Partial Depth Image Based Re-Rendering for Synthesized View Distortion Computation

Gerhard Tech, Karsten Müller, *Senior Member, IEEE*, Heiko Schwarz, and Thomas Wiegand, *Fellow, IEEE*

***Abstract*— 3D video systems transmit depth maps in order to render synthesized views (SVs) at a receiver. To anticipate this purpose when processing a depth map, a sender-side depth processing algorithm (DPA), e.g. a depth encoder, can also render the SVs, compute their SV distortion (SVD), and adapt to it. This requires a low-complexity algorithm as computational resources are usually limited. We propose such an algorithm in this paper. First, we discuss a measure that relates a depth change to an SVD change using rendering. Then, we present an optimized process combining basic rendering steps, as warping, occlusion handling, interpolation, hole filling, and blending. Furthermore, we analyze which parts of an SV are affected by a depth change and modify the process to re-render only them. The resulting algorithm is significantly less complex than an unoptimized rendering-based variant and quantifies the SVD more accurately than existing estimation methods. The algorithm is used by the 3D-High Efficiency Video Coding reference software encoder as the main method for distortion computation and can also be used by other DPAs.**

***Index Terms*— 3D video, 3D-High Efficiency Video Coding (HEVC) reference software, complexity reduction, depth image-based rendering (DIBR), partial re-rendering, synthesized view distortion change (SVDC), view synthesis optimization.**

I. Introduction

3D video can be presented on glasses-free autostereoscopic displays emitting a large number of views. As conventionally only a small number can be recorded and transmitted, synthesis of additional views should be supported by a 3D video format. Complying formats often consist of textures depicting the 3D scene from different perspectives and a depth map per texture [1]. The texture samples can be warped using disparities derived from the depth maps [commonly called depth image-based rendering (DIBR)] to create synthesized views (SVs) for an autostereoscopic display [2].

DIBR can be part of the receiver of a 3D video system (Fig. 1). Then, the system not only transmits texture, but also generates and transmits depth maps by additional sender-side depth processing steps. The first step, which is required when depth maps are not directly recorded, is depth estimation, which uses stereo matching [3] to provide a depth map. The depth map can then be filtered [4] or manually modified in an enhancement step. After enhancement, the depth map is lossy encoded and transmitted. Transmitted depth maps are decoded at the receiver and used to generate SVs by DIBR.

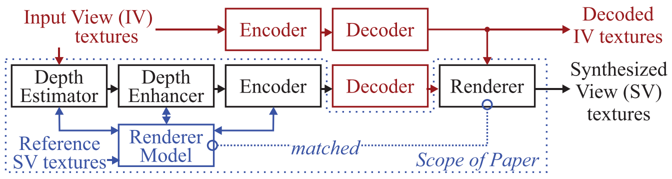


Fig. 1. Sender and receiver of a 3D video system.

Minimizing the SV distortion (SVD) is an obvious target for a 3D video system, and thus also in compression performed by the depth encoder. Conventional encoders, however, optimize based on the depth distortion and are not aware of the SVD. To overcome this, we proposed to already render at the encoder [5], [6]. This way, an encoder can compute the exact SVD. With such a modification, we reduced the depth bit rate to about 50% at constant SVD.

However, rendering at the encoder requires additional computational operations. Since computational resources are usually limited, a low-complexity algorithm especially tailored for SVD calculation is required. We present such an algorithm in this paper. Several questions concerning its optimized implementation, complexity, rendering performance, and application are addressed. Answers provide an insight how and at what costs rendering for SVD computation can be applied in a 3D video system. This might not only be in encoding, but also in other depth processing algorithms (DPAs) as depth estimation or enhancement.

A. Problem Statement and Paper Outline

The motivation for this paper is that a DPA improves when it considers the SVD. We define this use case with the notations in Table I[1] as follows. A DPA (e.g., an encoder) processes a depth map $s_{D,l}$ of an input view (IV), which is used in DIBR to render a texture $s'_{T,l}$ of an SV. The DPA changes the values

Manuscript received March 18, 2016; revised August 1, 2016 and September 14, 2016; accepted November 8, 2016. Date of publication November 22, 2016; date of current version June 4, 2018. This paper was recommended by Associate Editor G. Cheung.

G. Tech, K. Müller, and H. Schwarz are with the Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute, 10587 Berlin, Germany (e-mail: gerhard.tech@hhi.fraunhofer.de).

T. Wiegand is with the Fraunhofer Institue for Telecommunications, Heinrich Hertz Institute, 10587 Berlin, Germany, and also with the Department of Electrical Engineering and Computer Science, Technical University of Berlin, 10587 Berlin, Germany.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2016.2631568

[1]Most autostereoscopic displays use views from a parallel camera setup, such that rows with the same vertical position y in different 2D signals correspond to each other. As we constrain rendering similarly and avoid vertical dependencies, we can drop y and only treat 1D signals and regions.

TABLE I
OVERVIEW OF FREQUENTLY USED SYMBOLS

Signals		**Regions (multiple intervals)**	
IV texture$^+$:	$s_{T,l}$ $s_{T,r}$	Changed IV$^\#$:	$B = [x_{B,s}, x_{B,e}]$
Up-sampled IV texture^{*+}:	$\hat{s}_{T,l}$ $\hat{s}_{T,r}$	Processed IV:	B_P see (11)
IV depth*:	$s_{D,l}$ $s_{D,r}$	Old SV:	$B'_S = [x'_{S,s}, x'_{S,e}]$
Depth candidate in $B^\#$:	$\tilde{s}_B$	New SV:	$\tilde{B}'_S = [\tilde{x}'_{S,s}, \tilde{x}'_{S,e}]$
IV depth with $\tilde{s}_B$ in B:	$\tilde{s}_{D,l}$ $\tilde{s}_{D,r}$	Superset of $\tilde{B}'$:	$\tilde{B}'_M = [\tilde{x}'_{M,s}, \tilde{x}'_{M,e}]$
Distance from camera:	$s_{Z,l}$ $s_{Z,r}$	Changed SV:	$\tilde{B}' = [x'_{C,s}, x'_{C,e}]$
Disparities:	$s_{\Delta,l}$ $s_{\Delta,r}$	**Auxiliary variables**	
Occlusion signal.*:	$s_{O,l}$ $s_{O,r}$	Occlusion flag:	b_O
Extrapolated SV texture*:	$s'_{T,l}$ $s'_{T,r}$	Minimal occluding pos.:	x'_O $\dot{x}'_O$
Extrapolated SV depth*:	$s'_{D,l}$ $s'_{D,r}$	Minimal changed pos.:	x'_C
Disocclusion flags*:	$s'_{H,l}$ $s'_{H,r}$	**Other variables**	
Combined SV texture:	s'_T	Horizontal positions:	x $\hat{x}$ x' $\dot{x}'$
Reference texture^{*+}:	s'_{Ref}	FG boundary pos.:	$\dot{x}'_{FL}$ $\dot{x}'_{FR}$
Per sample SVD*:	s'_E	Width of B:	w_B
Intervals		Width of picture$^+$:	w
IV:	$[x_s, x_e]$	Stop position:	x_P
SV:	$[x'_s, x'_e]$ $[\dot{x}'_s, \dot{x}'_e]$	SVDC:	ΔD

IV: Input View; SV: Synthesized View; ($'$): In SV; ($\dot{}'$): Integer position in SV; ($\hat{}$): In up-sampled IV; ($\tilde{}$): Changed by depth change $\tilde{s}_B$ in B; ($_{,s}$) and ($_{,e}$): Left and right interval or region boundaries; $^+$constant; ($_{,l}$) and ($_{,r}$): Related to left and right view; *RM state; $^\#$RM input.

of a region B (e.g., a coding block) in $s_{D,l}$ to candidate values (e.g., provided by a particular coding mode), which are in the following called depth candidate $\tilde{s}_B$. In general, this creates a changed version of $s'_{T,l}$ denoted as $\tilde{s}'_{T,l}$. In order to reject or adopt the depth candidate $\tilde{s}_B$ in B (e.g., to decide whether to use the coding mode), it is now of interest for the DPA to quantify the distortion in $\tilde{s}'_{T,l}$ introduced by $\tilde{s}_B$.

The basic idea addressed in this paper is to quantify this distortion by partial re-rendering and direct computation. This means by: 1) re-rendering the region $\tilde{B}'$ in that the SV textures $s'_{T,l}$ and $\tilde{s}'_{T,l}$ differ and 2) computing the change of the SVD in $\tilde{B}'$ that occurs when $s'_{T,l}$ changes to $\tilde{s}'_{T,l}$. In doing so, the SVD in $s'_{T,l}$ and $\tilde{s}'_{T,l}$ is calculated as sum of squared differences (SSDs) compared with a reference texture s'_{Ref}, which can be a recorded texture or SV texture rendered from initial IV depth. Design, implementation, and application of such an approach raise several questions, which we address.

An initial question is how the SVD computation method described before is motivated. To answer this, we flesh out our findings from [5] in Section II and present a *rendering-based distortion measure* called SVD change (SVDC), which we embed in a framework called renderer model (RM).

To calculate the SVDC, the RM performs re-rendering, which increases computational complexity. Since resources are usually limited, a major question is how this can be done with low complexity. To achieve this, we combine different rendering approaches and implement them in an optimized way (Section III). Result is an integrated low complexity *rendering algorithm* that is used by the RM in SVDC calculation.

For SVDC calculation, the RM only needs to re-render the region $\tilde{B}'$ that changes in the SV when the DPA changes the region B in the depth map $s_{D,l}$ to the candidate $\tilde{s}_B$. This raises the questions, how to derive boundaries of $\tilde{B}'$ and how to extent the rendering algorithm to start and stop at these boundaries. Answers are provided in Section IV and enable the RM to perform *partial re-rendering and SVDC calculation*.

The RM implementation targets low complexity, a sufficient SV quality, and should also perform well when the receiver uses a view synthesis algorithm different from the RM. An *evaluation* in Section V discusses whether this has been reached by comparing the RM with other approaches. Finally, Section VI discusses evaluation results in the light of constraints imposed by the different *application scenarios* and addresses how DPAs can be improved by partial re-rendering for SVDC calculation.

B. Relationship to Other Works

In recent years, several methods have been proposed for SVD estimation, which model the relationship between depth distortion and SVD linearly [7] or with polynomials [8], estimate the SVD due to warping with incorrect disparities [9]–[11], or model sample interpolation [12], [13] and occlusions [12]. In summary, all methods use simplified models. Our method—the RM—is fundamentally different: it renders with an actual view synthesis algorithm and computes an exact SVDC. This way, it can consider all techniques usually used for view synthesis, as warping, occlusion handling, hole filling, fractional sample interpolation, and view combination.

For rendering, there is a great variety of methods to increase quality as by depth filtering [14], boundary noise removal [15], or advanced hole filling [16]; and to reduce complexity as for hardware implementations [17], by switching between IVs [18], or by input driven real-time implementations [19]. Furthermore, the *VSRS 3.5* software comprises a rich set of different methods [20]. Those prior art methods render whole SVs. In contrast, the RM is specialized to only re-render parts of the SV to determine the SVDC. We show how this can be done with low complexity by introducing a rendering algorithm in Section III and extending it for partial re-rendering in Section IV.

Our contribution in Section III is that we select and modify basic rendering techniques and combine them to an algorithm that can be extended by partial re-rendering. More specifically, we combine: 1) techniques commonly used (in [18] and [20]), such as warping, hole detection, hole, and margin filling; 2) occlusion detection from [19], which we extend for the detection of foreground (FG) edges; 3) correct rendering of FG edges [21] having an effect similar to boundary aware splatting [20]; 4) view blending similar to [20], but modified to operate after hole filling; and 5) a texture mapping approach, inspired by backward warping in [20], but embedded by us in an intervalwise processing scheme. In summary, Section III provides a comprehensive overview on how these techniques can be implemented in an optimized way that allows to understand the required complexity down to single operations. Then, in Section IV, we present our main contribution. We propose how the rendering algorithm can find IV positions that ensure that the whole changed region $\tilde{B}'$ is re-rendered and how it can be extended to start and stop at these positions.

Known as view synthesize optimization, our method is part of the 3D-High Efficiency Video Coding (HEVC) reference software encoder [22], [23], since we proposed it as starting

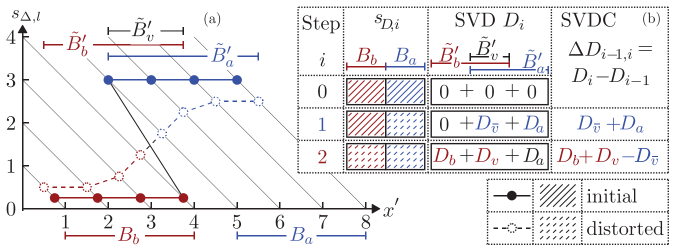


Fig. 2. (a) SV texture regions related to an IV depth distortion. x' is the SV position of a sample when shifted with the disparity $s_{\Delta,l}$. A sample's IV position can thus be found by following its intersecting diagonal line to the x' axis. Samples (and their connections) on the top are closer to the camera and thus occlude samples below. (b) Example for SVDC calculation.

point for the development of 3D-HEVC [24]. It is, according to JCT-3V's test conditions [25], the main method for distortion derivation in rate-distortion-based mode selection in depth coding (for details, see [23] and Section VI). The first encoder version including all optimizations we describe in this paper is HTM-16.1 [22]. We developed the RM in parallel with the *VSRS 1D-Fast* software [24] (the main renderer in 3D-HEVC development [23]). In contrast to the RM, *1D-Fast* does not support re-rendering, but boundary noise removal. Although we proposed the SVDC and the RM before for encoding [5], [6], [24], our previous papers and standardization contributions provide only a high-level description of the underlying algorithm; in particular, they do not present the ideas behind and analysis of partial re-rendering—the main contributions of this paper.

II. Rendering-Based Distortion Measure

This section addresses an initial question—how to measure the exact SVD introduced by a depth candidate using rendering. To answer this, we first analyze the relationship between the IV depth and SV texture, which is given by the DIBR process. DIBR shifts the samples of the IV texture $s_{T,l}$ by disparities $s_{\Delta,l}$ derived from the IV depth map $s_{D,l}$. More specifically, in a parallel coplanar camera setup, a sample at an IV position x is shifted to the SV position

$$x' = \mathrm{f}(x) = x - s_{\Delta,l}(x). \tag{1}$$

With a depth representation format similar to the one in [26], the disparity $s_{\Delta,l}$ can be derived as follows:

$$1/s_{Z,l}(x) = c_0 \cdot s_{D,l}(x) + c_1 \tag{2}$$
$$s_{\Delta,l}(x) = c_2/s_{Z,l}(x). \tag{3}$$

Parameters c_0, c_1, and c_2 are given by the camera setup. $s_{Z,l}$ represents the physical depth, thus the distance to the camera plane. $s_{D,l}$, in contrast, represents rather scaled disparity values although commonly called depth map.

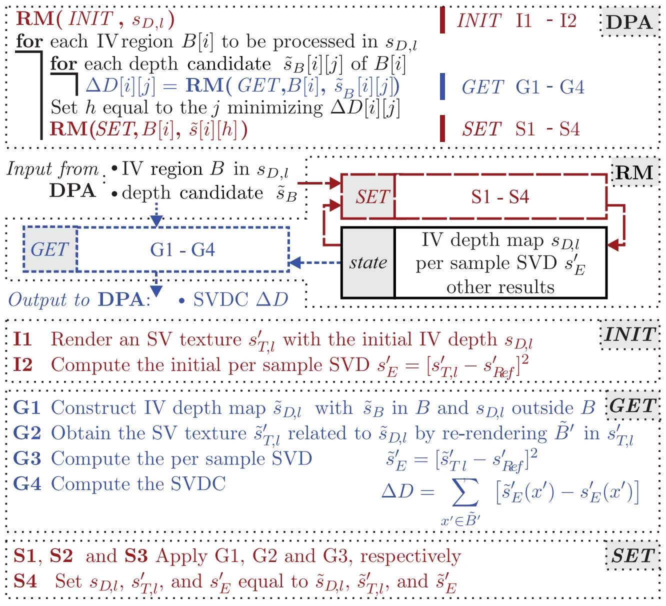
The relationship defined by (1) is visualized in an x'-$s_{\Delta,l}$-space in Fig. 2. Therefore, samples of the IV texture $s_{T,l}$ are marked with solid dots. The horizontal position of a dot, however, does not correspond to the sample's IV position x, but to its SV position x'. The vertical position corresponds to its disparity $s_{\Delta,l}(x)$ used for shifting. Equation (1) shows that when $s_{\Delta,l}(x)$ of a sample from an IV position x changes, its position in the x'-$s_{\Delta,l}$-space can only move diagonally. Furthermore, (3) shows that $s_{\Delta,l}$ increases with decreasing $s_{Z,l}$. This means, when two points share the same SV position x', the upper point in the x'-$s_{\Delta,l}$-space is closer to the camera plane and the lower point is occluded. To understand which samples are in the FG and visible in the SV, the x'-$s_{\Delta,l}$-space is thus a valuable tool.

We use this tool in Fig. 2 and show what happens when the initial IV depth (associated with the solid dots) gets distorted: the samples move diagonally in the x'-$s_{\Delta,l}$-space from their initial positions (solid dots) to distorted positions (dashed dots). This distorts the SV texture. To define an SVD-based measure, the question is now how to map the SVD to distorted IV depth regions. For simplification, we analyze this with respect to two IV regions B_a and B_b (shown in Fig. 2), which are related to two SV regions $\tilde{B}'_a$ and $\tilde{B}'_b$. $\tilde{B}'_a$ is the SV region affected by the depth distortion in B_a; it is thus spanned by the left- and rightmost of initial and distorted SV positions related to B_a. Accordingly, $\tilde{B}'_b$ is the SV region that is affected by the distortion in B_b. Obviously, the SVD in nonoverlapping parts of $\tilde{B}'_a$ and $\tilde{B}'_b$ can solely be related to B_a and B_b. In the overlapping part $\tilde{B}'_v$, however, the SVD is caused by the distortions in B_a and B_b (e.g., the sample from B_b erroneously shifted to $x' = 2.75$ is only visible as B_a is distorted). This shows that distorted SV regions cannot be mapped bijectively to IV regions, which is a first problem for determining the SVD related to a particular IV region, as it is unclear how to divide up the SVD in SV regions that map to more than one IV region. A second problem arises in practice: when a DPA processes B_a first, the distortion in B_b, and thus the SVD in the affected region $\tilde{B}'_b$ and in the overlap $\tilde{B}'_v$ are not known.

To resolve both problems, we proposed the SVDC [5], which is conceptually agnostic about where the SVD related to a distorted IV region B is located in the SV. The SVDC is the change of the total SVD of a whole SV texture that occurs when the depth in an IV region B changes from initial to distorted depth, while other IV depth regions contain distorted depth, if already known, and initial depth otherwise.

Fig. 2(b) shows how the SVDC can be calculated for B_a and B_b in three steps. Step 0 renders an SV texture from the initial depth $s_{D,0}$; as this SV texture is used as reference, its SVD D_0 is equal to zero. Step 1 renders an SV texture from a depth map $s_{D,1}$ with initial depth in B_b (as the distorted depth is not yet known) and distorted depth in B_a, derives the SVD D_1 related to $s_{D,1}$, and finally the SVDC for B_a as $\Delta D_{0,1} = D_1 - D_0$. Step 3 renders using a depth map $s_{D,2}$ with distorted depth in B_a and B_b (as B_a has already been processed its distortion is known), derives the related SVD D_2, and finally the SVDC for B_b as $\Delta D_{1,2} = D_2 - D_1$.

In doing so, both problems raised earlier are addressed. The second problem is solved by using initial depth when the distorted depth is not yet known. In step 1, the SVD $D_{\bar{v}}$ in $\tilde{B}'_v$ is calculated based on the assumption that B_b contains initial depth. The first problem is then solved in



```
RM( INIT , $s_{D,l}$ )
for each IV region $B[i]$ to be processed in $s_{D,l}$
    for each depth candidate $\tilde{s}_B[i][j]$ of $B[i]$
        $\Delta D[i][j]$ = RM( GET, $B[i]$, $\tilde{s}_B[i][j]$ )
    Set $h$ equal to the $j$ minimizing $\Delta D[i][j]$
    RM( SET, $B[i]$, $\tilde{s}[i][h]$ )
```

I1 Render an SV texture $s'_{T,l}$ with the initial IV depth $s_{D,l}$

I2 Compute the initial per sample SVD $s'_E = [s'_{T,l} - s'_{Ref}]^2$

G1 Construct IV depth map $\tilde{s}_{D,l}$ with $\tilde{s}_B$ in B and $s_{D,l}$ outside B

G2 Obtain the SV texture $\tilde{s}'_{T,l}$ related to $\tilde{s}_{D,l}$ by re-rendering $\tilde{B}'$ in $s'_{T,l}$

G3 Compute the per sample SVD $\tilde{s}'_E = [\tilde{s}'_{T\,l} - s'_{Ref}]^2$

G4 Compute the SVDC $\Delta D = \sum_{x' \in \tilde{B}'} [\tilde{s}'_E(x') - s'_E(x')]$

S1, **S2** and **S3** Apply G1, G2 and G3, respectively

S4 Set $s_{D,l}$, $s'_{T,l}$, and s'_E equal to $\tilde{s}_{D,l}$, $\tilde{s}'_{T,l}$, and $\tilde{s}'_E$

Fig. 3. RM and its application in a DPA.

step 2 by considering for $\tilde{B}'_v$ only the SVDC that occurs when B_b gets additionally distorted, i.e., $D_v - D_{\tilde{v}}$. This way, the sum of SVDCs separately calculated for B_a and B_b (i.e., $\Delta D_{0,1} + \Delta D_{1,2}$) is equal to the SVDC caused by a joint change of B_a and B_b from initial to distorted depth (i.e., $\Delta D_{0,2} = D_2 - D_0$). This property (commonly called additivity) is usually required by DPAs (e.g., by encoder for splitting of blocks) and the reason for using the SVDC instead of the SVD.

A. Renderer Model

So far, we described the SVDC only conceptually. The question is now how to efficiently derive the SVDC of an SV texture $s'_{T,l}$ that occurs when a DPA changes the related IV depth map $s_{D,l}$ in an IV region B to a depth candidate $\tilde{s}_B$. As B is, in general, only a small region in $s_{D,l}$, only a related small region $\tilde{B}'$ in $s'_{T,l}$ changes. Therefore, it suffices to re-render $\tilde{B}'$ and to calculate the SVDC within it. To do this, we embedded SVDC computation in a framework, which is called RM [5] and shown in Fig. 3. The RM has a state comprising the current IV depth $s_{D,l}$, the current SV texture $s'_{T,l}$, and its per sample SVD s'_E. The RM initializes $s'_{T,l}$ and s'_E by steps I1 and I2 in Fig. 3 before the DPA starts processing the IV depth $s_{D,l}$. While processing $s_{D,l}$, the DPA can input an IV region B and its depth candidate $\tilde{s}_B$ to the RM and invoke it in two modes, called *GET* and *SET* mode.

In *GET* mode, the RM computes the SVDC ΔD caused by $\tilde{s}_B$ in B with respect to its current state (G1–G4). For this, it re-renders $\tilde{B}'$ (G2), and computes the new per sample SVD $\tilde{s}'_E$ (G3) and the SVDC in $\tilde{B}'$ relative to the old SVD s'_E (G4). In doing so, the RM state remains unchanged, so that memory bandwidth is reduced. This way, the DPA can check the SVDC of multiple candidates $\tilde{s}_B$ for B (e.g., $\Delta D[i][j]$ for $B[i]$ in Fig. 3) before choosing one (e.g., $\tilde{s}_B[i][h]$).

When the DPA has chosen a candidate, the final depth for the respective IV region B is known and can be adopted to the

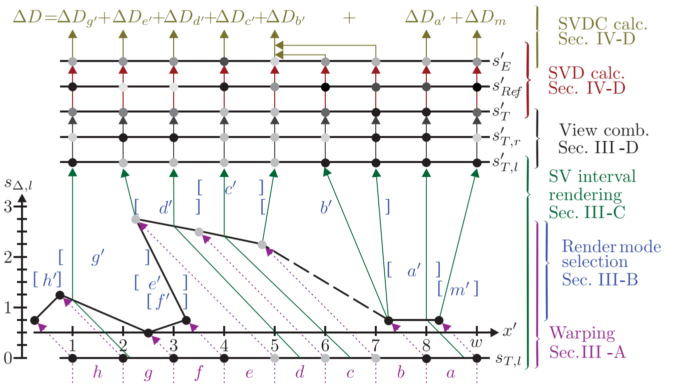


Fig. 4. Overview of the rendering approach, its single steps and involved signals. Depicted signals represent one row of input, intermediate, or output data. Arrows show the relationship between samples or their positions.

RM's state, so that it is regarded when testing candidates for other IV regions. For this, the DPA can invoke the RM's *SET* mode. In this mode, the RM adopts $\tilde{s}_B$ in B by performing steps S1–S4, i.e., it re-renders $\tilde{B}'$ (S2), computes $\tilde{s}'_E$ in $\tilde{B}'$ (S3), and stores changed signals as new state (S4).

This way, the RM's state is always aligned with decision taken by the DPA, so that SVDC calculation is always based on all adopted candidates and can be performed efficiently by partial re-rendering of $\tilde{B}'$.

III. Basic Rendering Algorithm

DPAs conventionally evaluate multiple candidates for a particular IV depth region. In this use case, SVDC calculation can easily exceed computational resources when re-rendering with a sophisticated high complexity approach. Consequently, a low-complexity algorithm that is extendable for partial re-rendering is needed. For this, we modify and combine several basic methods (as described in Section I) to a new rendering algorithm that can be implemented in an optimized way. Single steps of the resulting process are shown in Fig. 4 and discussed in this section. For simplification, we assume that the RM renders a whole SV texture s'_T by iterating once over all IV sample positions. The extension for partial re-rendering and SVDC calculation is then discussed in Section IV.

A. Intervalwise Processing and Warping Step

For rendering the RM iterates over the IV from right to left (to enable occlusion detection described later), as shown in Fig. 5, it starts with the current IV position x_s equal to the row width w. After the RM has initialized rendering of the current row (P1 in Fig. 5) and after further iterations, it stores x_s as the last processed IV position x_e before decrementing x_s for the next iteration (P2). This way, an IV interval $[x_s, x_e]$ (e.g., b in Fig. 4) is provided in each iteration.

To render the SV samples related to $[x_s, x_e]$, the RM derives two SV positions (P3). The first, x'_s, is the SV position related to the current IV position x_s and thus derived using $x'_s = \mathrm{f}(x_s)$ with (1) using quarter sample precision. The second, x'_e, is the SV position related to the last IV position x_e and is, as such,

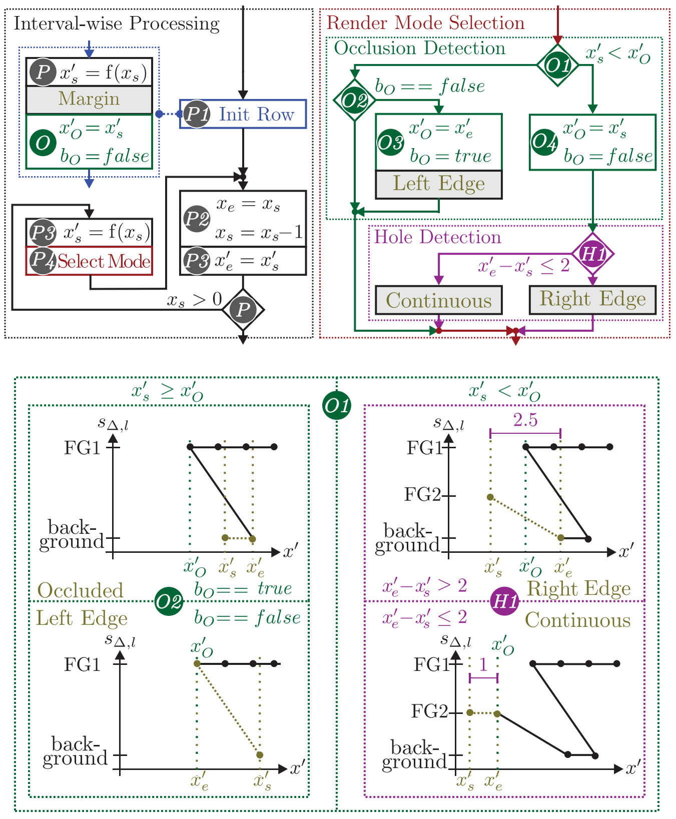


Fig. 5. Top: intervalwise processing and render mode selection. Bottom: examples for render mode selection.

set equal to x'_s of the last iteration. Both positions provide the SV interval $[x'_s, x'_e]$ (e.g., b' in Fig. 4).

Starting with render mode selection (P4), the RM then applies steps described in Sections III-B–III-D consecutively for the SV interval of the current iteration before continuing with the next iteration. This way, the RM renders all SV samples related to an IV interval in one iteration, so that it can start and stop easily at arbitrary IV positions, as later required for partial re-rendering.

B. Render Mode Selection Step

When rendering a right SV from a left IV, IV samples are shifted leftward with (1). As FG objects are shifted further leftward than the background, they occlude it on their left side and holes (disocclusions) occur on their right-hand side. Moreover, no IV sample usually remains on the right SV margin. To adapt to such scenarios, the RM distinguishes the following SV interval types, which are shown in Figs. 4 and 5.

A *left edge interval* (e.g., e' in Fig. 4) occurs on the left of an FG object. x'_e is the leftmost FG object position and is not occluded. Other positions are occluded by the FG object. An *occluded interval* (f') can occur on the left of an FG object after a left edge interval. x'_s and x'_e are both occluded by the FG object. A *right edge interval* (b') occurs on the right-hand side of an FG object. x'_s is the rightmost FG object position and is not occluded. x'_e belongs to the background. SV positions between x'_s and x'_e are disoccluded. The right-hand side may be occluded by another FG object on the right-hand side of x_e in the IV. In a *continuous interval* (a', c', d', and g'), all positions in $[x'_s, x'_e]$ belong to the same object. x'_s is not occluded. The right-hand side may additionally be occluded (g') by another FG object on the right-hand side of x_e. When all IV samples are shifted leftward, a gap occurs on the right-hand side in the SV in a *margin interval* (m'). x'_s is given by shifting the rightmost IV position, and thus by $f(w)$. x'_e is the rightmost SV position w.

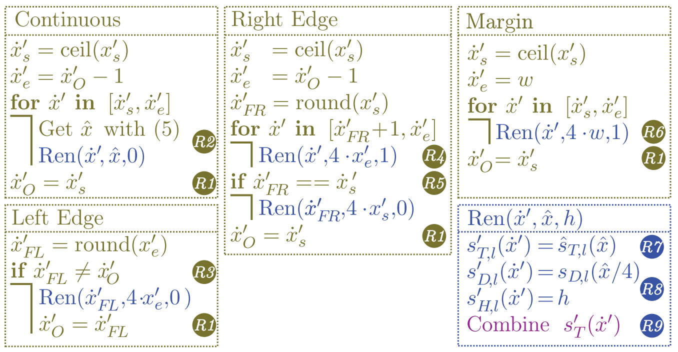
As margin intervals can only occur at the right SV margin, they are directly rendered as described in Section III-C5 when the RM initializes a row (P1). In subsequent iterations, the question is how the RM can differentiate between the other SV interval types to select the corresponding render mode. This is done by *occlusion detection* and *hole detection* as follows.

The first objective of *occlusion detection* is to find left edge and occluded intervals. Both differ from other types in that the current SV position x'_s is occluded. To detect this, the RM utilizes an auxiliary variable x'_O, which is called minimal occluding position. While processing the IV from right to left, x'_O corresponds to the leftmost fractional SV position that has already been rendered [19], [27] in previous iterations. When x'_s is greater than or equal to x'_O, it is occluded (see [27] or the Appendix) and the current SV interval is a left edge or occluded interval. To distinguish between both is the second objective. They differ in that the last SV position x'_e is also occluded for occluded intervals. To detect this, we extended the method from [27] by another auxiliary variable b_O, which is called occlusion flag and indicates whether x'_s of the SV interval rendered in the previous iteration (and thus x'_e in the current) is occluded.

More specifically, the RM addresses both objectives by evaluating x'_O and b_O in conditions (O1) and (O2) in Fig. 5. The result determines how the RM continues.

1) $x'_s \geq x'_O$ and $b_O = false \rightarrow$ left edge interval.
   The RM sets x'_O equal to x'_e, as it is now the leftmost SV position occluding other positions (O3); sets the b_O equal to true, to indicate that x'_s is occluded (O4); and uses the render mode for left edge intervals (Section III-C3).
2) $x'_s \geq x'_O$ and $b_O = true \rightarrow$ occluded interval.
   Rendering or update of auxiliary variables is not required.
3) $x'_s < x'_O \rightarrow$ right edge or continuous interval.
   The RM sets x'_O equal to x'_s, as samples shifted further right are occluded (O5); sets b_O equal to false to indicate that x'_s is not occluded (O6); and chooses the render mode for nonoccluded SV interval parts by hole detection.

Hole detection is based on the depth difference at the SV interval boundaries x'_s and x'_e. This difference is large in right edge intervals as x'_s and x'_e belong to the FG and the background, respectively; in continuous intervals, they belong to the same object and the depth difference is small. For simplification, hole detection distinguishes between both cases based on the SV interval length $x'_e - x'_s$ (H1), which is proportional to the depth difference. If $x'_e - x'_s$ is greater than 2, the RM assumes a right edge (Section III-C4) and otherwise a continuous interval (Section III-C2).



```
Continuous
$\dot{x}'_s = \mathrm{ceil}(x'_s)$
$\dot{x}'_e = \dot{x}'_O - 1$
for $\dot{x}'$ in $[\dot{x}'_s, \dot{x}'_e]$
    Get $\hat{x}$ with (5)                         R2
    Ren($\dot{x}'$, $\hat{x}$,0)
$\dot{x}'_O = \dot{x}'_s$                          R1

Left Edge
$\dot{x}'_{FL} = \mathrm{round}(x'_e)$
if $\dot{x}'_{FL} \neq \dot{x}'_O$                 R3
    Ren($\dot{x}'_{FL}$,$4 \cdot x'_e$,0 )
    $\dot{x}'_O = \dot{x}'_{FL}$                   R1

Right Edge
$\dot{x}'_s = \mathrm{ceil}(x'_s)$
$\dot{x}'_e = \dot{x}'_O - 1$
$\dot{x}'_{FR} = \mathrm{round}(x'_s)$
for $\dot{x}'$ in $[\dot{x}'_{FR}+1, \dot{x}'_e]$
    Ren($\dot{x}'$,$4 \cdot x'_e$,1)               R4
if $\dot{x}'_{FR} == \dot{x}'_s$                   R5
    Ren($\dot{x}'_{FR}$,$4 \cdot x'_s$,0)
$\dot{x}'_O = \dot{x}'_s$                          R1

Margin
$\dot{x}'_s = \mathrm{ceil}(x'_s)$
$\dot{x}'_e = w$
for $\dot{x}'$ in $[\dot{x}'_s, \dot{x}'_e]$
    Ren($\dot{x}'$,$4 \cdot w$,1)                  R6
$\dot{x}'_O = \dot{x}'_s$                          R1

Ren($\dot{x}'$, $\hat{x}$, $h$)
$s'_{T,l}(\dot{x}') = \hat{s}_{T,l}(\hat{x})$      R7
$s'_{D,l}(\dot{x}') = s_{D,l}(\hat{x}/4)$          R8
$s'_{H,l}(\dot{x}') = h$
Combine $s'_T(\dot{x}')$                           R9
```

Fig. 6. Render modes for an SV interval. Ren($\dot{x}'$, $\hat{x}$, h) maps a sample from the upsampled IV texture to the current SV position $\dot{x}'$ (R7). The IV depth is mapped similarly, but for simplification only with integer precision (R8).

In conclusion, the RM selects a render mode for an SV interval based on its boundaries and auxiliary variables only. This way, information from state variables related to other intervals is not necessary and no complex z-buffering method is required to detect occlusions.

C. SV Interval Rendering Step

The render mode selection step has identified the SV interval type. The RM can now adapt to this type to derive the SV texture samples at SV interval positions that are not occluded. How this is done by interval type specific render modes is shown in Fig. 6 and described in the following.

1) Occluded SV Interval Parts: As discussed before, the right-hand side of continuous and right edge intervals can be occluded. In this case, the occluding SV positions have already been rendered in previous iterations as the RM processes the IV from right to left (see the Appendix). They should not be overwritten when rendering the current SV interval. For this, the RM uses an auxiliary variable $\dot{x}'_O$, which tracks the leftmost integer SV position $\dot{x}'$ rendered in iterations before (R1 in Fig. 6) and thus corresponds to x'_O but in integer precisions. The RM must only render integer SV positions $\dot{x}'$ of the current SV interval that are on the left of $\dot{x}'_O$ [27].

2) Continuous Intervals: In this mode, the SV interval $[x'_s, x'_e]$ boundaries belong to the same object. The RM must derive the nonoccluded samples in $[x'_s, x'_e]$, which are at integer positions $\dot{x}'$ in the interval

$$[\dot{x}'_s, \dot{x}'_e] = [\mathrm{ceil}(x'_s), \dot{x}'_O - 1]. \tag{4}$$

To interpolate them, a renderer could, in general, employ information of both sides of $[x'_s, x'_e]$, since they belong to the same object. However, the RM cannot do this, as it operates intervalwise and does not know SV samples that are warped to the left of x'_s yet. To resolve this, the RM interpolates the IV and maps positions in the SV interval to the interpolated samples in the IV. More specifically, on initialization, the RM quadruples the horizontal sampling rate of the IV texture $s_{T,l}$ to derive an upsampled texture $\hat{s}_{T,l}$. Then, while rendering, the RM derives the value of a sample in $[\dot{x}'_s, \dot{x}'_e]$ by mapping its SV position $\dot{x}'$ to a position $\hat{x}$ in the corresponding IV interval in $\hat{s}_{T,l}$ and setting $s'_{T,l}(\dot{x}')$ to $\hat{s}_{T,l}(\hat{x})$ (R2 and R7). The RM maps positions with (5), so that the relative position of $\hat{x}$ in the interval in $\hat{s}_{T,l}$ is equal to the relative position of $\dot{x}'$ in the SV interval

$$\hat{x} = 4 \cdot \left(\frac{\dot{x}' - x'_s}{x'_e - x'_s} + x_s \right). \tag{5}$$

In conclusion, the interpolation mode has the advantage that computational complex interpolation is only required once for initialization of the RM and not during re-rendering.

3) Left Edge Interval: In this mode, the left SV interval boundary x'_e is the leftmost position of an FG object, which is, in contrast to the remaining SV interval positions, not occluded. Consequently, the RM must render the related SV integer position $\dot{x}'_{FL} = \mathrm{round}(x'_e)$ correctly. The leftmost integer position already rendered is $\dot{x}'_O$. Therefore, if $\dot{x}'_{FL}$ is equal to $\dot{x}'_O$, then the RM has already rendered the SV sample at $\dot{x}'_{FL}$ in the last iteration. However, when $\dot{x}'_{FL}$ is not equal to $\dot{x}'_O$ (R3), the RM sets the SV sample at $\dot{x}'_{FL}$ to $s_{T,l}(x_e)$ [i.e., $\hat{s}_{T,l}(4 \cdot x_e)$]. This way, the fractional left FG edge position is correctly mapped to the integer SV sampling grid [21], which has an effect similar to boundary aware splatting [20].

4) Right Edge Interval: In this mode, x'_s is the rightmost SV position of an FG object, whereas x'_e belongs to a background object. To derive SV samples within the integer SV boundaries given by (4), the RM must fill the disocclusion and render the right FG edge correctly. The rightmost integer FG object position is given by $\dot{x}'_{FR} = \mathrm{round}(x'_s)$. Therefore, the disocclusion is in the SV interval $[\dot{x}'_{FR} + 1, \dot{x}'_e]$. To fill it, the RM extrapolates SV samples from the background sample $s_{T,l}(x_e)$ at the right (R4). Then, the RM examines whether the left integer SV interval boundary $\dot{x}'_s$ is equal to the rightmost integer FG object position $\dot{x}'_{FR}$ (R5). If this is true, $\dot{x}'_{FR}$ belongs to the current SV interval and the RM sets the SV sample at $\dot{x}'_{FR}$ to $s_{T,l}(x_s)$. Otherwise, $\dot{x}'_{FR}$ belongs to the next SV interval and will be rendered later. This way, the RM provides an estimate for the hole and maps the right FG edge positions correctly to the integer positions.

5) Margin Interval: To fill the gap that occurs to the right-hand side of $x'_s = \mathrm{f}(w)$, the RM sets the SV sample values in the margin interval $[\dot{x}'_s, w]$ equal to value of rightmost IV sample at w (R6), so that a rough estimate for the gap is provided.

6) Signals for View Combination: In right edge and margin intervals, some SV texture samples are extrapolated from their neighbors and thus unreliable. To identify them later, the RM derives two signals while rendering an SV interval (R8). The disocclusion flags $s'_{H,l}$ indicate whether an SV texture sample is extrapolated by hole or margin filling; and the SV depth $s'_{D,l}$ represents the depth of SV texture samples. Both signals are used to replace unreliable SV samples in view combination (R9).

D. View Combination Step

To replace unreliable samples in SV texture $s'_{T,l}$, it is common practice to combine $s'_{T,l}$ rendered from a left IV, with a second SV texture $s'_{T,r}$. $s'_{T,r}$ is rendered from a right IV, so that it contains information not available in $s'_{T,l}$. The RM does this similar to [20] based on the SV depths $s'_{D,l}(\dot{x}')$

and $s'_{D,r}(\dot{x}')$, and the disocclusion flags $s'_{H,l}(\dot{x}')$ and $s'_{H,r}(\dot{x}')$. In contrast to [20], the RM derives $s'_T(\dot{x}')$ instantly after the corresponding sample $s'_{T,l}(\dot{x}')$ has been derived as follows.

1) When only $s'_{T,l}(\dot{x}')$ or only $s'_{T,r}(\dot{x}')$ is disoccluded, take the other.
2) When both are disoccluded, take the background sample to avoid that FG samples occur in the disocclusion.
3) When no sample is disoccluded, compare their depth difference $d_\Delta = |s'_{D,l}(\dot{x}') - s'_{D,r}(\dot{x}')|$ to a threshold $d_{\text{th}} = 0.3 \cdot (d_{\max} - d_{\min})$ with $d_{\max}$ and $d_{\min}$ denoting the maximal and minimal possible depth value [20].
   a) If $d_\Delta > d_{\text{th}}$ is true, take the FG sample to not impair the FG object.
   b) Otherwise, combine them, as both are reliable.

In case 3b), the sample of the SV texture rendered from the closer IV view is more reliable. The RM thus derives the weighted average of both samples as follows [20]:

$$s'_T(\dot{x}') = s'_{T,l}(\dot{x}') + \left[s'_{T,r}(\dot{x}') - s'_{T,l}(\dot{x}')\right] \cdot \frac{x'_V - x_{V,r}}{x_{V,l} - x_{V,r}} \quad (6)$$

with x'_V, $x_{V,l}$, and $x_{V,r}$ denoting the horizontal camera positions of the SV, the left IV, and the right IV, respectively.

IV. Partial Re-Rendering and SVDC Calculation

So far, we discussed rendering of the complete SV by processing the complete IV depth map $s_{D,l}$. However, re-rendering only the SV region $\tilde{B}'$ that changes when the IV region B in $s_{D,l}$ changes to $\tilde{s}_B$ is the basic idea of the RM. We thus present RM modifications in this section, which enable such partial re-rendering. In particular, they enable re-rendering from a random access position in the IV depth map (Section IV-A) and stopping at a position ensuring that $\tilde{B}'$ has been updated entirely (Section IV-C). To get an insight how these positions can be derived, we provide an analysis of $\tilde{B}'$ in Section IV-B. Finally, we discuss how the RM can exploit partial re-rendering for SVDC calculation (Section IV-D).

A. Recovery of Auxiliary Variables for Random Access

As presented in Section III-B, the RM employs three auxiliary variables: x'_O and $\dot{x}'_O$ track the leftmost fractional and integer SV positions that have already been processed; b_O indicates whether the left boundary of the last and thus the right boundary of the current SV interval is occluded. These variables are initialized when the RM starts at the right picture boundary w and then continuously updated while iterating. However, when the RM starts re-rendering at a position not equal to w, they are unknown.

To resolve this and recover them, we modified the RM, as shown in Fig 7. One of the modifications derives an occlusion signal $s_{O,l}$. For this, the RM sets $s_{O,l}(x_s)$ equal to x'_O as it is when starting the iteration for x_s (A2). Another modification, called recovery process (A3), then employs $s_{O,l}$ to derive x'_O, $\dot{x}'_O$ and b_O at random access at a particular position x_s.

When re-rendering starts at x_s, the first IV interval to be processed is $[x_s, x_e]$. The recovery process must consequently determine whether the right boundary $x'_e = \mathrm{f}(x_e)$ of the corresponding SV interval is occluded to derive b_O. This is

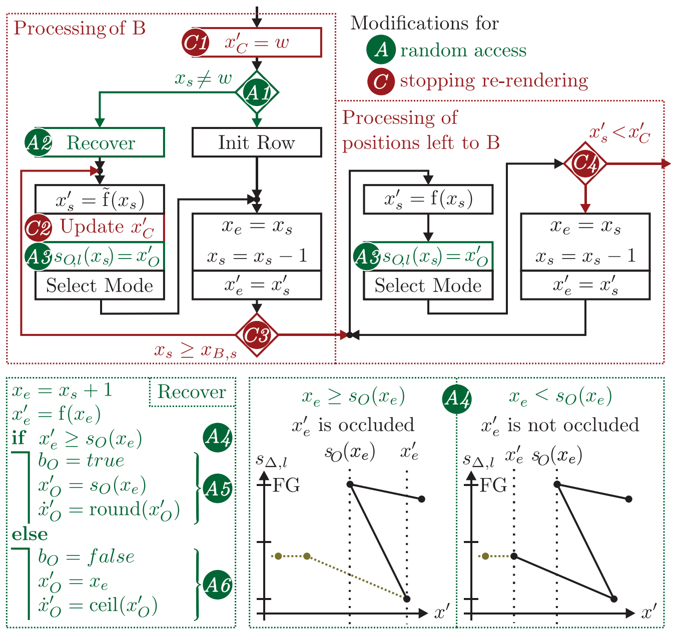


Fig. 7. Top: modifications for partial re-rendering. Bottom: recovery of auxiliary variables b_O and x'_O at random access at x_s.

done by comparing x'_e to $s_O(x_e)$, as $s_O(x_e)$ was the minimal occluding position before x'_e was rendered (A4). When x'_e is greater than or equal to $s_O(x_e)$, x'_e is occluded by an FG object that ends at $s_O(x_e)$. Consequently, the RM raises the flag b_O, sets x'_O equal to $s_O(x_e)$, and finally, in order to regard the correct position of the left FG edge, $\dot{x}'_O$ to round(x'_O) (A5). When x'_e is less than $s_O(x_e)$, x'_e is not occluded. The RM thus sets the flag b_O to zero, and finally x'_O and $\dot{x}'_O$ to x'_e and ceil(x'_e), respectively, since these are the minimal fractional and integer SV position already rendered (A6).

This way, the RM can recover all three auxiliary variables at random access by only storing x'_O while rendering. Consequently, required memory bandwidth is reduced.

B. Determination of the Exact Changed SV Region $\tilde{B}'$

In Section IV-A, we explained how the RM can start re-rendering at a random access position. The exact start and stop position for re-rendering the whole SV region $\tilde{B}' = [x'_{C,s}, x'_{C,e}]$ to be changed when the IV region $B = [x_{B,s}, x_{B,e}]$ changes to the depth candidate $\tilde{s}_B$ (and thus the IV depth from $s_{D,l}$ to $\tilde{s}_{D,l}$) is determined in this section. In the analysis, we first neglect rounding to the SV integer grid.

The change of the IV depth has two effects on the SV. First, SV samples are removed from their old positions, and second, they are moved to new positions. To identify these positions, we introduce two other regions in Fig. 8. The first is the old SV region B'_S, which includes the old sample positions. Its boundaries $x'_{S,s}$ and $x'_{S,e}$ are, therefore, given by the left- and the rightmost positions that are obtained by shifting samples of B with the unchanged IV depth $s_{D,l}$ as follows:

$$B'_S = [x'_{S,s}, x'_{S,e}] = [\min_{x \in B} \mathrm{f}(x), \max_{x \in B} \mathrm{f}(x)]. \quad (7)$$

The second, the new SV region B'_S, includes the new sample positions. It is derived similar to B'_S, but by shifting samples

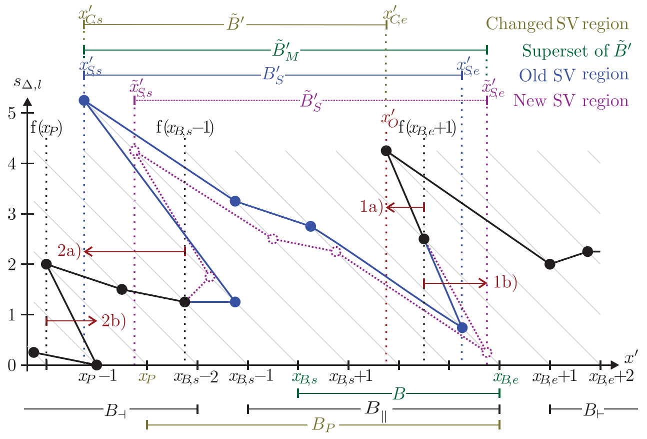


Fig. 8. SV and IV regions related to the IV depth change in B: $\tilde{B}'$ changes in the SV. $\tilde{B}'_M$ is a superset of $\tilde{B}'$. B'_S and $\tilde{B}'_S$ include the samples shifted from B before and after the depth change, respectively. $B_\parallel$ needs to be processed to re-render directly affected SV intervals. B_P is processed to re-render $\tilde{B}'$.

of B with the changed IV depth $\tilde{s}_{D,l}$, which is denoted by $\tilde{\mathrm{f}}$:

$$\tilde{B}'_S = [\tilde{x}'_{S,s}, \tilde{x}'_{S,e}] = [\min_{x\in B} \tilde{\mathrm{f}}(x), \max_{x\in B} \tilde{\mathrm{f}}(x)]. \tag{8}$$

Obviously, $B'_S \cup \tilde{B}'_S$ is a subset of SV positions that may change. However, rendering of an SV interval depends on both of its boundaries, i.e., when the depth at $x_{B,s}$ and $x_{B,e}$ changes, the SV intervals related to $[x_{B,s}-1, x_{B,s}]$ and $[x_{B,e}, x_{B,e}+1]$ are affected. Regarding this additionally, the SV region that might change is $\tilde{B}'_M = [\tilde{x}'_{M,s}, \tilde{x}'_{M,e}]$ with

$$\tilde{x}'_{M,s} = \min\left(x'_{S,s}, \tilde{x}'_{S,s}, \mathrm{f}(x_{B,s}-1)\right) \tag{9}$$
$$\tilde{x}'_{M,e} = \max\left(x'_{S,e}, \tilde{x}'_{S,e}, \mathrm{f}(x_{B,e}+1)\right). \tag{10}$$

In summary, $\tilde{B}'_M$ is the region in that SV samples may depend on the IV depth change in B. However, IV samples outside B can also be warped to $\tilde{B}'_M$, as we found in Section II, and their SV intervals may overlap with and also affect samples in $\tilde{B}'_M$. In order to re-render $\tilde{B}'_M$, it must thus be answered: 1) which IV positions need to be processed to re-render SV intervals directly related to the depth change in B and 2) which IV positions need to be processed additionally to regard overlaps of their SV intervals with $\tilde{B}'_M$. The answer to 1) is that processing IV positions in $B_\parallel = [x_{B,s}-1, x_{B,e}]$ is sufficient, as an SV interval is rendered when its left IV boundary is processed. To answer 2) and so to find where to start and stop in the IV, we analyze overlaps in the following.

1) Start Position: To find the position to start, we analyze whether the RM needs to process any IV positions that is on the right-hand side of $B_\parallel$ in the region $B_\vdash = [x_{B,e}+1, w_B]$. This could be true when processing $B_\parallel$ would alter SV position that are also related to $B_\vdash$. When starting at $x_{B,e}$, the right boundary of its related SV interval is $\mathrm{f}(x_{B,e}+1)$. Taking this as reference position, SV intervals of $B_\parallel$ and $B_\vdash$ can only overlap when one or both of the following cases occur.

1a) IV samples in $B_\vdash$ are shifted to the left of $\mathrm{f}(x_{B,e}+1)$.
1b) IV samples in $B_\parallel$ are shifted to the right-hand side of $\mathrm{f}(x_{B,e}+1)$.

In case 1a), SV intervals of $B_\vdash$ [e.g., (Fig. 8) $x_{B,e}+2$] occlude SV intervals of $B_\parallel$ on the left of $\mathrm{f}(x_{B,e}+1)$. The occluding SV intervals of $B_\vdash$ are, however, not altered when processing $B_\parallel$, as the minimal occluding position x'_O, which is recovered at random access at $x_{B,e}$ excludes them from being re-rendered. Case 1b) occurs when $x'_{S,e}$ or $\tilde{x}'_{S,e}$ [e.g., (Fig. 8) $x_{B,e}$] is greater than $\mathrm{f}(x_{B,e}+1)$. Then, parts of the old or new SV region, and thus SV intervals of $B_\parallel$, are occluded by SV intervals of $B_\vdash$. However, as in case 1a), the SV intervals of $B_\vdash$ are preserved by using x'_O.

In conclusion, as in both cases processing $B_\parallel$ does not alter occluding SV intervals of $B_\vdash$, re-rendering can start at $x_{B,e}$.

2) Stop Position: After starting at $x_{B,e}$, the RM processes remaining IV positions in $B_\parallel$. The question is whether it needs to continue with IV positions in $B_\dashv = [1, x_{B,s}-2]$. This could be true when processing $B_\parallel$ alters SV position that are also related to $B_\dashv$, which can be in case that the following holds.

2a) IV samples in $B_\parallel$ are shifted to the left to $\mathrm{f}(x_{B,e}-1)$.

Case 2a) occurs when the old or the new SV region boundary $x'_{S,s}$ or $\tilde{x}'_{S,s}$ is less than $\mathrm{f}(x_{B,s}-1)$. Then, parts of the old or new SV region, and thus SV intervals of $B_\parallel$, occlude SV intervals of $B_\dashv$. To analyze this, we distinguish two subcases.

In the first subcase, $x'_{S,s}$ is on the left of $\tilde{x}'_{S,s}$ [e.g., (Fig. 8) $x_{B,s}+1$]. Thus, SV intervals of $B_\dashv$ that overlap with $[x'_{S,s}, \tilde{x}'_{S,s}]$ have been occluded before by SV intervals of $B_\parallel$ and become visible. To re-render them, the RM needs to continue until it has processed a position x_P with $\mathrm{f}(x_P) < x'_{S,s}$.

In the second subcase, $x'_{S,s}$ is on the right-hand side of $\tilde{x}'_{S,s}$ (e.g., imagine Fig. 8 with B'_S and $\tilde{B}'_S$ swapped). Then, SV intervals of $B_\dashv$ that overlap with $[\tilde{x}'_{S,s}, x'_{S,s}]$ have been visible previously and become occluded by SV intervals of $B_\parallel$. Thus, to re-render s'_T only, processing could be stopped at the position x_s in $B_\parallel$ with $\tilde{\mathrm{f}}(x_s) = \tilde{x}'_{S,s}$. However, as the SV region $[\tilde{x}'_{S,s}, x'_{S,s}]$ is occluded now, its occlusion signal $s_{O,l}$ changes and needs to be updated. Therefore, the RM needs to continue, until a position x_P that is not occluded by $\tilde{B}'_S$ has been processed, which is given at the position x_P with $\mathrm{f}(x_P) < \tilde{x}'_{S,s}$.

In summary, the subcases show that, after processing $x_{B,e}-1$, the RM needs to continue until a position x_P with $\mathrm{f}(x_P) < \min(x'_{S,s}, \tilde{x}'_{S,s})$ has been processed. Having reached x_P (which can also be equal to $x_{B,e}-1$), a further overlap occurs when the following is true.

2b) IV positions in $[1, x_P-1]$ are shifted to the right-hand side of $\mathrm{f}(x_P)$

[e.g., (Fig. 8), x_P-1]. However, since related SV intervals have been and are occluded, processing of positions in $[1, x_P-1]$ is not required and the RM can stop processing.

3) Conclusion: Discussed cases show that the RM needs to process $B_\parallel$ first. Then, it must continue in $B_\dashv$ until position x_P with $\mathrm{f}(x_P) < \min(x'_{S,s}, \tilde{x}'_{S,s})$ has been processed. [2] Therefore, the IV region to be processed for re-rendering $\tilde{B}'$ is

$$B_P = [\min(x_P, x_{B,s}-1), x_{B,e}]. \tag{11}$$

[2]It should be noted that although we neglected rounding to the integer SV grid, it can be easily shown that this condition is still sufficient when rendering with quarter sample precision and rounding half sample position up.

When processing B_P, the whole changed SV region $\tilde{B}'$ is re-rendered. Cases 1a) and 1b) show that $\tilde{B}'$ is a subset of $\tilde{B}'_M$ as its right-hand side can be occluded. More specifically, with the minimal occluding position $s_O(x_{B,e})$ (as derived in Section IV-A), $\tilde{B}'$ is equal to $[\tilde{x}'_{M,s}, s_O(x_{B,e})]$.

C. re-rendering the Changed Region $\tilde{B}'$

In Section VII, we found B_P as the IV region that needs to be processed to re-render the changed region $\tilde{B}'$. How to start at $x_{B,e}$ was discussed in Section IV-A. The question is now, how the RM can find x_P to stop re-rendering there. To answer this, Fig. 7 shows further modifications.

As shown on the left, re-rendering starts with *processing B*. One of the modifications there is the derivation of the minimal changed position x'_C. For initialization, x'_C is set equal to w (C1). In subsequent iterations, x'_C is updated (C2) to

$$x'_C = \begin{cases} \min(\tilde{\mathrm{f}}(x_s), x'_C) & \text{if } \tilde{s}_{D,l}(x_s) > s_{D,l}(x_s) \\ \min(\mathrm{f}(x_s), x'_C) & \text{otherwise.} \end{cases} \quad (12)$$

With (1), (7), and (8), it can easily be shown that the iterative derivation using (12) results in x'_C equal to $\min(x'_{S,s}, \tilde{x}'_{S,s})$, after the RM has processed B (C3).

Then, the RM continues with *processing positions on the left of B'*, as shown on the right-hand side in Fig. 7. In the first iteration, the RM processes $x_{B,s} - 1$. Subsequently, the RM evaluates the condition $x'_s < x'_C$ (C4). If this is true, x_s is equal to x_P and the RM stops processing. Otherwise, it continues iterating.

This way, the RM can derive x'_C, which is required to find x_P and stop directly while re-rendering. Consequently, a prior analysis of B to find x_P is not necessary.

D. SVD and SVDC Calculation Step

This section discusses how the RM can use partial re-rendering for its major purpose—the quantification of the SVDC that is related to a depth candidate. As we discussed in Section II, this quantification is enabled by two modes, the *SET* and the *GET* mode. In both, a depth candidate $\tilde{s}_B$ for an IV region B is provided to the RM, either to adopt it (S4) or to calculate the related SVDC (G4).

To this end, the RM re-renders the changed SV region $\tilde{B}'$ (S2, G2) related to the depth candidate by iteratively processing the IV region B_P. In each iteration, the RM renders zero (e.g., f' in Fig. 4) or more changed SV texture samples $\tilde{s}'_T(\dot{x}')$. Directly after a sample has been rendered [i.e., after (R9)], the RM computes its SVD $\tilde{s}'_E(\dot{x}') = [\tilde{s}'_T(\dot{x}') - s'_{Ref}(\dot{x}')]^2$ (G3, S3). In the *SET* mode, the RM stores $\tilde{s}'_E(\dot{x}')$ as $s'_E(\dot{x}')$ (S4). In *GET* mode, the stored SVD is used to derive the SVDC according to G4, by incrementing ΔD, which is set to zero when re-rendering starts (C1), by

$$\Delta D(\dot{x}') = \left(\left[\tilde{s}'_T(\dot{x}') - s'_{Ref}(\dot{x}')\right]^2 - s'_E(\dot{x}')\right). \quad (13)$$

This way, when the RM reaches position x_P and stops re-rendering, $\tilde{B}'$ has been entirely re-rendered, so that ΔD is the SVDC related to the depth candidate $\tilde{s}_B$.

V. Evaluation

A. Complexity Analysis

As we target a low-complexity RM design, we analyze how the RM's complexity is related to different functionalities and depends on the input depth data. What we neglect is the complexity of the RM initialization, i.e., steps I1 and I2, as it is insignificant when the RM performs a huge number of *SET* and *GET* operations. Evaluation results are discussed in Section VI, in light of different application scenarios.

1) Complexity of Different RM Setups and Modes: The RM design enables rendering with a subset of functionalities (in the following called setup). This can avoid overfitting or reduce complexity. Moreover, rendering can be applied in *GET* and *SET* mode. Which complexity is added by which functionality and how modes differ in complexity is thus of interest and evaluated in this section.

a) Methodology: When re-rendering small IV regions B, the overhead for the recovery process (Section IV-A) can become significant. Furthermore, when $\tilde{s}_B$ is a large depth change, an overhead for processing positions on the left of B (Section IV-C) occurs. Both effects are evaluated in Section V-A2. To avoid them here, *GET* and *SET* operations are performed with B corresponding to the full size initial IV depth. This way, operations related to the depth candidate's characteristics are insignificant in the evaluation.

For evaluation, we employed two different methods. First, we counted the number of operations and memory accesses that are conceptually needed per IV depth sample in a simulation. This provides a platform independent measure. Second, we measured the number of IV depth samples that can be processed per second on a PC system. For both, the IVs were at stereo distance to the SV and we averaged over eight sequences provided by JCT-3V [25] (listed in Table III and in the following called JCT-3V sequences). Results for different setups and modes are provided in Table II and show which kind and number of operations and memory accesses are performed. Their total number is denoted as N_T. Sample rates are denoted with R_S. We show only averages, as results depend only slightly on the sequence content. In all sequences, the majority of intervals are continuous intervals, which thus determine the complexity. Large deviations might only occur for uncommon sequences with large occlusions.

b) RM setups: Evaluated setups are shown in the top row of Table II. The *base setup* supports extrapolation of an IV luma component with integer sample precision. On top, this distortion computation, quarter sample precision, and view combination are added in the *extended setup*. In addition, two chroma components with the same resolution as the luma component are rendered in the *full setup*. For reference, we used a calculation of the SSD.

Table II shows that compared with *SSD*, N_T (averaged over both modes) for the base, extended, and full setup is increased by the factors of 2.5, 5.5, and 8, respectively. These results are supported by the sample rate measurement. Compared with *SSD*, R_S (averaged over both modes) of the three setups is divided by 2.6, 5.7, and 8.6, respectively. Considering that full rendering is performed, these factors are relative low.

TABLE II

AVERAGE NUMBER OF OPERATIONS AND MEMORY ACCESSES PER DEPTH SAMPLE, AND SAMPLE RATE DEPENDING ON THE RM SETUP

	Abb.	SSD[c]	Base[d]	+Dist[e]	+QPel[f]	+Com[g]	Ext.	+Chr.[h]	Full
Addition or subtraction	AD	2	2	+3	+2	+2.9	9.9	+7.9	17.8
				+1			7.9		15.8
Unary AD	UA	2	3		+1		4		4
Comparison	CP	1	9			+3	12		12
Unary shift	US	0	0		+3		3	+1	4
Multiplicat.	ML	1	0	+1			1	+2	3
Tab. look-up	LT	0	1[i]		+1[j]	+1[k]	3	+1.9	4.9
Memory read[a]	MR	2	3	+2		+3	8	+5.9	13.9
				+1			7		12.9
Memory write[a]	MW	0	0	+0		+0	0	+0	0
			3	+1		+3	7	+4	11
Total	N_T	8	18	+6	+7	+9.9	40.9	+18.7	59.6
			21	+4		+12.9	44.9	+22.7	67.6
Sample rate[b]	R_S	186	102	51	50	34	31	23	23
			38	37	37	31	26	20	20

In cells with two values the upper is related to *GET* and the lower to *SET*.

[a] Access to variables in the current flow that can be assumed to be locally available are not regarded (e.g. results of previous operations, LUT access).

[b] In 10^6sampels/s. On *Intel Xeon X5472*, 3 GHz, 8 GByte RAM,

[c] 2 UA, 1 CP: iterate over $\tilde{s}_B$, $s_{D,l}$; 2 MR: $\tilde{s}_B$, $s_{D,l}$; 1 AD, 1 ML: derive squared difference; 1 ADD: sum up squared differences.

[d] 3 UA: decrement the current positions in $\tilde{s}_B$, $s_{D,l}$, $s'_{T,l}$; 2 AD: calculate x'_s (1), $x'_e - x'_s$; 3 CP: most probable path (O1),(H1),(C3); 4 CP: check region and picture size limits; 2 CP: determine x'_C; 1 LT: $s_{D,l}$ to $s_{\Delta,l}$; 3 MR: $\tilde{s}_{D,l}$, $\hat{s}_{T,l}$, $s_{D,l}$; In *SET* mode 3 MW: $s_{D,l}$, $s'_{T,l}$, $s_{O,l}$.

[e] 1 MR: s'_{Ref}; 1 AD, 1 ML: derive s'_E; In *SET* mode 1 MW: s'_E; In *GET* mode 1 MR: old s'_E, 1 AD: derive $\Delta D(x')$, 1 AD: sum up $\Delta D(x')$.

[f] 2 AD, 1 US, 1 LT: for (5); 1 UA, 2 US: round quarter to integer accuracy.

[g] 3 MR: $s'_{T,r}$, $s'_{D,r}$, $s'_{H,r}$; 1 AD: derive $s'_{D,l} - s'_{D,r}$; 3 CP: logic in Section III-D; 1.9 AD, 1 LT: (6); In *SET* mode 2 MW: $s'_{D,l}$, $s'_{H,l}$.

[h] For view combination and SVDC calculation of two chroma components.

[i] To convert $s_{D,l}$ to $s_{\Delta,l}$ [i.e. (2) and (3) for (1)] with a table look-up.

[j] As x'_e and x'_s are given in quarter sample precision and their difference is not greater than 2 for a Continuous Interval, the RM avoids the division in (5) by using a 2D look-up table with $\dot{x}' - x'_s$ and $x'_e - x'_s$ inputs.

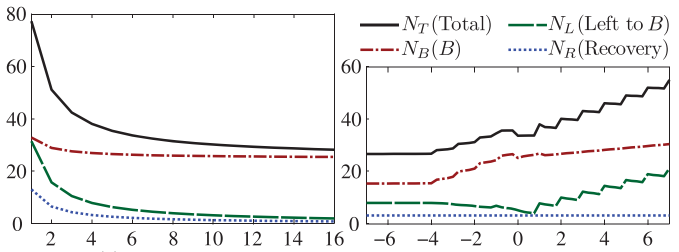
[k] As the right factor in (6) is constant for a particular SV, the RM avoids the division and the multiplication by using a table look-up.

TABLE III

(A) RM VERSUS SVD ESTIMATION. (B) RM VERSUS RENDERING METHODS

Correlation Coefficient [%]										SV texture vs. Original					
RM			*1D-Fast*			*VSRS*				PSNR [dB]			SSIM		
RM	Kim	VSD	RM	Kim	VSD	RM	Kim	VSD		*RM*	*1D-F.*	*VSRS*	*RM*	*1D-F.*	*VSRS*
100	73	71	**92**	68	64	**86**	62	59	*Poznanhall2*	**36.3**	36.2	34.7	.920	.920	.899
100	92	90	**99**	91	89	**94**	93	88	*Poznanstreet*	**35.4**	35.2	32.3	**.923**	.922	.889
100	81	64	**95**	79	62	**91**	80	63	*Undodancer*	40.2	**40.5**	38.8	.985	.985	.978
100	91	78	**96**	91	75	**93**	91	74	*Gtfly*	42.9	**43.1**	41.1	.984	.984	.981
100	73	70	**98**	73	68	**88**	79	75	*Kendo*	**36.0**	35.9	35.8	.965	.965	.964
100	83	80	**97**	84	80	**88**	87	80	*Balloons*	36.9	36.9	36.7	.965	.965	**.966**
100	63	74	**95**	59	68	**80**	72	79	*Newspapercc*	32.4	32.4	32.0	.934	.934	**.935**
100	91	67	**98**	90	65	92	92	54	*Shark*	**46.1**	45.7	44.4	**.991**	.990	.990
100	81	74	**96**	79	71	**89**	82	71	Average	**38.3**	38.2	37.0	.958	.958	.950
									(C) Average coded→	32.5	32.5	32.1	.888	**.889**	.887

c) RM modes: Table II shows that the *GET* mode does not write to the memory. In contrast, the *SET* mode does to adopt the depth change and thus to update the RM's state. More specifically, it stores the changed state variables that are marked with (*) in Table I. When the *GET* mode renders



(a) Region width w_B (b) Relative disparity change in B

Fig. 9. Average number of operations and memory accesses per IV depth sample depending on (a) w_B and (b) disparity change.

a current interval, these variables change as well. However, their change is irrelevant for rendering further intervals, as this depends only on the old RM state, the auxiliary variables, and the RM input. Therefore, the *GET* mode does not store them and the RM state is preserved.

This way, the RM can test further candidates without reverting its state. Moreover, memory bandwidth is approximately halved, which can also be seen from measured sample rates: R_S increases drastically in the base setup when performing a *GET* instead of the *SET* operation. Since the major difference between both modes is the number of writing memory accesses, the memory bandwidth seems to be the determining factor on the used platform.

2) Dependency from the Depth Candidate: So far, we analyzed re-rendering of a whole SV with an unchanged IV depth. However, re-rendering is usually performed for a small region B with changed depth $\tilde{s}_B$. So it is of interest how such characteristics increase the complexity. To answer this, we analyze how the average number of operations and memory accesses per IV depth sample increases: 1) for the recovery process (N_R); 2) for processing B (N_B); 3) for processing positions on the left of B (N_L); and 4) in total ($N_T = N_B + N_L + N_R$). For evaluation, we used the base setup with quarter sample precision enabled. We used the same view setup and averaged over sequences as before.

a) Dependency from IV region width: To evaluate how complexity depends on the width $w_B = (x_{B,e} - x_{B,s} + 1)$ of B, we first partitioned an IV depth in regions B of size w_B. Then, we performed a *GET* operation for each region B with a depth candidate $\tilde{s}_B$ corresponding to the depth values in B plus an offset leading to a disparity change of 1.

Results in Fig. 9 show that when B corresponds to one IV sample only ($w_B = 1$), the additional complexity ($N_R + N_L$) introduced by the recovery process and by processing samples left of B is even larger than the complexity (N_B) required for processing B. However, N_L and N_R and thus N_T decrease fast with w_B. In conclusion, for wider IV regions ($w_B \geq 8$), the additional complexity is insignificant.

b) Dependency from depth change: A second aspect impacting the complexity is the magnitude of the depth change. For evaluation, we used the same approach as in Section V.A.2.a, with the differences that w_B has been fixed to a value of 4 and the depth value offset and thus disparity was varied.

For negative disparity changes, N_T decreases, as shown in Fig. 9. Reason is that the new SV region $\tilde{B}'_S$ (as shown in Fig. 8) moves to the right in s'_T, consequently samples at the right-hand side of $\tilde{B}'_S$ become occluded. As occluded samples can be skipped while rendering, N_B decreases. Furthermore, as a disocclusion appears on the left of $\tilde{B}'_S$, hole filling increases N_L. When $\tilde{B}'_S$ is entirely occluded, the hole has its maximal size and N_B and N_L remain constant.

For positive disparity changes N_B and N_L and thus N_T increase almost linearly. This is because $\tilde{B}'_S$ is shifted left in s'_T and occludes left neighboring samples. As the additionally occluded positions need to be processed to update the occlusion signal $s_{O,l}$, N_L increases. Furthermore, the hole on the right-hand side of $\tilde{B}'_S$ gets larger and additional operations are required for hole filling, which increases N_B.

In summary, the complexity increases significantly when testing large positive disparity changes.

3) Memory Requirements: The RM stores state variables marked with (*) in Table I as its state. However, since the RM operates rowwise, it accesses only rows that belong to a currently processed IV block. For a horizontal processing order and small block sizes, it can thus easily be derived from Table I that the RM requires memory for less than a picture.

B. Comparison to Alternative Approaches

This section evaluates whether the optimizations of the RM are effective and how the RM compares with other SVD estimation methods and rendering algorithms.

1) Complexity Reduction Due to Key Features: The RM uses several features for optimization and re-rendering. The question is if these features are effective. As we are not aware of other partial depth image-based re-rendering algorithms, we compare the RM to a hypothetical unoptimized variant. For comparison, we assume that the variant and the RM have the functionality of the base setup with distortion calculation in the *GET* mode. In this setup, the RM performs $N_O = 19$ operations and $N_M = 5$ memory accesses per IV sample (Table II). How N_O and N_M increase when a particular RM feature is not supported is very roughly estimated in the following.

a) Partial re-rendering: To enable our main contribution—re-rendering—we suggested: 1) the storage of state variables as basis; 2) the recovery process to start at a random access position; 3) the derivation of the changed region to find out where to stop re-rendering; and 4) intervalwise processing to simplify starting and stopping. Without these features, an unoptimized algorithm would re-render a complete row upon a depth change in a small part of it, since the unoptimized algorithm would neither know whether samples at the start position are occluded, nor where to stop rendering. Assuming a row width of $w = 1024$ and a changed IV region of width $w_B = 16$, the number of processed IV samples and thus N_O and N_M would increase by a factor of $w/w_B = 64$.

b) Intervalwise processing: Per IV depth sample, the RM performs all steps consecutively to derive the related SV samples and the SVD. To enable that, we modified and combined basic rendering techniques (as described in Section I), e.g., instant interpolation, hole filling, view combination, and SVD calculation. A step instantly processes intermediate results of its preceding step, as shown in Fig. 4 by vertical arrows. Crucial for intervalwise processing is the occlusion detection method from [19]: no information of other intervals or z-buffering methods are required to handle occlusions, since the minimal occluding position is tracked while rendering.

Without intervalwise processing, an unoptimized algorithm would process all samples of the change region consecutively in one step, before starting the next step. This way, it would need to store and read additional intermediate results (i.e., sample values before interpolation or hole filling). Moreover, additional operations would be necessary to iterate multiple times over the changed region. Estimating that an unoptimized algorithm performs three additional iterations over the changed region and that each iteration requires one addition and one comparison per sample, N_O increases by a factor of $25/19 \approx 1.3$. Assuming that two additional intermediate results are stored and read, N_M increases by a factor of $9/5 = 1.8$.

c) GET mode: Intervalwise processing enables a further contribution—the *GET* mode. In *GET* mode, the RM can calculate the SVDC mode without changing its state and thus does not perform writing accesses. Table II shows that for an unoptimized algorithm with *SET* mode only, N_M would increase by a factor of $8/5 = 1.6$.

d) Fast interpolation: For quarter sample precision, the RM maps SV positions to an upsampled IV texture, similar to backward warping [28], but embedded by us in the intervalwise processing scheme. In contrast, the *VSRS 3.5* 1D mode warps a complete IV texture, which has been upsampled to obtain an SV texture. This SV texture is then decimated. Assuming that an unoptimized algorithm operates the same way in the base setup and upsamples by a factor of 4, it requires about $4 \cdot 20$ operations and $4 \cdot 3$ memory accesses (Table II) when neglecting decimation. The RM requires about 30 operations and three memory accesses (Base + QPel in Table II) only. Therefore, for the unoptimized algorithm, N_O and N_M increase by factors of $80/30 \approx 2.7$ and $16/4 \approx 4$.

e) Conclusion: As features are orthogonal, the derived factors can be multiplied. Inverting them shows that intervalwise processing, the GET mode and fast interpolation reduce N_O and N_M to about $100/(1.3 \cdot 1 \cdot 2.7) \approx 29\%$ and $100/(1.8 \cdot 1.6 \cdot 4) \approx 9\%$, respectively, and in total N_T to about 19% (as N_M is about one quarter of N_T after reduction). On top of this, the most significant reduction is achieved by our main feature—partial re-rendering—as it avoids rendering of a complete row.

2) Comparison to Estimation Methods: The SVDC provided by the RM is always exact with respect to its rendering algorithm. However, when the receiver-side renderer in the 3D video system and the RM differ, deviations can occur. Then, it is of interest how the SVDC derived by the RM correlates with the SVD at the receiver-side renderer.

We evaluate this in comparison to two existing methods, which are based on the disparity difference $s_p(x) = \tilde{s}_{\Delta,l}(x) - s_{\Delta,l}(x)$ due to the depth distortion. The first has been proposed besides another method by Kim *et al.* [9], [10]. It derives the SSD between the texture samples $s_{T,l}$ in B and the corresponding samples at positions shifted by the disparity difference

$s_p(x)$, i.e., it sums up $[s_{T,l}(x) - s_{T,l}(x - s_p(x))]^2$. The second metric—the View Synthesis Distortion (VSD) [12], [29]—derives a weighted sum of squared disparity differences $s_p(x)$ over $x \in B$ with weights being the square of the horizontal gradient at $s_{T,l}(x)$.

We evaluated the performance of a particular sender-side SVD derivation method M (the RM's *extended setup*, Kim's method, or the VSD) for a receiver-side rendering method R (RM, *1D-Fast* [23], or *VSRS 3.5* [20]) as follows. Similar to [10], we first partitioned several distorted SV textures rendered with method R in horizontal slices with a height of 16 samples. Then, we evaluated the correlation of: 1) the exact SVDs of the slices with respect to the rendering method R and 2) the estimated SVDs of the slices provided by method M. We used horizontal slices instead of blocks as distorted SV regions might be related to multiple IV blocks as discussed (Section II).

More specifically, we used method R to render: 1) distorted SV textures from coded IV texture and coded IV depth and 2) respective reference SV textures from coded IV texture and original IV depth. In doing so, the SV was at stereo distance between the IVs. Then, we computed the exact SVDs as SSDs between the luma components of the distorted and reference SV texture slices. This way, the exact SVDs are the distortions introduced by depth coding when method R is used, thus the distortions that should be provided by method M. To derive the distortions that are actually provided by method M for an SV slice, we applied M to the associated IV data slices (i.e., the respective parts of the coded IV textures and coded and original IV depth). This means, we used method M on the left IV data, on the right IV data, and averaged over both results.

This way, we processed slices of all pictures of all JCT-3V sequences using 3D-HEVC for coding with four different quantization parameters (QPs) according to JCT-3V's test conditions [25], but using the SSD of depth for mode selection in coding to avoid a potential bias. Then, we pooled results of each sequence in two vectors—one for method M and one for the rendering approach R. Finally, we computed the correlation coefficient between the vectors, i.e., we divided their covariance by the product of their variances.

Results in Table III(a) show that when sender and receiver use the RM, the distortions match perfectly as targeted. The correlation decreases the more R deviates from the RM: to 96% for *1D-Fast* and to 89% for *VSRS*.

Kim's method and the VSD show a constant correlation of about 82% and 73%, respectively. However, as discussed in Section V-A, the RM's *extended setup* complexity is increased by a factor of at least 5.5 compared with SSD. Factors for Kim's method and the VSD are only about 1.5–1, respectively, as it can easily be deduced from [9] and [12]. In summary, the RM performs also well when the receiver uses a different rendering algorithm and outperforms the existing estimation methods. However, increased correlation comes at increased computational complexity.

3) Comparison to View Synthesis Methods: The RM's primary objective is SVDC calculation. The calculated SVDC should correlate with the actual SVDC at a receiver-side renderer even when this differs from the RM. For this, the RM only uses basic techniques that are commonly applied. Those techniques should render geometrically correct with sufficient quality when using undistorted depth. The *Undodancer* pictures in Fig. 10 verify this: the picture rendered with the RM is very similar to the original captured picture. For further evaluation, we rendered SV textures using the JCT-3V sequences and compared them to original textures. Results for rendering from original IV data are given in Table III(b) and show that the RM, *1D-Fast*, and *VSRS* perform similarly.

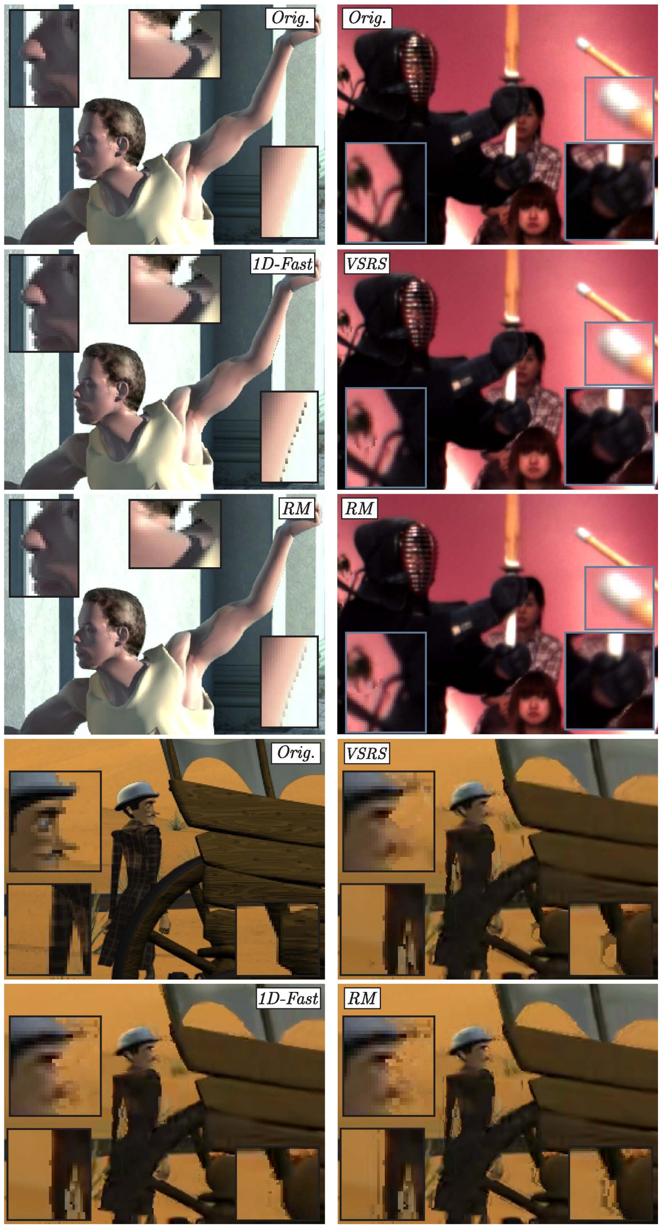


Fig. 10. Originals and pictures rendered from two IVs at stereo distance. Top-left: *Undodancer* (undistorted). Top-right: *Kendo* (imperfect original depth). Bottom: *Gtfly* (strongly coded texture and depth).

To evaluate the RM for strongly distorted IVs, we coded the JCT-3V sequences with 3D-HEVC using JCT-3V's test conditions [25] and the lowest rate point defined there, but using depth SSD for mode selection to avoid a bias to the RM. Results in Table III(c), which are averaged over the sequences, indicate still a similar performance. However, we observed that

VSRS and *1D-Fast* outperform the RM at strongly distorted edges, as shown in Fig. 10 for the *Gtfly* sequence. Reason for this is a boundary noise removal tool, which is not part of the RM and that can conceal SVDs due to distorted depth.

In summary, the RM renders geometrically correct with sufficient quality as targeted. In case of strongly distorted depth, its rendering quality could be improved by boundary noise removal tools. For this, the JCT-3V uses the RM for encoding while applying the *1D-Fast* at the receiver [25].

The RM design primarily targets low complexity for partial re-rendering. However, for completeness, we measured run times for rendering of whole SV textures using the RM (without functionalities for partial re-rendering and SVDC calculation) and available software implementations [22], [28] of the other methods on the PC system also used for results in Table II. Results averaged over the JCT-3V sequences show that the RM, *1D-Fast*, and *VSRS* require 0.34, 0.68, and 1.16 s to render a full HD picture with 4 : 2 : 0 chroma sampling. Results indicate that the RM enables also rendering of whole SV textures with low complexity.

VI. Application Scenarios

In Section VII, we analyzed how the RM can be implemented, which complexity is introduced and which SV quality is reached. What these results mean in different DPAs and how they can use the RM is discussed in this section.

A. Encoding

The RM can be used by a depth encoder to select coding modes based on the SVDC [5]. This is done by the 3D-HEVC reference software encoder [22], which uses the RM's full setup. By default [25], the SVDC is computed in six SVs using reference textures s'_{Ref} rendered from original texture and depth, so that the SVD and thus the SVDC are the distortion introduced by coding only. This ensures a proper quality in the whole viewing range [6], and saves about 19% of the total bit rate compared with conventional encoding using the SSD of the depth for mode selection.

However, rate savings come at the expense of higher computational complexity related to: 1) the full setup of the RM; 2) the depth candidate's characteristics (which is limited, as the w_B is not less than 4 and depth changes are small in predictive coding); 3) additional *SET* operations; and 4) the six SVs. This increased complexity is addressed by two modifications. First, coding modes are preselected using the VSD [29] before selecting finally based on the SVDC. Second, re-rendering is skipped when a change in the SV is unlikely [30]. With these and other modifications [23], the encoding time increases in average by 46% compared with using the SSD of the depth in mode selection. When averaging over sequences coded with the lowest and highest QP, the increase is about 40% and 50%, respectively. Main reason is the RM's complexity dependence from the magnitude of the depth change, which decreases with lower QPs. Further results for compression are provided in our prior works with respect to different receiver-side renderers [5] and to different RM setups and SV positions [6]. A comparison to the SVD is given in [29].

B. Depth Filtering

A depth filter [4] can identify and remove irrelevant information from depth maps by calculating the SVDC using reference textures s'_{Ref} rendered with initial depth. Alternatively, it can refine a depth map by calculating the SVDC using a recorded texture as reference s'_{Ref}, so that the SVDC would be related to the SVD caused by the initial depth. When filtering adaptively, the depth change is generally small, and therefore, the overhead due to its magnitude is minor. For approaches working on small windows, the overhead for partial re-rendering becomes significant.

C. Depth Estimation

Depth estimation is usually based on matching regions in different IVs [3]. The SVDC of an SV texture extrapolated to the position of a recorded reference texture can be an additional term of matching cost. However, N_T increases with the magnitude of depth change and thus in depth estimation with the search region. For this, initial light weight matching would be beneficial, before refinement steps using the SVDC.

D. Interactive Depth Editing Tools

The RM can provide an SV as feedback to a human user of a depth editing tool by applying a *SET* operation. Typically, users interact with low frequency and changes affect a fraction of the depth map only. Therefore, even for large depth changes, re-rendering with the RM requires minimal computational resources.

VII. Conclusion

DPAs can be improved by regarding the SVDC obtained by partial depth image-based re-rendering. For re-rendering, we proposed a fast algorithm—the RM—which operates on a single IV depth sample basis, so that only the related SV interval can be re-rendered. To start re-rendering at a random access position, we extended the RM by a recovery process. Moreover, we analyzed the changed SV region to determine where rendering can be stopped in the IV and modified the RM based on our findings. This way, only the fraction of the SV that is affected by the depth change is re-rendered.

Such partial re-rendering—the RM's main feature—reduces complexity significantly. On top of this, other optimizations (i.e., intervalwise processing, *GET* mode, and fast interpolation) reduce the number of operation and memory accesses to about 20% compared with an unoptimized re-rendering variant. Compared with SSD calculation, the RM's complexity is increased by a factor of 3–9 (depending on its setup). It increases further for large depth changes or small IV region.

The RM provides an exact measure for the introduced SVD when the receiver also uses the RM's rendering algorithm. For other rendering methods, the RM achieves a high correlation ($\geq$89%), which still outperforms the evaluated SVD estimation methods. However, compared with the estimation methods, the computational complexity increases by a factor of about 5.5.

As receiver-side renderer—its secondary use case—the RM renders geometrically correct with a sufficient quality. For distorted depth, a higher quality can be achieved with method supporting boundary noise removal. A comparison to the available software of other methods shows that rendering time is reduced by factors of about 2 and 4.

For SVDC calculation, the RM has proved its applicability in rate-distortion optimization of the 3D-HEVC reference software encoder and can also improve other DPAs, such as depth editing, filtering, or estimation. It is thus a valuable tool in systems with autostereoscopic displays to improve compression and quality of 3D video.

Appendix

We show that an IV sample shifted from the IV position $x-n$, with $n > 0$, is occluded in the SV when $\mathrm{f}(x-n) \geq \mathrm{f}(x)$ is true. We assume that a left IV is processed to render an SV on its right and that background samples on the left of an FG object in the IV do not appear in a hole at the right-hand side of the FG object in the SV. The same is shown in [27] differently.

Proof:

$$\begin{aligned} & \mathrm{f}(x-n) \geq \mathrm{f}(x) \\ \Leftrightarrow\ & x-n-s_{\Delta}(x-n) \geq x-s_{\Delta}(x) && \text{with (1)} \\ \Rightarrow\ & s_{\Delta}(x-n) < s_{\Delta}(x) \\ \Leftrightarrow\ & s_{Z}(x-n) > s_{Z}(x) && \text{with (3).} \end{aligned}$$

This means, the sample at IV position x is located closer to the camera than the sample at IV position $x-n$. Therefore, the sample from IV position $x-n$ is shifted behind an FG object and thus occluded in the SV. ■

Consequently, when processing the IV from right to left, a rendered SV sample will never be occluded by any SV sample rendered later and can thus be regarded as final [27].

References

[1] A. Smolic, K. Müller, P. Merkle, P. Kauff, and T. Wiegand, "An overview of available and emerging 3D video formats and depth enhanced stereo as efficient generic solution," in *Proc. Picture Coding Symp. (PCS)*, Chicago, IL, USA, May 2009, pp. 1–4.

[2] C. Fehn, "Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV," *Proc. SPIE*, vol. 5291, pp. 93–104, May 2004.

[3] D. Scharstein, R. Szeliski, and R. Zabih, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," in *Proc. IEEE Workshop Stereo Multi-Baseline Vis.*, Kauai, HI, USA, Dec. 2001, pp. 131–140.

[4] S. Smirnov, A. Gotchev, and K. Egiazarian, "Methods for depth-map filtering in view-plus-depth 3D video representation," *EURASIP J. Adv. Signal Process.*, vol. 2012, no. 1, pp. 1–21, Dec. 2012.

[5] G. Tech, H. Schwarz, K. Müller, and T. Wiegand, "3D video coding using the synthesized view distortion change," in *Proc. Picture Coding Symp. (PCS)*, Kraków, Poland, May 2012, pp. 25–28.

[6] G. Tech, H. Schwarz, K. Müller, and T. Wiegand, "Synthesized view distortion based 3D video coding for extrapolation and interpolation of views," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, Melbourne, VIC, Australia, Jul. 2012, pp. 634–639.

[7] W.-S. Kim, A. Ortega, P. Lai, D. Tian, and C. Gomila, "Depth map distortion analysis for view rendering and depth coding," in *Proc. 16th IEEE Int. Conf. Image Process. (ICIP)*, Cairo, Egypt, Nov. 2009, pp. 721–724.

[8] H. Yuan, S. Kwong, J. Liu, and J. Sun, "A novel distortion model and Lagrangian multiplier for depth maps coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 3, pp. 443–451, Mar. 2014.

[9] W.-S. Kim, A. Ortega, P. Lai, D. Tian, and C. Gomila, "Depth map coding with distortion estimation of rendered view," *Proc. SPIE*, vol. 7543, p. 75430B, Jan. 2010.

[10] W.-S. Kim, A. Ortega, P. Lai, and D. Tian, "Depth map coding optimization using rendered view distortion for 3D video coding," *IEEE Trans. Image Process.*, vol. 24, no. 11, pp. 3534–3545, Nov. 2015.

[11] R. Ma, N.-M. Cheung, O. C. Au, and D. Tian, "Novel distortion metric for depth coding of 3D video," in *Proc. 20th IEEE Int. Conf. Image Process. (ICIP)*, Melbourne, VIC, Australia, Sep. 2013, pp. 1714–1718.

[12] B. T. Oh, J. Lee, and D.-S. Park, "Depth map coding based on synthesized view distortion function," *IEEE J. Sel. Topics Signal Process.*, vol. 5, no. 7, pp. 1344–1352, Nov. 2011.

[13] L. Wang and L. Yu, "Rate-distortion optimization for depth map coding with distortion estimation of synthesized view," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Beijing, China, May 2013, pp. 17–20.

[14] Y. Mori, N. Fukushima, T. Yendo, T. Fujii, and M. Tanimoto, "View generation with 3D warping using depth information for FTV," *Signal Process., Image Commun.*, vol. 24, nos. 1–2, pp. 65–72, 2009.

[15] Y. Zhao, C. Zhu, Z. Chen, D. Tian, and L. Yu, "Boundary artifact reduction in view synthesis of 3D video: From perspective of texture-depth alignment," *IEEE Trans. Broadcast.*, vol. 57, no. 2, pp. 510–522, Jun. 2011.

[16] P. Ndjiki-Nya *et al.*, "Depth image-based rendering with advanced texture synthesis for 3-D video," *IEEE Trans. Multimedia*, vol. 13, no. 3, pp. 453–465, Jun. 2011.

[17] Y.-R. Horng, Y.-C. Tseng, and T.-S. Chang, "VLSI architecture for real-time HD1080p view synthesis engine," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 9, pp. 1329–1340, Sep. 2011.

[18] P.-K. Tsung, P.-C. Lin, L.-F. Ding, S.-Y. Chien, and L.-G. Chen, "Single iteration view interpolation for multiview video applications," in *Proc. 3DTV Conf., True Vis.-Capture, Transmiss. Display 3D Video*, Potsdam, Germany, May 2009, pp. 1–4.

[19] R.-P. M. Berretty, F. J. Peters, and G. T. G. Volleberg, "Real-time rendering for multiview autostereoscopic displays," *Proc. SPIE*, vol. 6055, p. 60550N, Jan. 2006.

[20] D. Tian, P.-L. Lai, P. Lopez, and C. Gomila, "View synthesis techniques for 3D video," *Proc. SPIE*, vol. 7443, p. 74430T, Sep. 2009.

[21] G. Tech, K. Müller, and T. Wiegand, "Evaluation of view synthesis algorithms for mobile 3DTV," in *Proc. 3DTV Conf., True Vis.-Capture, Transmiss. Display 3D Video*, Antalya, Turkey, May 2011, pp. 1–4.

[22] JCT-3V. (Mar. 2016). *3D-HEVC Reference Software, HTM-16.1*. [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_3DVCSoftware/tags/HTM-16.1

[23] Y. Chen, G. Tech, K. Wegner, and S. Yea, *Test Model 11 of 3D-HEVC and MV-HEVC*, document JCT3V-K1003, Joint Collaborative Team on 3D Video Coding Extension Development, Feb. 2015.

[24] H. Schwarz *et al.*, *Description of 3D Video Coding Technology Proposal by Fraunhofer HHI*, document MPEG11/M22570, ISO/IEC JTC1/SC29/WG11, 2011.

[25] K. Müller and A. Vetro, *Common Test Conditions of 3DV Core Experiments*, document JCT3V-G1100, Joint Collaborative Team on 3D Video Coding Extension Development, Jan. 2014.

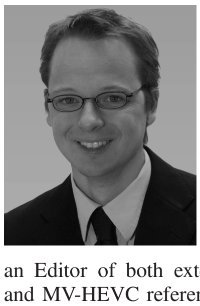
[26] *Report on Experimental Framework for 3D Video Coding*, document MPEG/N11631, ISO/IEC JTC1/SC29/WG11, 2010.

[27] R.-P. Berretty and F. Ernst, "High quality images from 2.5D video," in *Proc. Eurographics*, 2003, pp. 255–262.

[28] MPEG. (Nov. 2013). *View Synthesis Reference Software (VSRS) 3.5*. [Online]. Available: http://wg11.sc29.org/svn/repos/MPEG-4/test/trunk/3D/view_synthesis/VSRS

[29] B. T. Oh and K.-J. Oh, "View synthesis distortion estimation for AVC- and HEVC-compatible 3-D video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 6, pp. 1006–1015, Jun. 2014.

[30] S. Ma, S. Wang, and W. Gao, "Low complexity adaptive view synthesis optimization in HEVC based 3D video coding," *IEEE Trans. Multimedia*, vol. 16, no. 1, pp. 266–271, Jan. 2014.



Gerhard Tech received the Dipl.-Ing. degree in electrical engineering from RWTH Aachen University, Aachen, Germany, in 2007.

He has been with Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute, Berlin, Germany, since 2008. His current research interests include video coding and processing, including 3D representations.

Mr. Tech co-chaired several JCT-3V ad hoc groups during the development of the multiview and 3D extensions of H.265/HEVC. He has been an Editor of both extensions, and Software Coordinator of the 3D-HEVC and MV-HEVC reference software.

Karsten Müller (M'98–SM'07) received the Dipl.-Ing. and Dr.-Ing. degrees in electrical engineering from Technical University of Berlin, Berlin, Germany, in 1997 and 2006, respectively.

Since 1997, he has been with Fraunhofer Institute for Telecommunications, Heinrich Hertz Institut, Berlin, where he is currently a Project Manager of 3D video projects. His research interests include representation, coding and reconstruction of 3D scenes in free viewpoint video scenarios and coding, multiview applications and combined 2D/3D similarity analysis. He has been involved in ISO/IEC MPEG activities on multiview, multitexture, and 3-D video coding.

Heiko Schwarz received the Dipl.-Ing. degree in electrical engineering and the Dr.-Ing. degree from University of Rostock, Rostock, Germany, in 1996 and 2000, respectively.

In 1999, he joined the Image and Video Coding Group, Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute, Berlin, Germany. Since 1999, he has contributed successfully to the standardization activities of the ITU-T Video Coding Experts Group (ITU-T SG16/Q.6-VCEG) and the ISO/IEC Moving Pictures Experts Group (ISO/IEC JTC 1/SC 29/WG 11-MPEG). During the development of the scalable video coding extension of H.264/AVC.

Mr. Schwarz co-chaired several ad hoc groups of the Joint Video Team of ITU-T VCEG and ISO/IEC MPEG investigating particular aspects of the scalable video coding design. He has been appointed as a Co-Editor of ITU-T H.264 and ISO/IEC 14496-10 and as a Software Coordinator for the SVC reference software.

Thomas Wiegand (M'05–SM'08–F'11) received the Dipl.-Ing. degree in electrical engineering from Technical University of Hamburg-Harburg, Germany, in 1995 and the Dr.-Ing. degree from University of Erlangen-Nuremberg, Bavaria, Germany, in 2000.

He was a Visiting Researcher with Kobe University, Kobe, Japan; University of California at Santa Barbara, Santa Barbara, CA, USA; and Stanford University, Stanford, CA, where he also returned as a Visiting Professor. He served as a Consultant to several start-up ventures. He is currently a Consultant with Vidyo, Inc., Hackensack, NJ, USA. He has been an active participant in standardization for video coding multimedia with many successful submissions to ITU-T and ISO/IEC. He is an Associated Rapporteur of ITU-T VCEG. He is currently a Professor with the Department of Electrical Engineering and Computer Science, Technical University of Berlin, Berlin, Germany, and is jointly heading the Fraunhofer Heinrich Hertz Institute, Berlin.

Mr. Wiegand was a recipient of the ITU150 Award. The projects that he co-chaired for the development of the H.264/MPEGAVC standard have been recognized by the ATAS Primetime Emmy Engineering Award and a pair of NATAS Technology & Engineering Emmy Awards. For his research in video coding and transmission, he received numerous awards, including the Vodafone Innovations Award, the EURASIP Group Technical Achievement Award, the Eduard Rhein Technology Award, the Karl Heinz Beckurts Award, the IEEE Masaru Ibuka Technical Field Award, and the IMTC Leadership Award. He received multiple Best Paper Awards for his publications. Since 2014, Thomson Reuters named him in their list of The World's Most Influential Scientific Minds as one of the most cited researchers in his field.