm can be described as follows. Consider four children nodes denoted by N_1, N_2, N_3 and N_4 that have a common parent node which is represented by N_0. For each node k, a Lagrangian coding cost $(D_{N_k} + \lambda R_{N_k}) k \in \{0,1,2,3,4\}$ can be calculated. Using the Lagrangian cost function, the four children nodes should be pruned whenever the sum of the four coding cost functions is higher than the cost function of the parent node. When the children nodes are not pruned, the algorithm assigns the sum of the coding costs of the children nodes to the parent node. Subsequently, this tree-pruning technique is recursively performed in a bottom-up fashion. It has been proved [5] that such a bottom-up tree-pruning leads to an optimally pruned tree, thus in our case to an R-D optimal quadtree decomposition of the image.

2.3.2.3. Quantizer selection. So far, we have assumed that the coefficients of the modeling functions are scalar quantized prior to model selection and tree-pruning. However, no detail has been provided about the selection of an appropriate quantizer. Therefore, the problem is to select the optimal quantizer, denoted $\tilde{q}$, that meets the desired R-D constraint. We propose to select the quantizer $\tilde{q}$ out of a given set of possible scalar quantizers $\{q_2, \dots q_8\}$, operating at 2–8 bits per level, respectively. To optimally select the quantizer, we re-use the application of the Lagrangian cost function and select the quantizer $\tilde{q}$ that minimizes the Lagrangian coding cost of the image

$$\tilde{q} = \underset{q_l \in \{q_2, \dots, q_8\}}{\operatorname{argmin}} D_i(R_i, q_l) + \lambda R_i(q_l) \tag{5}$$

Here, $R_i(q_l)$ and $D_i(R_i, q_l)$ correspond to the global rate R_i and distortion $D_i(R_i)$ in which the parameter q_l is added to represent the quantizer selection. To solve the optimization problem of Eq. (5), the image is encoded using all possible quantizers and the quantizer $\tilde{q}$ that yields the lowest coding cost $J_i(R_i, \tilde{q}) = D_i(R_i, \tilde{q}) + \lambda R_i(\tilde{q})$ is selected. Finally, the parameter λ, that controls the desired R-D trade-off, has to be determined. To calculate λ, a well-known approach is to perform a bisection search of the parameter, such that λ yields the highest image-quality for a specific bitrate [16]. Practically, the values for λ range from 20 to 1000, increasing in a logarithmic fashion.

2.3.3. *Predictive coding of parameters*

In this section, the entropy coding of our coding algorithm is described.

2.3.3.1. Quadtree structure. The quadtree is transmitted top-down, where for each node, a binary flag indicates whether this node is subdivided or not. This decision is coded with an arithmetic encoder with fixed probabilities. The probability that the node is subdivided is depending on the number of neighboring nodes at the same tree-level that are also subdivided. The neighborhood context contains the left, the top-left, and the neighboring top block, such that the number of subdivided neighbors is between 0 and 3. In the example of Fig. 3, the number of subdivided neighbors is two.

2.3.3.2. Coding mode. The coding mode for each block in the quadtree is coded with an input-adaptive arithmetic coder. Fixed probabilities are not suitable here, since the selection of coding modes depends on the bitrate (more complex models for higher bitrates).

2.3.3.3. DC coefficients. The DC coefficients of all input blocks are highly correlated. For this reason, we predict their value from previously coded blocks and only code the residual value. More specifically, we consider the three blocks that are adjacent to the top-left pixel in the current block. The predictor is formed by the median of the DC coefficients of these three blocks. If a neighboring block is subdivided into two models and hence has two DC

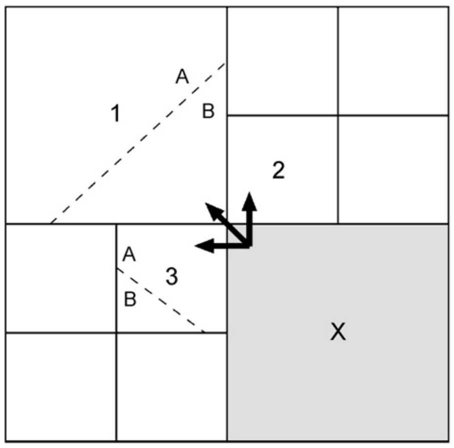


Fig. 3. Illustration for parameter encoding. The grey block X is currently being coded. DC-predictors are formed from the three numbered blocks (median of DC-coefficient of 1-B, 2, and 3-A).

coefficients, we use the DC-coefficient of the region that is adjacent to the top-left pixel of the current block. If the current block is represented with two functions, the predictor is used for both. Note that this scheme always works properly, whatever coding modes have been used for these blocks. In the example in Fig. 3, the DC coefficients of the regions 1-B, 2, and 3-A would be used to build the predictor for block X. The distribution of the residual signal after prediction subtraction is non-uniform. Consequently, coding of the residual value is carried out with an input-adaptive arithmetic encoder.

2.3.3.4. AC coefficients. The AC parameter corresponds to the first-order parameters of the functions $\hat{f}_2$ and $\hat{f}_4$. For $\hat{f}_4$, this means θ_{1A}, θ_{2A}, θ_{1B} and θ_{2B}. These four parameters and the two parameters for function $\hat{f}_2$ are non-uniformly distributed. This implies the use of a variable-length coder. Therefore, the four parameters were coded with an input-adaptive arithmetic encoder as well.

2.3.4. Algorithm summary

The algorithm requires as an input the depth-image and a weighting factor λ that controls the R-D trade-off. The image is first recursively subdivided into a full quadtree decomposition. All nodes of the tree are then approximated by the four modeling functions $\hat{f}_1, \hat{f}_2, \hat{f}_3, \hat{f}_4$. In a second step, the coefficients of the modeling functions are quantized using one scalar quantizer q_l. For each node of the full tree, an optimal modeling function $\tilde{f}$ can now be selected using Eq. (4). Employing the tree-pruning technique described in Section 2.3.2, the full tree is then pruned in a bottom-up fashion. The second step of the algorithm (i.e. the coefficient quantization) is repeated for all quantizers $q_l \in \{q_2,\ldots,q_8\}$ and the quantization that leads to the lowest global coding cost is selected (see Eq. (5)). Subsequently, for each leaf of the tree, the quantized zero-order coefficients are predicted using the neighboring values as explained in Fig. 3 and because the residual values are non-uniformly distributed, they are coded arithmetically. Note that the first-order coefficients satisfy a Laplacian distribution. If it would be a fixed Laplacian distribution, we could have "simply" used a fixed probability table. However, we do not know the exponential coefficient value. For this reason, we encode these coefficients using an adaptive arithmetic encoder.

Note that the R-D optimization is based on fixed-length codes to enable a fast computation. The final bitrate is lower since the parameters are further compressed with an arithmetic encoder. The prediction scheme presented in the previous section together with subsequent arithmetic encoding yields 0.5–1.5 dB of gain for a fixed bitrate.

3. 3D scene surface representation

This section discusses the relation between depth-images and 3D scene geometry as well as the evaluation of geometry distortions caused by depth coding artifacts.

The depth-images that come with MVD data are a 2D representation of the 3D scene surface from the corresponding camera perspective. Therefore, a depth-image can be transformed into a 3D representation via perspective projection [7,8]. For a camera $\boldsymbol{C}$ the relation between a point $\bar{X}$ in 3D scene coordinates and a point $\bar{x}$ in camera coordinates can be described as follows.

$$\begin{aligned} \bar{x} &= \bar{\boldsymbol{P}}\bar{X} \\ \bar{X} &= \bar{\boldsymbol{P}}^{-1}\bar{x} \end{aligned} \tag{6}$$

with $\bar{x} = (x,y,z,1)^T$, $\bar{X} = (X,Y,Z,1)^T$. $\bar{\boldsymbol{P}}$ is the 4×4 projection matrix, which is defined by the camera extrinsic and intrinsic parameters as

$$\bar{\boldsymbol{P}} = \bar{\boldsymbol{K}}\bar{\boldsymbol{E}} = \bar{\boldsymbol{K}}[\bar{\boldsymbol{R}}|\bar{t}]. \tag{7}$$

Here $\bar{\boldsymbol{K}}$ is the 4×4 intrinsic and $\bar{\boldsymbol{E}}$ is the 4×4 extrinsic matrix, containing the rotation matrix $\bar{\boldsymbol{M}}$ and the translation vector $\bar{t}$ of $\boldsymbol{C}$ relative to the world coordinate system. $\bar{\boldsymbol{K}}$ and $\bar{\boldsymbol{E}}$ contain the intrinsic and extrinsic parameters that are obtained from camera calibration [21]. Camera calibration represents a fundamental element for capturing MVV sequences besides further processing steps like synchronization or color balance of the multiple cameras. For a depth-image the continuous camera coordinates of $\bar{x}$ are transformed into image coordinates, resulting in the discrete pixel position (u,v) and the assigned depth value d. Unlike the pixel position, the depth value is logarithmically quantized to keep more details in foreground regions of the 3D scene.

By projecting all pixels of a depth-image into 3D scene space, a point cloud is generated, consisting of regularly sampled feature points of the 3D scene surface. In the field of computer graphics, these 3D scene points are also known as vertices and rendering techniques from computer graphics can be used to render such point clouds. However, mesh representations are much more frequently used in computer graphics, thus software and hardware for rendering is highly optimized for processing meshes. We now describe how connectivity is added to the vertices of our depth-image-based point cloud, resulting in a 3D surface mesh representation. Typically, such meshes consist of triangle primitives. In the case of depth-images the connectivity can easily be generated from the pixel raster, but two restrictions should be observed: first, the number of triangles should be kept small for efficient storage, processing and rendering of the mesh and second, foreground objects should not be connected with the background along their edges to avoid spanning the surface over unconnected regions of the scene. The algorithm for mesh construction starts with dividing the depth-image into blocks of 32×32 pixel size and generates the mesh for each of these in an iterative refinement process. If all pixels of a block have the same depth value, the corresponding triangles are coplanar, the iteration stops and the vertices of the four corner pixels can be connected as two triangles. Otherwise, the block is subdivided into four sub-blocks and each one is again checked for coplanarity. If this recursion reaches the minimum block size of 2×2 pixels all triangles are added to the mesh connectivity, except those with a depth value difference above a certain threshold. This keeps foreground objects separated from the background. Appropriate values for the threshold are relatively low (e.g. a depth value difference of 5 for 8-bit depth pixels), in order to ensure a reliable separation. Note, that for logarithmically quantized depth values a certain threshold represents different distances in 3D scene space.

Such 3D surface meshes generated from depth-images are the basis for an evaluation of the geometry distortions caused by depth coding artifacts. The depth coding algorithms presented in Section 2 lead to distortions or artifacts in the compressed depth-images, where the depth value d of certain pixels differs from the original value. Projecting distorted depth-images into the 3D scene space leads to geometry distortions. Consequently, the 3D surface mesh from a compressed depth-image differs from the original undistorted one. Evaluating the impact of coding artifacts in terms of geometry distortions means to measure the distance between these two meshes. For this purpose we employ the method described in [9]. This method measures the error between surfaces using the Hausdorff distance and is well-known from quality evaluation of mesh compression techniques. Due to the described correlation between depth-images and surface meshes, depth-image coding can be seen as a special case of mesh compression and consequently this method is applicable to evaluate the geometry distortions caused by depth-image coding artifacts.

4. View synthesis rendering

Applications like 3DTV and FVV are realized from MVV plus depth via view synthesis rendering. View synthesis rendering is a technique for generating arbitrary intermediate views from a 3D representation of the scene [3,4]. The effect of depth compression needs to be evaluated with respect to the quality of rendered virtual views, because depth information is only useful for generating intermediate camera views, but not for showing the original camera view. This section first explains view synthesis rendering and second defines a quality measure for virtual views rendered with compressed depth.

4.1. Virtual camera, geometry and texture blending

The basic concept of view synthesis rendering with MVD data is to use pairs of neighboring original camera views in order to render arbitrary virtual views on a specified camera path between them. Here rendering means to map or transform 3D information into 2D by applying camera projection. As described in Section 3 for original camera views, the geometrical relation between points in 3D scene space and 2D image space is defined by the projection matrix, which is derived from the camera calibration parameters. Therefore, the initial step for view synthesis rendering is to define a virtual camera $\boldsymbol{C_V}$. Given two original cameras $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ with their extrinsic matrices $\bar{\boldsymbol{E}}_1$ and $\bar{\boldsymbol{E}}_2$, the extrinsic matrix $\bar{\boldsymbol{E}}_V$ of the virtual camera is derived as follows. First, the position of the virtual camera is calculated by linear interpolation

$$\bar{t}_V = (1-\alpha)\,\bar{t}_1 + \alpha\,\bar{t}_2 \quad \alpha \in [0,1], \tag{8}$$

where α is the virtual view weighting factor and $\bar{t}_1$ and $\bar{t}_2$ are the translation vectors of $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$. By varying α between 0 and 1 any position on the linear camera path between $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ can be defined for $\boldsymbol{C_V}$. Beside the translation vector an extrinsic matrix contains a rotation matrix $\bar{\boldsymbol{R}} = [\bar{\boldsymbol{r}}_x\ \bar{\boldsymbol{r}}_y\ \bar{\boldsymbol{r}}_z]$. Unlike the translation, $\bar{\boldsymbol{R}}_V$ needs to be calculated by spherical linear interpolation (Slerp) [19]. Spherical linear interpolation originates from computer graphics in the context of quaternion interpolation and has the characteristic that the resulting coordinate system of the virtual camera remains orthonormal. As an example, the x coordinate rotation vector $\bar{\boldsymbol{r}}_{Vx}$ is calculated from the two original cameras' x coordinate rotation vectors $\bar{\boldsymbol{r}}_{1x}$ and $\bar{\boldsymbol{r}}_{2x}$, using the same virtual view weighting factor α as for translation.

$$\bar{r}_{Vx} = \frac{\sin((1-\alpha)\,\theta_x)}{\sin(\theta_x)}\,\bar{r}_{1x} + \frac{\sin(\alpha\,\theta_x)}{\sin(\theta_x)}\,\bar{r}_{2x}, \tag{9}$$

where θ_x is the angle between $\bar{\boldsymbol{r}}_{1x}$ and $\bar{\boldsymbol{r}}_{2x}$. The y and z coordinate rotation vectors are then calculated accordingly. Thus our virtual camera is derived from the two original cameras, with extrinsic matrix $\bar{\boldsymbol{E}}_V$ for translation and rotation. For the used datasets, the intrinsic matrix $\bar{\boldsymbol{K}}$ is the same for all original and virtual cameras such that no interpolation is required.

The second step for view synthesis rendering is to project the input depth maps of $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ into 3D scene space. According to the description in Section 3 the inverse projection matrix is used to calculate the 3D vertex position for each pixel in the depth map, resulting in a 3D point cloud for $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ that represents the scene surface from the two original cameras' perspectives. Since the MVD representation uses color textures with associated per pixel depth maps, the value of the corresponding texture color pixel can be assigned to each of the 3D vertices. The result of this operation is a so-called particle cloud for $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$, where each particle consists of a 3D vertex and its color.

The third step for view synthesis rendering is to individually project the particle cloud of each camera into the virtual camera view $\boldsymbol{C_V}$ with the projection matrix $\bar{\boldsymbol{P}}_V$ calculated from the extrinsic and intrinsic matrix derived in the first step. To avoid overwriting foreground with background information, a depth buffer is used here. The result of this operation are two pairs of depth with associated texture color images with the same resolution as the original images of $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$, but showing the scene from the perspective of $\boldsymbol{C_V}$. Due to pixel quantization in the last step of the projection, rounding error artifacts in terms of small holes of one pixel size occur in the resulting images. These artifacts are reduced by applying an adaptive filtering technique that detects these rounding error artifacts and iteratively fills them with the information of weighted neighboring pixels.

The fourth step for view synthesis rendering is texture blending, which basically means to overlay the two texture images into one output image. We apply a technique called view-dependent texture blending, where depth and perspective information is used to render a consistent color output for synthesized views along the camera path between the original cameras. For each pixel T_V in the texture color output image the depth information of the corresponding pixel in both projected depth-images is analyzed, resulting in three possible scenarios:

- Both depth pixels have a zero value, indicating that due to disocclusion no depth information is present at this position. These output pixels remain empty and are a matter of post-processing.

  $$T_V(u,v) = 0$$

- One of the depth pixels has a non-zero value. This is the typical disocclusion case, where background areas of the scene can be seen from one original camera perspective, but are occluded by foreground objects in the other. In this case the output pixel is filled with the texture color of the non-zero depth pixel.

$$T_V(u,v) = \begin{cases} T_1(u,v) & d_1(u,v)\neq 0, d_2(u,v) = 0 \\ T_2(u,v) & d_1(u,v) = 0, d_2(u,v)\neq 0 \end{cases}$$

- Both depth pixels have a non-zero value. This is the most frequent case with view synthesis rendering and the two corresponding texture color values are mixed by using the virtual view weighting factor α for view-dependent texture blending.

$$T_V(u,v) = (1-\alpha)T_1(u,v) + \alpha T_2(u,v)$$

Altogether the described view synthesis rendering algorithm allows the rendering of any virtual camera view of the 3D scene, with smoothly blended textures, along the path between the two original input camera views by simply varying the virtual view weighting factor α.

4.2. Rendering quality evaluation

For classical video coding the quality of different approaches is usually evaluated as the R-D performance by comparing compressed with uncompressed versions of the input data. Since depth-images are a 2D representation of the 3D scene surface they are utilized for rendering virtual intermediate views, while at original camera views the original texture contains all the information. Therefore, the effect of depth coding artifacts needs to be evaluated further with respect to the rendering quality of virtual camera views. Based on the view synthesis rendering algorithm described in the previous section we introduce a quality measure for virtual intermediate views (Fig. 4).

The impact of compression on video sequences is measured by calculating the PSNR for the compressed and thereby distorted picture with respect to the original uncompressed picture. This approach can be adapted to view synthesis rendering with compressed depth data as depicted in Fig. 5. Two original cameras (black) are used to generate a virtual camera view (white). The problem with evaluating the effect of depth coding for synthesized views is the lack of a ground truth to compare the results with. Therefore the corresponding synthesized view from

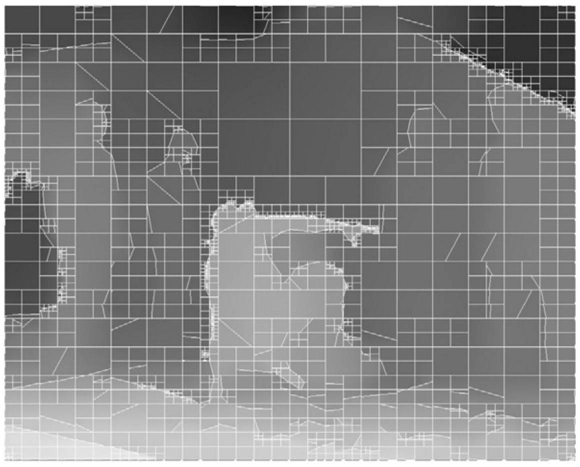


Fig. 4. Platelet-based depth-image compression and decomposition ("Breakdancers" sequence at 0.05 bits per pixel with a PSNR of 42.7 dB).

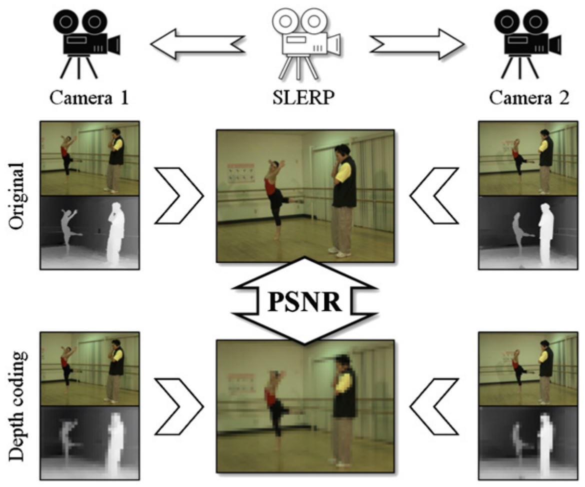
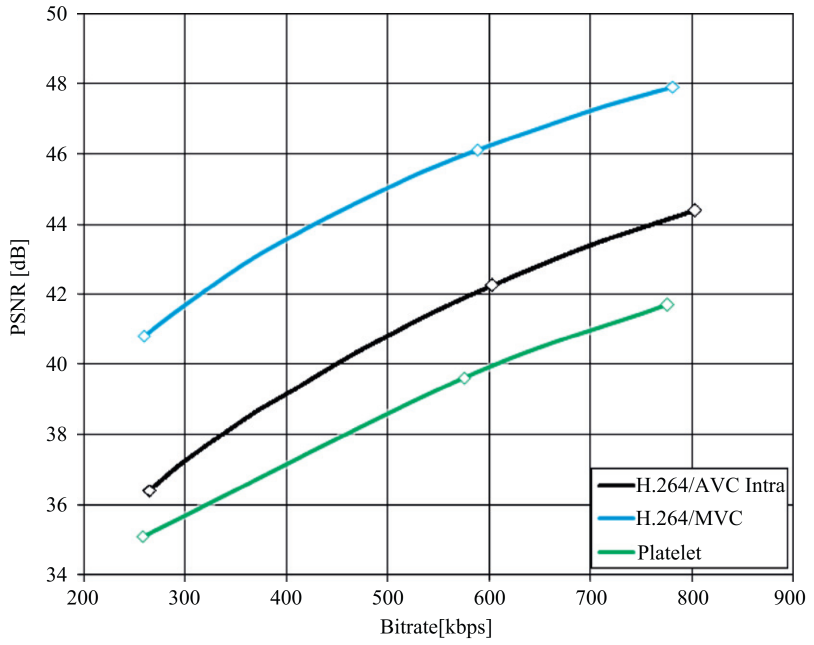


Fig. 5. Evaluation of view synthesis rendering quality with compressed depth video.

uncoded data is defined as a reference (Fig. 5, top). The reference is rendered from uncompressed original depths and original textures. In a second step, the same view is rendered from the compressed depth-images with original textures (Fig. 5, bottom). This view is distorted by the depth coding artifacts and their impact on the rendering quality can now be analyzed by comparing the reference picture with the distorted one in terms of objective and subjective quality. For objective evaluation the final step is to calculate the PSNR between the two images for view synthesis rendering with and without compressed depth data.

5. Experimental results

We conducted the experiments for two MVD test data sets named "Breakdancers" and "Ballet", both consisting of eight linearly arranged camera views. From both test data sets the first 25 frames were used.



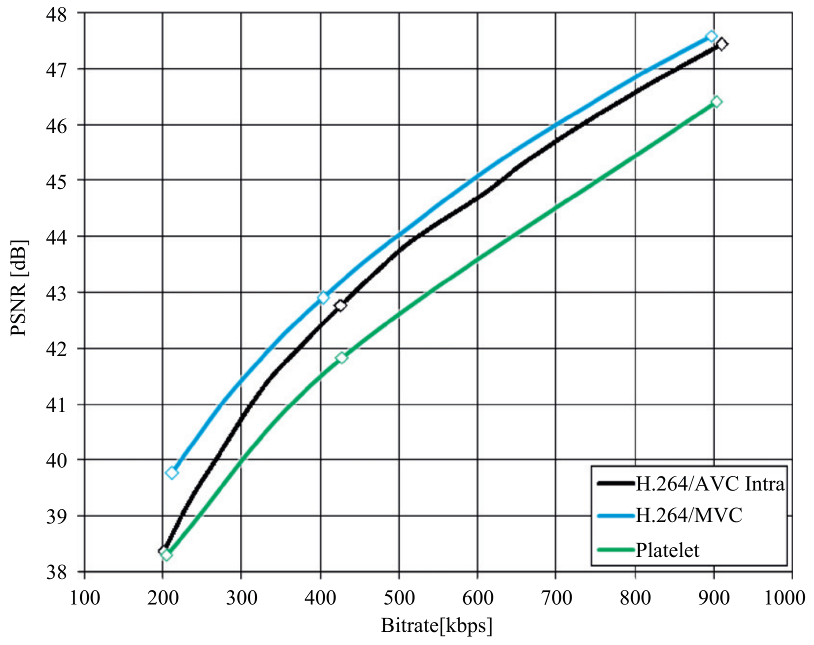


Fig. 6. R-D curves for multiview depth video compression with H.264/AVC Intra, H.264/MVC and Platelet-based coding. Average per camera bitrate vs. average PSNR(Y) for 8 cameras and 25 frames of "Ballet" (top) and "Breakdancers" (bottom) sequence.

5.1. Depth coding results

For the coding experiments the depth sequences were compressed for each camera view at different qualities. First these multiview depth videos were encoded and decoded with a standard conforming H.264/AVC coder as I pictures. This coding method is used as a reference for the two advanced coding algorithms. In a second experiment the same multiview depth video test data sets were encoded and decoded using H.264/MVC with typical settings for MVC, like variable block size, a search range of ± 96, CABAC enabled and rate control via Lagrangian techniques. The third experiment used the Platelet-based coder presented in Section 2.3. These three experiments allow us to compare the properties of the different approaches regarding bitrate efficiency as well as distortion and coding artifacts.

Fig. 6 shows the depth compression results in terms of R-D performance. H.264/MVC clearly outperforms H.264/AVC Intra as well as Platelet-based depth coding, because of additionally utilizing both temporal and inter-view reference pictures for prediction. Note, that the current implementation of the Platelet-based coder only uses intra prediction. Accordingly reference coding was done with H.264/AVC restricted to intra prediction. Except for very low bitrates H.264/AVC Intra coding outperforms the Platelet-based coding approach in terms of PSNR performance. The reason is that H.264 as an international video coding standard is optimized for bitrate efficiency to a great extent. Comparing H.264/MVC to the reference results clearly indicates the advantages video coding takes of using temporal and inter-view reference pictures for motion-compensated prediction. Especially for “Ballet” where a coding gain of 4 dB is achieved. In addition to these objective results Fig. 7 shows examples for the subjective quality at a very low bitrate, highlighting the typical coding artifacts for the two evaluated coding algorithms, namely blurring and ringing for H.264 and angular shaped edges and flattened surfaces for Platelet-based.

5.2. Mesh distance results

The mesh distance experiments were realized with a tool named MESH [2], which is a fast implementation of the Hausdorff distance measure approach described in Section 3. For the experiments we used the decoded depth-images from the coding experiments presented in the previous section. From these coding results three ratepoints for low, medium and high quality were chosen, which are indicated by the markers in the diagrams of Fig. 6. They are selected from the range of R-D coding results as typical representatives—one for the upper and one for the lower range limit plus one intermediate. Due to the different content and thereby R-D performance the values of the ratepoints vary for different test sequences.

The geometrical distortion is measured as the average RMS value of the Hausdorff distance over the complete mesh surface between the meshes from a coded and the corresponding uncoded depth-image. Therefore a lower value indicates that depth-image coding artifacts produce less geometry distortion of the mesh surface, with 0 indicating identical surfaces. Because the Hausdorff distance is in general not symmetrical, meaning that the distance from mesh A to mesh B is unequal to the distance from mesh B to mesh A, both distances are calculated and the higher one is defined as the symmetrical Hausdorff distance.

The diagrams in Fig. 8 shows the symmetrical Hausdorff distance over the coding bitrate as the average value for all 25 frames of all eight cameras. Comparing these results to the R-D performance for depth-image coding in Fig. 6 leads to the conclusion that Platelet-based coding performs better in terms of geometry distortion. Especially for the “Breakdancers” sequence Platelet-based coding has about half the Hausdorff distance of the two H.264-based coding methods. Although Platelet-based coding is outperformed by H.264/MVC for middle and high bitrates of the “Ballet” sequence in absolute values, the difference in Hausdorff distance between both

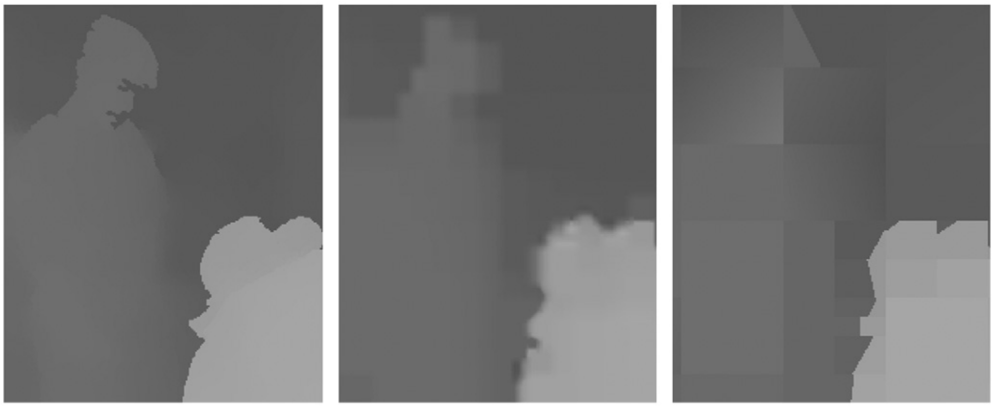
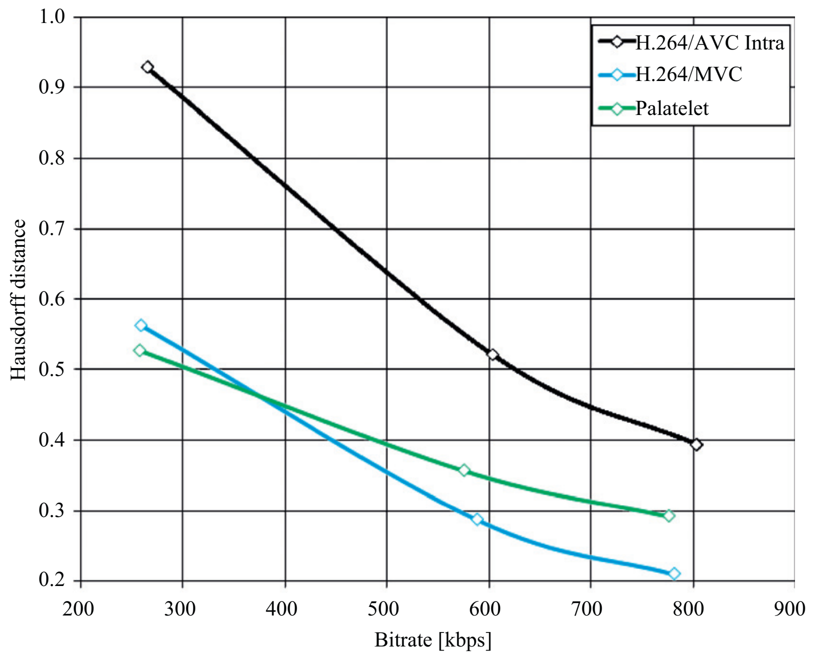


Fig. 7. Examples for typical depth coding artifacts: Original uncompressed (left), H.264-based (middle) and Platelet-based (right).



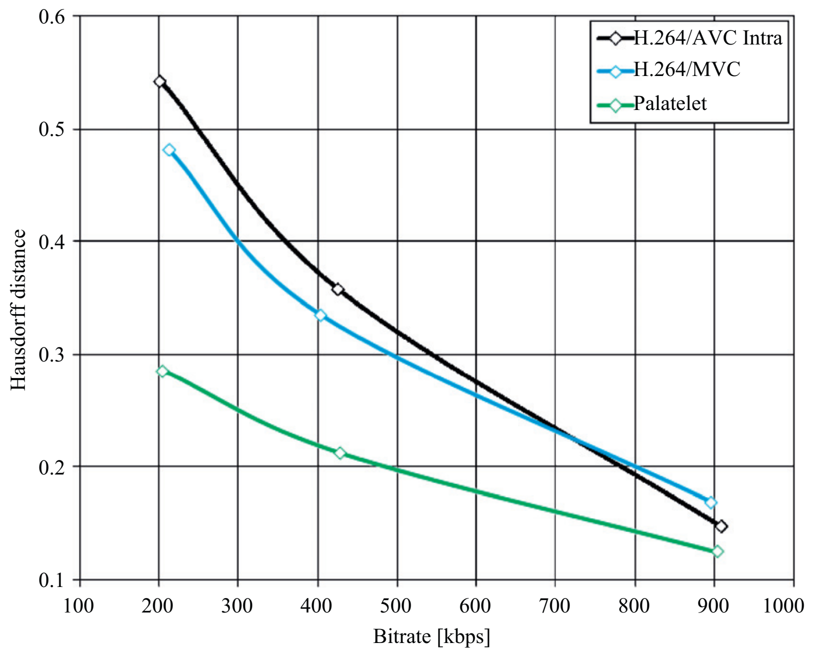


Fig. 8. Symmetrical Hausdorff distance between meshes of compressed and uncompressed depth-images for H.264/AVC Intra, H.254/MVC and Platelet-based coding at low, middle and high bitrates, averaged over 8 cameras and 25 frames of "Ballet" (top) and "Breakdancers" (bottom) sequence.

methods is surprisingly small with respect to the large difference in coding PSNR values. This evaluation highlights, that the typical coding artifacts of H.264-based coding lead to considerably higher geometry distortions than those of Platelet-based coding.

5.3. *Rendering results*

In this section we present the results of the view synthesis rendering quality experiments as described in Section 4.2. For this purpose we used the depth coding results presented in Section 5.1 with the three ratepoints for low, medium and high quality. Note that the markers in the diagrams of Fig. 6 correspond to these ratepoints. For each ratepoint a series of virtual views was rendered along the camera path of each pair of neighboring cameras for each of the three coding approaches, using a virtual view weighting factor α step-size of 0.1. This was done for each timepoint of the 25 picture long sequences.

Fig. 9 shows the results of the experiments on view synthesis rendering quality with compressed depth data. Each diagram contains the results for one ratepoint of one MVD test sequence. The horizontal axis represents the virtual camera path along the original cameras, while the vertical axis shows the resulting PSNR(Y) of the three

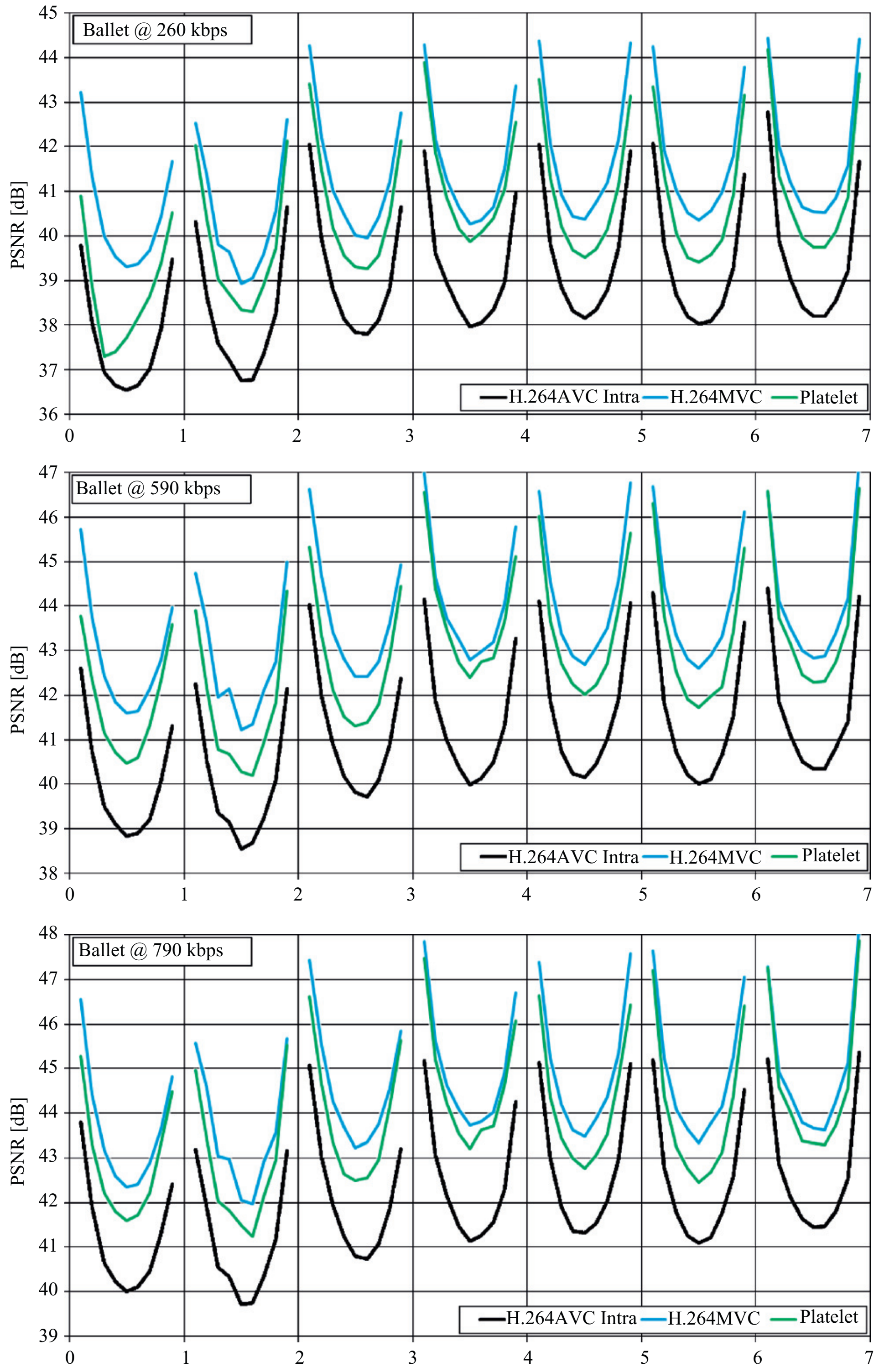
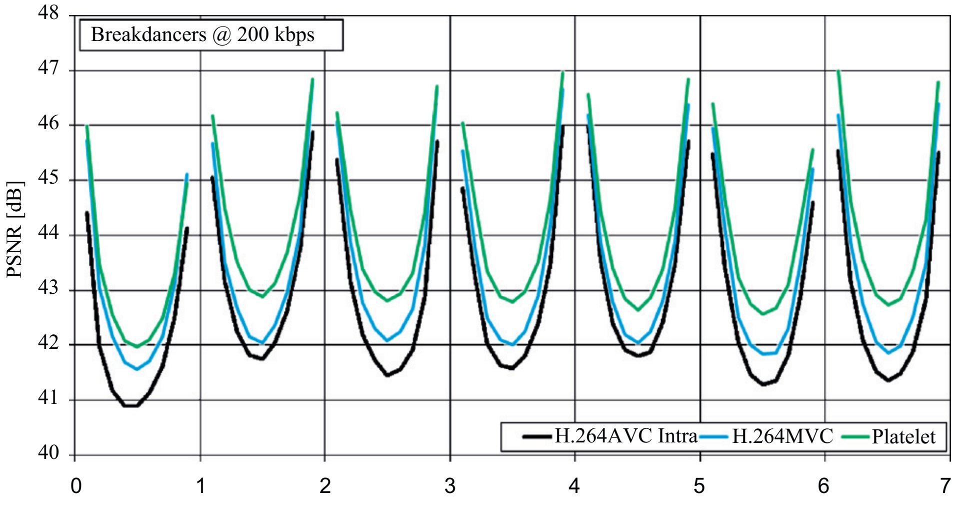
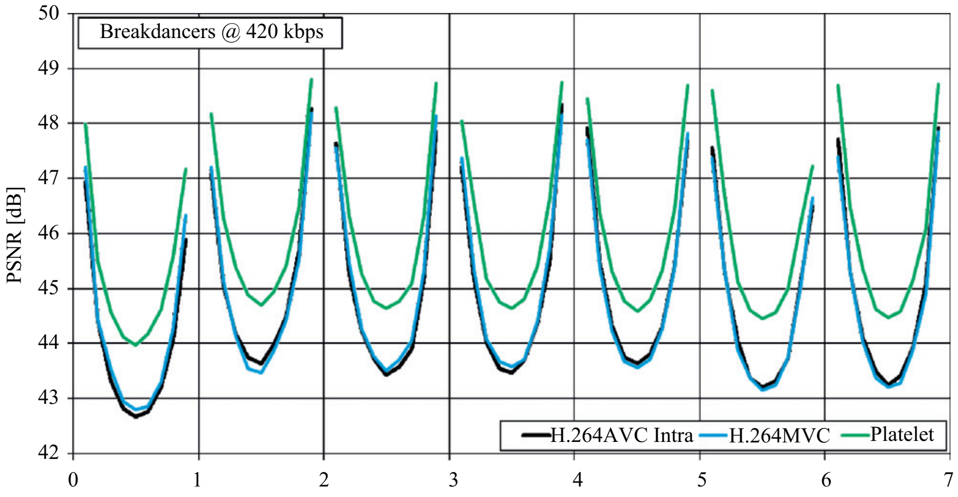


Fig. 9. View synthesis rendering quality with H.264/AVC Intra, H.264/MVC and Platelet-based compressed multiview depth video at low, middle and high bitrates. Camera position vs. average PSNR(Y) for 8 cameras and 25 frames of "Ballet" (top) and "Breakdancers" (bottom) sequence.





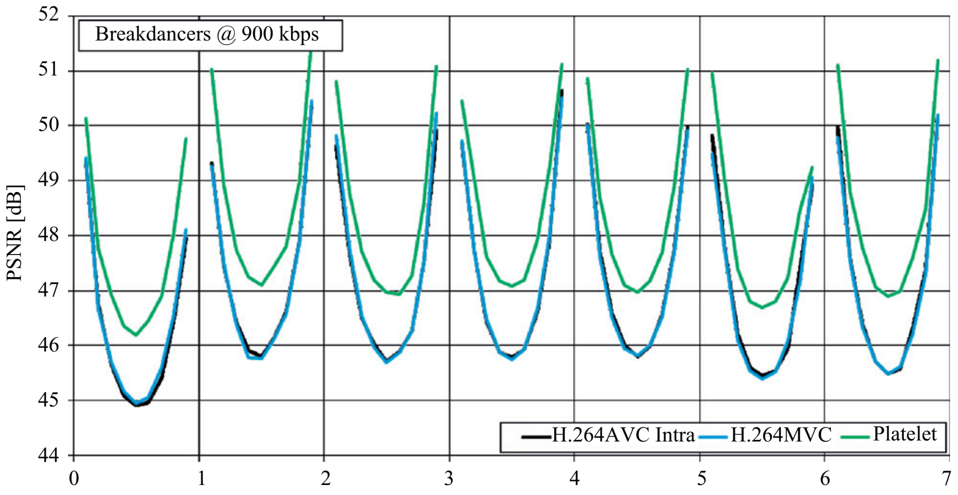


Fig. 9. (*Continued*)

coding approaches H.264/AVC Intra, H.264/MVC and Platelet as the average value over the 25 picture long sequences. The characteristic of the curves highlights the general effect of geometry distortions on view synthesis rendering. The errors in rendered virtual view increase with the distance between original and virtual camera view. Thus, the lowest rendering quality is observed for virtual views at half distance between the two original cameras. Table 1 additionally compares the performance differences of the three coding approaches that result from coding on and rendering, respectively. For each of the three ratepoints the average PSNR difference relative to H.264/AVC Intra was calculated, once from the coding experiments results and once from the view synthesis rendering quality results. These results indicate that Platelet-based coded depth video always achieves a higher rendering quality than the reference coding method, although the coding quality is worse. For the “Breakdancers” sequence Platelet-based coding even outperforms H.264/MVC, while for “Ballet” H.264/MVC achieves the highest rendering quality of all approaches due to the significant gain in coding PSNR. In return this means that Platelet-based coding would clearly outperform H.264 in virtual view rendering quality, if comparing compressed depth-images with equal PSNR instead of equal bitrate.

Besides the objective results rendering output examples are presented in Fig. 10. The image on the left is rendered from original, uncompressed color and depth data and shows a detail for a foreground object which is delimited from the background by a sharp edge. The other images show the same detail rendered with H.264/AVC Intra, Platelet-based and H.264/MVC compressed depth-images at a low bitrate. According to the objective results these examples highlight the effect of the different types of coding artifacts produced by H.264-based and Platelet-based coding. Due to the blurring and ringing artifacts the object boundaries in the two examples rendered with H.264 coded depth are fuzzy. In contrast the example for Platelet-based depth coding

Table 1
Comparison of ΔPSNR for coding and virtual view rendering for low, middle and high bitrates.

Ballet	Platelet-based			H.264/MVC		
Bitrate (kbps)	260	590	790	260	590	790
ΔPSNR (dB) coding	−1.304	−2.637	−2.705	4.423	3.884	3.538
ΔPSNR (dB) rendering	1.477	1.838	1.750	2.338	2.633	2.401
Breakdancers						
Bitrate (kbps)	200	420	900	200	420	900
ΔPSNR (dB) coding	−0.081	−0.926	−1.006	1.388	0.134	0.162
ΔPSNR (dB) rendering	1.155	1.063	1.162	0.568	0.008	−0.004

ΔPSNR shows the average difference in PSNR results between H.264/AVC Intra and Platelet-based and between H.264/AVC Intra and H.264/MVC depth coding.

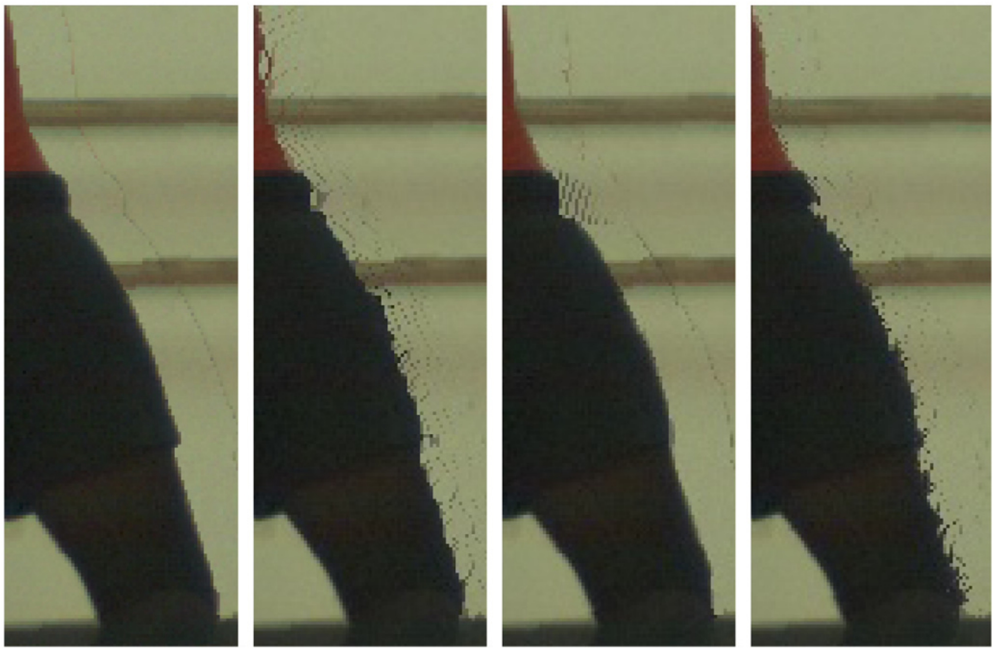


Fig. 10. Detail examples for depth coding artifacts with view synthesis rendering. “Ballet” sequence at low bitrate rendered from original texture plus (from left to right) original depth, H.264/AVC Intra coded depth, Platelet coded depth, and H.264/MVC coded depth.

shows that the object boundaries are preserved very well. Compared to the original the edges are more angular shaped and some rendering artifacts from flattened surfaces can be seen.

6. Summary and conclusions

The article presents a comparative study on the interaction of multiview depth video compression and view synthesis rendering for MVD data. Two approaches for depth compression were introduced, namely H.264/MVC and Platelet-based coding. H.264/MVC is the latest extension to the H.264 video coding standard, characterized by achieving the best R-D performance for multiview color video from temporal and inter-view prediction. The other approach is Platelet-based coding, a novel depth coding algorithm, characterized by being adapted to the special characteristics of depth-images. For better comparability of these two approaches H.264/AVC Intra was chosen as a reference method, combining intra only prediction like Platelet-based and H.264 like MVC coding. A comparison of the R-D performance turned out that H.264/MVC is the most efficient coding method due to inter prediction. Moreover Platelet-based coding is also outperformed by the reference method since H.264 is optimized for bitrate efficiency to a great extent. In contrast to color video, depth coding artifacts lead to geometry distortions in the 3D representation of a depth-image that propagate into rendered virtual views. Therefore two approaches for analyzing the effect of geometry distortions caused by depth coding artifacts were presented. First is a method for generating 3D surface meshes from depth-images, including a quality measure for the similarity between meshes from compressed and uncompressed depth based on the Hausdorff distance. The second approach evaluates how the different types of depth coding artifacts influence the quality of rendered virtual views. For this purpose view synthesis rendering with geometry projection and texture blending is introduced, including a method for evaluating compression effects. In contrast to the R-D performance, the experimental results for the mesh distance as well as the view synthesis rendering evaluation clearly indicate that Platelet-based coding performs best, due to its ability to preserve sharp object boundaries and edges. This leads to the conclusion that Platelet-based coding enables higher rendering quality than H.264-based coding, because of being adapted to the special characteristics of depth-images. For improving the rate efficiency the approach may be extended by inter prediction in the future. There is evidence that a multiview optimized Platelet compression technique could show further gains, taking into account that Platelet-based coding outperforms H.264/AVC and the improvement of H.264/MVC over H.264/AVC. Consequently the development of advanced depth coding algorithms in the context of MVD needs to optimize the performance with respect to geometry distortions, especially the quality of rendered virtual views. This requires future research to address joint compression and rendering algorithms as well as appropriate quality metrics.

Acknowledgements

This work is partially supported by European Union 6th Framework Program under Grant no. FP6-511568 (3DTV NoE Project). We would like to thank the Interactive Visual Media Group of Microsoft Research for providing the “Ballet” and “Breakdancers” data sets.

References

[1] O.G.G. Albert Cohen, Ingrid Daubechies, M.T. Orchard, On the importance of combining wavelet-based non-linear approximation in coding strategies, IEEE Trans. Inf. Theory 48 (1997) 1895–1921.
[2] N. Aspert, D. Santa-Cruz, T. Ebrahimi, MESH: Measuring Error between Surfaces using the Hausdorff distance, in: Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), 2002, pp. I-705-708.
[3] G.-C. Chang, W.-N. Lie, Multi-view image compression and intermediate view synthesis for stereoscopic applications, in: Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), 2000, pp. 277–280.
[4] S. Chen, L. Williams, View interpolation for image synthesis, ACM Comput. Graphics (Proceedings of SIGGRAPH’93) (1993) 279–288.
[5] P.A. Chou, T.D. Lookabaugh, R.M. Gray, Optimal pruning with applications to tree-structured source coding and modeling, IEEE Trans. Inf. Theory 35 (2) (1989) 299–315.
[6] D. Donoho, Wedgelets: Nearly minimax estimation of edges, Ann. Stat. 27 (3) (1999) 859–897.
[7] O. Faugeras, Three-Dimensional Computer Vision: A Geometric Viewpoint, MIT Press, Cambridge, Massachusetts, 1993.
[8] R. Hartley, A. Zisserman, Multiple View Geometry in Computer Vision, Cambridge University Press, Cambridge, 2003.
[9] D. Huttenlocher, G. Klanderman, W. Rucklidge, Comparing images using the Hausdorff distance, IEEE Trans. Pattern Anal. Mach. Intell. 15 (9) (1993) 850–863.
[10] ISO/IEC JTC1/SC29/WG11, Text of ISO/IEC FDIS 23002-3 Representation of Auxiliary Video and Supplemental Information, Doc. N8768, Marrakech, Morocco, January 2007a.
[11] ISO/IEC JTC1/SC29/WG11, Text of ISO/IEC 13818-1:2003/FDAM2 Carriage of Auxiliary Data, Doc. N8799, Marrakech, Morocco, January 2007b.
[12] ITU-T, ISO/IEC JTC 1, Advanced video coding for generic audio-visual services, ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG4-AVC), Version 1: May 2003, Version 2: Jan. 2004, Version 3: Sep. 2004, Version 4: July 2005.
[13] P. Merkle, A. Smolic, K. Müller, T. Wiegand, Multi-view Video Plus Depth Representation and Coding, in: Proceedings of the IEEE International Conference on Image Processing (ICIP, San Antonio), TX, USA, September 2007a, pp. I-201–204.
[14] P. Merkle, A. Smolic, K. Mueller, T. Wiegand, Efficient prediction structures for multiview video coding, invited paper,in: Proceedings of the IEEE Transactions on Circuits and Systems for Video Technology, 17(11)(2007b)1461–1473.
[15] Y. Morvan, D. Farin, P. H. N. de With, Depth-Image Compression based on an R-D Optimized Quadtree Decomposition for the Transmission of Multiview Images, in: Proceedings of the IEEE International Conference on Image Processing (ICIP), San Antonio, TX, USA, September 2007, pp. V-105–108.
[16] A. Ortega, K. Ramchandran, Rate-distortion methods for image and video compression, IEEE Signal Process. Mag. 15 (6) (1998) 23–50.
[17] H. Ozaktas, L. Onural, Three-Dimensional Television: Capture, Transmission, and Display, Springer, Heidelberg, December 2007.
[18] H. Schwarz, D. Marpe, T. Wiegand, Analysis of hierarchical B pictures and MCTF, in: Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), Toronto, Ontario, Canada, July 2006, pp. 1929–1932.
[19] K. Shoemake, Animating rotation with quaternion curves, ACM Comput. Graphics (Proceedings of SIGGRAPH) (1985) 245–254.

[20] R. Shukla, P.L. Dragotti, M.N. Do, M. Vetterli, Rate-distortion optimized tree-structured compression algorithms for piecewise polynomial images, IEEE Trans. Image Process. 14 (2005) 343–359.
[21] R.Y. Tsai, A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV camera and lenses, IEEE J. Robotics Autom. RA 3 (4) (1987) 323–344.
[22] T. Wiegand, G.J. Sullivan, G. Bjøntegaard, A. Luthra, Overview of the H.264/AVC video coding standard,, IEEE Trans. Circuits Syst. Video Tech. 13 (7) (2003) 560–576.
[23] R.M. Willett, R.D. Nowak, Platelets: A multiscale approach for recovering edges and surfaces in photon-limited medical imaging, IEEE Trans. Med. Imaging 22 (2003) 332–350.