# A polygon soup representation for multiview coding

Thomas Colleu, Stéphane Pateux, Luce Morin, Claude Labit

## To cite this version:

Thomas Colleu, Stéphane Pateux, Luce Morin, Claude Labit. A polygon soup representation for multiview coding. Journal of Visual Communication and Image Representation, 2010, 21 (5-6), pp.561-576. 10.1016/j.jvcir.2010.01.003 . hal-00457634

**HAL Id: hal-00457634**
**https://hal.science/hal-00457634**
Submitted on 9 Mar 2010

# A polygon soup representation for multiview coding

T. Colleu [a] [b], S. Pateux [a], L. Morin [c], C. Labit [b] *

[a]Orange Labs, FT/IMG/RD/TECH/OPERA/CVA
4, rue du Clos Courtel, BP 91226 35512 Cesson Sevigne Cedex, France

[b]INRIA-Rennes Bretagne Atlantique
IRISA, Campus de Beaulieu, 35042 Rennes, France

[c]IETR/INSA Rennes
20, Avenue des Buttes de Coësmes, 35043 Rennes, France

email : `thomas.colleu@irisa.fr, stephane.pateux@orange-ftgroup.com, luce.morin@insa-rennes.fr, claude.labit@irisa.fr`

**Abstract**

This paper presents a polygon soup representation for multiview data. Starting from a sequence of multi-view video plus depth (MVD) data, the proposed quad-based representation takes into account, in a unified manner, different issues such as compactness, compression, and intermediate view synthesis. The representation is extracted from MVD data in two steps. First, a set of 3D quads is extracted thanks to quadtree decomposition performed on depth maps. Second, a selective elimination of the quads is performed in order to reduce inter-view redundancies and thus provide a compact representation. Moreover, the proposed methodology for extracting the representation allows to reduce ghosting artifacts. Finally, an adapted compression technique is proposed that limits coding artifacts. The results presented on two real sequences show that the proposed representation provides a good trade-off between rendering quality and data compactness.

**Keywords**

3D video; multiview video plus depth; polygon soup; quadtree; multiview coding; view synthesis; redundancy simplification; ghosting artifact.

## 1. INTRODUCTION

3D video is expected as the logical evolution of 2D video [1]. Two types of 3D video applications are envisioned. The first one, called 3DTV for *3 dimensional television*, provides a relief sensation to the user by reproducing the human binocular vision with two views. Display devices allowing 3D visualization are today available. They may not require to wear special glasses. They display two views (autostereoscopic displays) and

*This work was done within the Inria/Orange Labs contract 46146063-076 CIFRE convention 150/2008 even $N$=8,10,12... views (multiview autostereoscopic displays) in order to maximize the user comfort. The second application, called FTV for *Free-viewpoint TeleVision*, allows for interactive selection of the viewpoint and direction in the scene within a certain operating range.

In order to achieve 3DTV and FTV, one can capture all the views required at the rendering stage [2]. This method may be used for stereoscopic video (2 views) but it can hardly be generalized to $N$ views due to acquisition and storage complexity. An alternative is to reduce the number of cameras and to synthesize the required intermediate views by using information about the geometry of the scene. Many studies are being conducted on this issue. In particular, within the normalization group ISO-MPEG, the working group 3D Video is currently studying the representation and coding of multiview data in order to achieve a compression standard suitable for 3D video. Here, the quality of synthesized intermediate views is essential.

Considering 3D video applications displayed on multiview autostereoscopic screens, interest for depth image-based representations has increased a lot. A depth map is an image that associates one depth value (i.e. the distance to the camera) to each pixel. It enables to synthesize intermediate views using a perspective projection method.

## Depth image-based representations

The simplest representation consists of using only one view made up of an image plus a depth map per time instant (2D+Z) [3]. But occluded regions are not contained in the depth map, and therefore disocclusion regions are not well reconstructed during view synthesis and might create strong visual artifacts.

For a wider range of viewpoints, multiple views made of 2D+Z data must be used. It is called MVD (Multi-view Video Plus Depth) data and enables to synthesize an intermediate view based on a set of views. Using MVD data, most of the occluded regions in one view can be filled with the other views [4,5]. The redundancies between all views are usually high since the same scene is captured from several close viewpoints, therefore the data load is high. To limit this overload, DES (Depth Enhanced Stereo) [1] has been recently proposed. DES is a specific case of MVD with only two views and associated depth maps.

In order to deal with both the disocclusion areas and inter-view redundancies, a solution is to select a certain view as reference and extract, from the other views, only the information which is not contained in the reference view, i.e. the occluded areas [6–8]. This is called LDV (Layered Depth Video). The advantage is that the inter-view redundancies may be easily reduced while the disocclusion areas are available. However, some color differences can appear since only a central view plus the occlusion areas are used. Moreover, the construction and compression of such data is still an open problem. In the Layered Depth Images (LDI) [6], all the side information is projected into the reference view so that one pixel can contain multiple depth and color values, yet this leads to a loss of quality due to resampling during the projection.

In addition, many contributions in the field of global model reconstruction present efficient algorithms for merging multiple depth-maps and polygonal meshes [9,10].

**Depth maps compression**

Depth maps are gray level images, so they can be compressed with an efficient video codec such as H.264. However, depth maps describe the surface of a scene and have different properties compared to an image describing the texture. Therefore, rendering intermediate views using compressed depth maps creates visually disturbing artifacts, especially around depth discontinuities (objects boundaries) as studied in [11]. With this in mind, several approaches have been proposed such as platelet-based depth coding [12]. This algorithm employs a decomposition of the depth maps using geometric primitives such that the depth discontinuities are preserved. Wavelet based coding of depth map has also been proposed in order to preserve depth discontinuities [13]. An other interesting approach is the use of scalable coding technique with ROI to ensure lossless coding around depth discontinuities [14]. These methods provide a better rendering quality for a given compression rate.

**Depth based rendering**

Rendering intermediate views using depth maps is generally performed using a point-based method: each pixel is independently reconstructed in 3D and then re-projected into the desired intermediate view. As a result, many small holes appear in the intermediate view and must be filled with post-processing techniques [5]. An alternative is to transform the depth maps into a surface using geometric primitives such as triangles [4] or quadrilaterals [15] and to disconnect these primitives at depth discontinuities so that the background and foreground are not connected. This solution eliminates the post-processing stage but requires a graphic processor.

As stated above, multiview processing for 3DTV and FTV applications raises several issues concerning data compactness, depth map compression, and intermediate view synthesis. The contribution of this study is to propose a new data representation that takes into account all these issues as a whole, i.e. that is appropriate for both the compression, transmission and rendering stages. This representation is based on 3D polygons and takes as an input, MVD data (*Multi-View plus Depth*).

In section 2, an overview of the proposed representation is given in 2.1 and the next subsections detail the polygon extraction taking into account discontinuity preservation and geometry refinement. Inter-view redundancies in that preliminary representation have to be subsequently reduced (section 2.3). Afterwards, the encoding techniques for depth data are presented (section 3). Section 4 presents several visual results illustrating the different stages of the algorithm and representation extraction, and results on depth map coding efficiency.

## 2. PROPOSED REPRESENTATION

In this section, an overview of the proposed representation is first given. Then an extraction methodology is proposed to generate such representation.

### 2.1. Overview

The proposed representation is based on a set of 3D polygons that are defined with 2D+Z data: a polygon is delimited by a block of pixels in one view; the polygon's depth

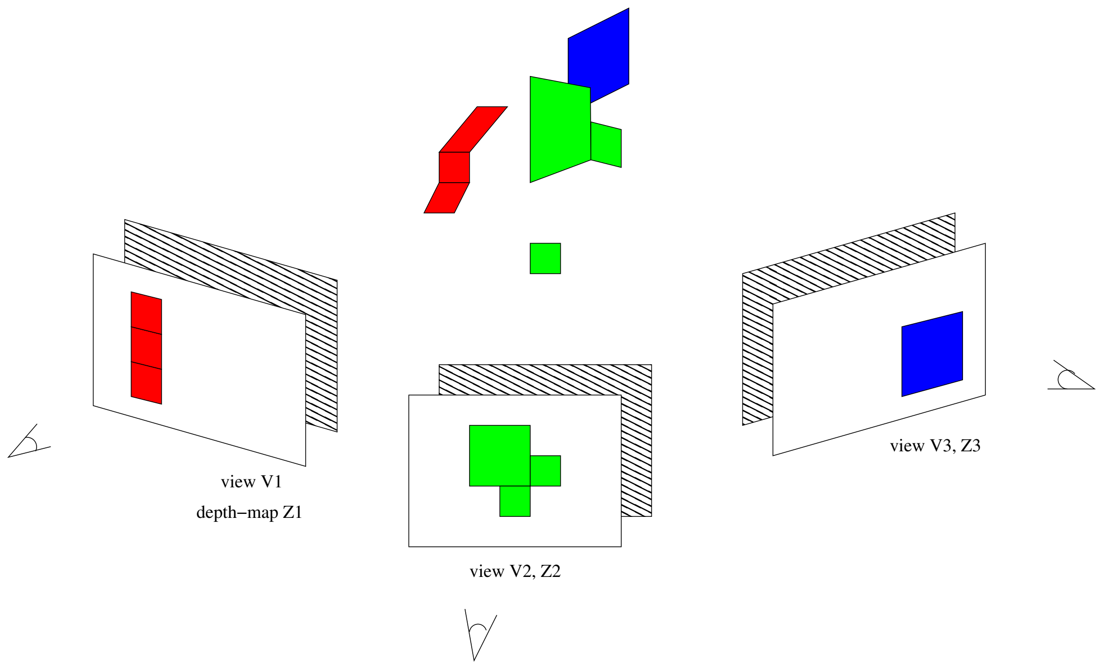


Figure 1. Proposed representation : a set of 3D polygons. Each 3D polygon is defined by a 2D polygon (here a quadrilateral) in one of the acquired views, and by the depth information for each corner of the polygon.

is defined by the depth information at the corners of the block; the polygon's texture is given by the block's texture in the image. No assumption is made on the fact that a quad is of constant depth or connected to its neighbors (see Figure 1).

Polygonal geometric primitives have several advantages.

- The size of the polygons can be adaptively determined so as to keep the number of polygons low. Thus a compact representation can be obtained (such as in [15]).
- As presented in [12], a polygonal decomposition of the depth maps that preserves the object boundaries results in a compression algorithm offering a better rendering quality compared to an H.264 algorithm, for a given compression rate.
- Polygons are frequently used as primitives at the rendering stage [4,15] since they model the continuity of the surface and thus avoid post-processing operation that fill empty pixels in the rendered image [5] [16]. Figure 2 shows an example where the data from one view has been synthesized into a different view. The rendering result can be observed with a point-based representation (depth map) (a) and a polygon-based representation (b). In (a), a post-processing algorithm is necessary to fill the small disocclusion areas (thin white lines). In (b), the continuity of the surface is preserved and only large depth discontinuities create disocclusion areas.
- Since the representation is a set of textured 3D facets, real-time rendering can easily be achieved using off-the shelf rendering tools from the computer graphics

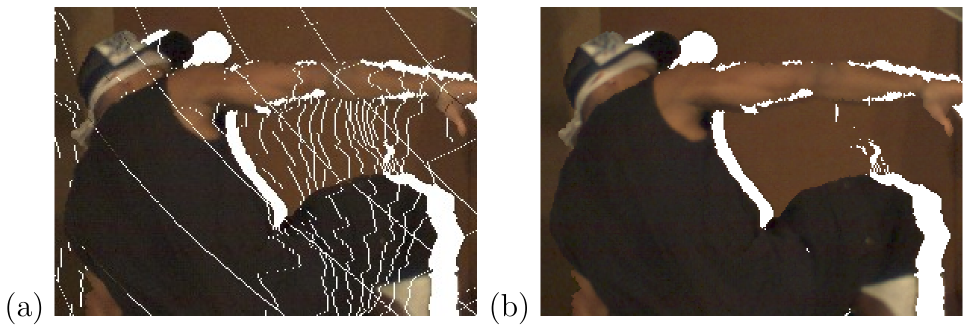


Figure 2. Comparison: point-based VS quad-based rendering

community (such as OpenGL or Direct-X APIs that benefit from GPU for real time rendering).

- This representation is very well suited for ensuring inter-operability between acquisition and display : once the representation has been extracted, it can be rendered for any view, either original views or virtual views, using the same rendering procedure in both cases.

- On the contrary to [4,15] where the polygons are used only at the rendering stage, the representation proposed here is directly based on polygons.

- The representation is based on a set of possibly *disconnected* polygons, contrary to a 3D polygonal mesh. This is why we call it a *polygon soup*. A polygon soup has several advantages over a 3D mesh: depth discontinuities are easily represented without extra information, and 3D surface continuity can be ensured by setting same coordinates to some vertexes of distinct polygons.

In the proposed representation, the chosen polygon type is the quadrilateral (quad). This choice is motivated by the following considerations:

- In the context of video coding, using such a quad-based representation enables texture information to readily be coded using a classical block-based video coding algorithm. Moreover, the coherence between block-based depth information and texture information can be exploited.

- Use of quadtree decomposition will allow to subdivide quads at depth discontinuities. Object contours, when associated with depth discontinuity, will thus be included in the quad-based representation, without the need to transmit the contour map as extra information. Furthermore adaptive quad size provides a uniform representation for smooth areas and depth discontinuities.

- Use of quadtree allows to retrieve $(x, y)$ position of quads thanks to quadtree structure, thus limiting the information sent to quadtree structure and depth information at quadtree leafs.

In a word, the polygon-based proposed representation offers the same advantages as a 3D mesh-based representation in terms of 3D description and rendering, and it also offers the same advantages as a 2D+Z representation in terms of coding simplicity.

The proposed algorithm follows the following steps:

- Generation of the polygon soup is performed using a quadtree decomposition to extract and structure the set of quads. This first step will be explained in section 2.2.
- Polygons are then selectively eliminated to reduce redundancies between views (using a similar concept to the one proposed in [8]), and also to reduce artifacts around discontinuities due to mixing colors between the background and foreground (e.g. ghosting artifacts). This second step is presented in section 2.3.
- Once the representation has been extracted, rendering can be performed. Thanks to the chosen representation, ghosting artifacts have already been removed by the polygon selection step. Only texture blending and hole filling is performed during the rendering process, which will be briefly described in section 2.4.

Figure 3 sums up the different steps of the method as well as its application framework (compression of MVD data and rendering of virtual views). Note that three views are here considered. But the proposed framework applies to any number of views.

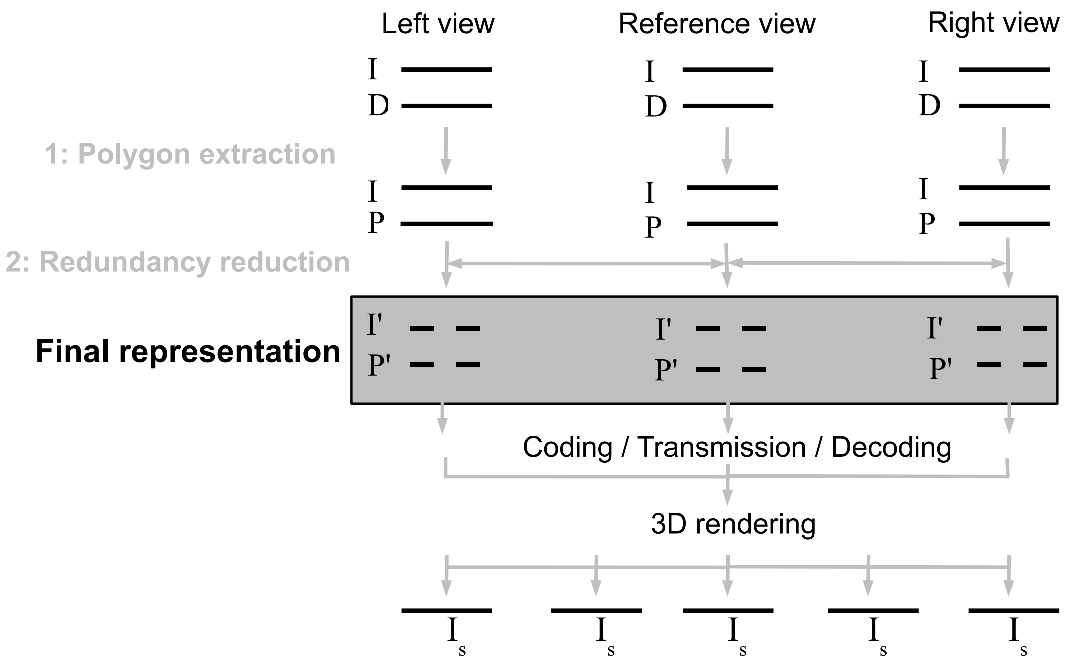


Figure 3. General overview of the proposed framework. I: image, D: depth, P: polygons, I'(resp. P'): I (resp. P) reduced, $I_S$: synthesized image

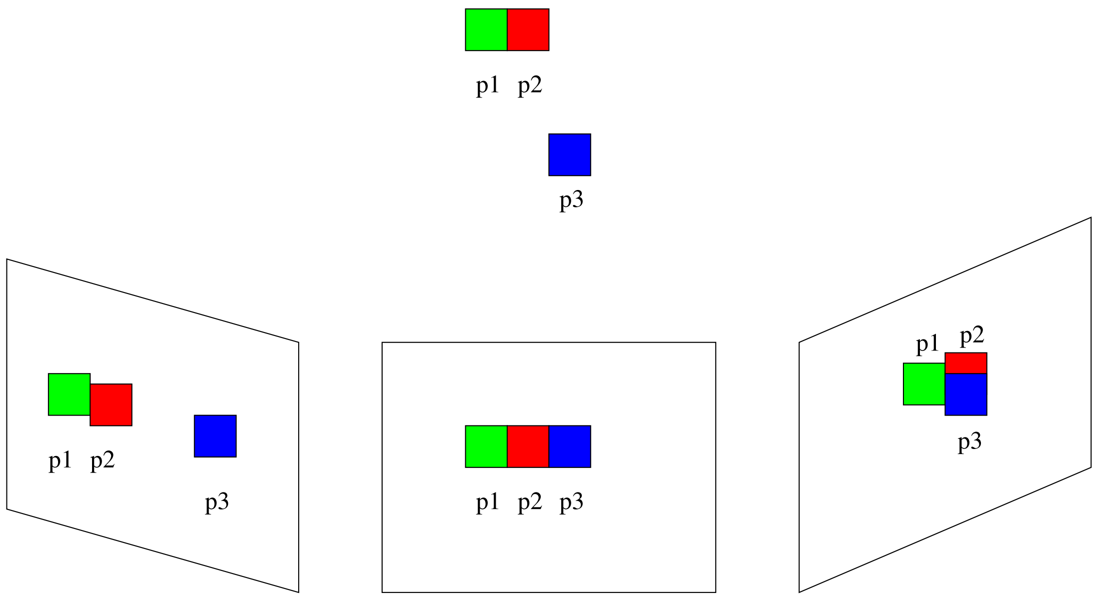


Figure 4. Warping projection distance is small between $p_1$ and $p_2$, and large between $p_2$ and $p_3$.

## 2.2. Polygon extraction

This section describes the polygons extraction from the MVD data, using a quad-tree decomposition. At this step, only depth information is used. Each depth map is processed independently so that a quad-tree is created for each view.

The quad-tree decomposition for one view aims at providing an accurate model of the associated depth map while preserving depth discontinuities. Quads are recursively scanned and divided either if the quad contains a depth discontinuity, or if it does not provide a good enough approximation of the depth map.

Since depth information will be used for warping, and not for 3D reconstruction, the two division criteria are not based on a 3D distance between original depth data and approximate depth model, as it is usually proposed in the computer vision community. Instead, we propose to use an image-based distance which aims to reduce the projection error, that is the distance between a point warped using the original depth map and a point warped using the approximated depth map.

The proposed measure that we call warping projection error, is the euclidean distance between two pixels after projection into the right-most or left-most camera (see Figure 4). This measure allows to take into account the range of projection (navigation range related to the application) and allows to tune the precision of the quadtree.

Let $p$ be a pixel in the reference view. Let $\mathcal{P}_{ref\to left}(p)$ and $\mathcal{P}_{ref\to right}(p)$ be the projections of $p$ into the left-most and right-most camera. $dist(p_1, p_2)$ is the euclidean distance between $p_1$ and $p_2$ and $\mathcal{P}_{ref\to right}()$ (resp. $\mathcal{P}_{ref\to left}()$ ) denotes the warping function from reference view onto view $right$ (resp. $left$), using depth information.

The subdivision method holds in two steps which we call discontinuity preservation and geometry refinement.

### 2.2.1. Discontinuity preservation

Blocks in the image are sub-divided if they contain large discontinuities. Let $p_1$ and $p_2$ be any two adjacent pixels in block $B$ in view $V_{ref}$. Considering threshold $T_d$, then block $B$ is sub-divided if:

$$\max_{p_1,p_2 \in B\times B, |p_1-p_2|<1} (dist(\mathcal{P}_{ref\to left}(p_1), \mathcal{P}_{ref\to left}(p_2)), dist(\mathcal{P}_{ref\to right}(p_1), \mathcal{P}_{ref\to right}(p_2)) > T_d$$

Similar procedure is performed for pixels in view $V_{left}$ (resp. $V_{right}$) projected onto $V_{ref}$ and $V_{right}$ (resp. $V_{left}$). This criterion will lead to have small quads located close to depth discontinuities.

### 2.2.2. Geometry refinement

From the coarse representation obtained, a block is again sub-divided if the geometry approximation is too high. First, a block is represented as 2 triangles corresponding to the 2 triangles that form a 3D quad. The plane equations $\pi_1, \pi_2$ of these 2 triangles are computed using the depth values of the corners of the block. Let $p_{orig}$ be a pixel at position $(x, y)$ and original depth value $Z$, and $p_{\pi_{1,2}}$ a pixel at the same position $(x, y)$ but depth value $Z_{\pi_{1,2}}$ on the plane $\pi_1$ or $\pi_2$. Then, the block is sub-divided if:

$$\max_{p_{orig}, p_{\pi_{1,2}} \in B} \left( dist(p^{left}_{orig}, p^{left}_{\pi_{1,2}}), dist(p^{right}_{orig}, p^{right}_{\pi_{1,2}}) \right) > T_g$$

As both criteria use the image-based distance, thresholds are distances $T_d$ and $T_g$ pixels. It thus quite easy to set their values. Typical values are a couple of pixel for $T_g$ and around 10 pixels for $T_d$. Moreover, the thresholds $T_d$ and $T_g$ used for this decomposition may be adjusted according to the desired bitrate.

## 2.3. Redundancy reduction

From the set of quads obtained previously, inter-view redundancies are now reduced by selecting a set of active quads in each view, which then define the final representation. The proposed process also enables to reduce ghosting artifacts around depth discontinuities.

Typically, quads situated around depth discontinuities are small and unreliable because they contain mixed color between foreground objects and background objects. Moreover, small quads often correspond to 3D surfaces whose normal vector is parallel to the image plane. Such 3D surfaces are more reliably represented using side view points (see figure 5).

To achieve this objective the following framework is proposed. The principle is to initialize the representation with a reference view (generally the central one) and to incrementally fill the disocclusion areas using the quads from the side views. In order to select preferably quads which are both large (to reduce coding cost) and reliable (to enhance rendering quality), the selection operates in two rounds. In a first step, small quads are discarded from the initial representation and selection. Then, during a second step, small quads are considered as candidates for selection to fill remaining disocclusions and to complete the foreground information if necessary. This second step is performed from outside viewpoints to reference viewpoint in order to prevent the emergence of small quads over other quads that induce ghosting artifacts.

The goal of the 2 step procedure is to exclude small quads from the final representation, as they are both costly and unreliable. Note that large quads selected during the first selection step are not the same as the ones that would be obtained by using a larger threshold during quad-tree extraction. As the polygons approximate the geometry of the surface, if larger quads are extracted, then a coarser approximation of the depth map is obtained, whereas in the selection step, only large quad adequately modeling the depth map are selected.

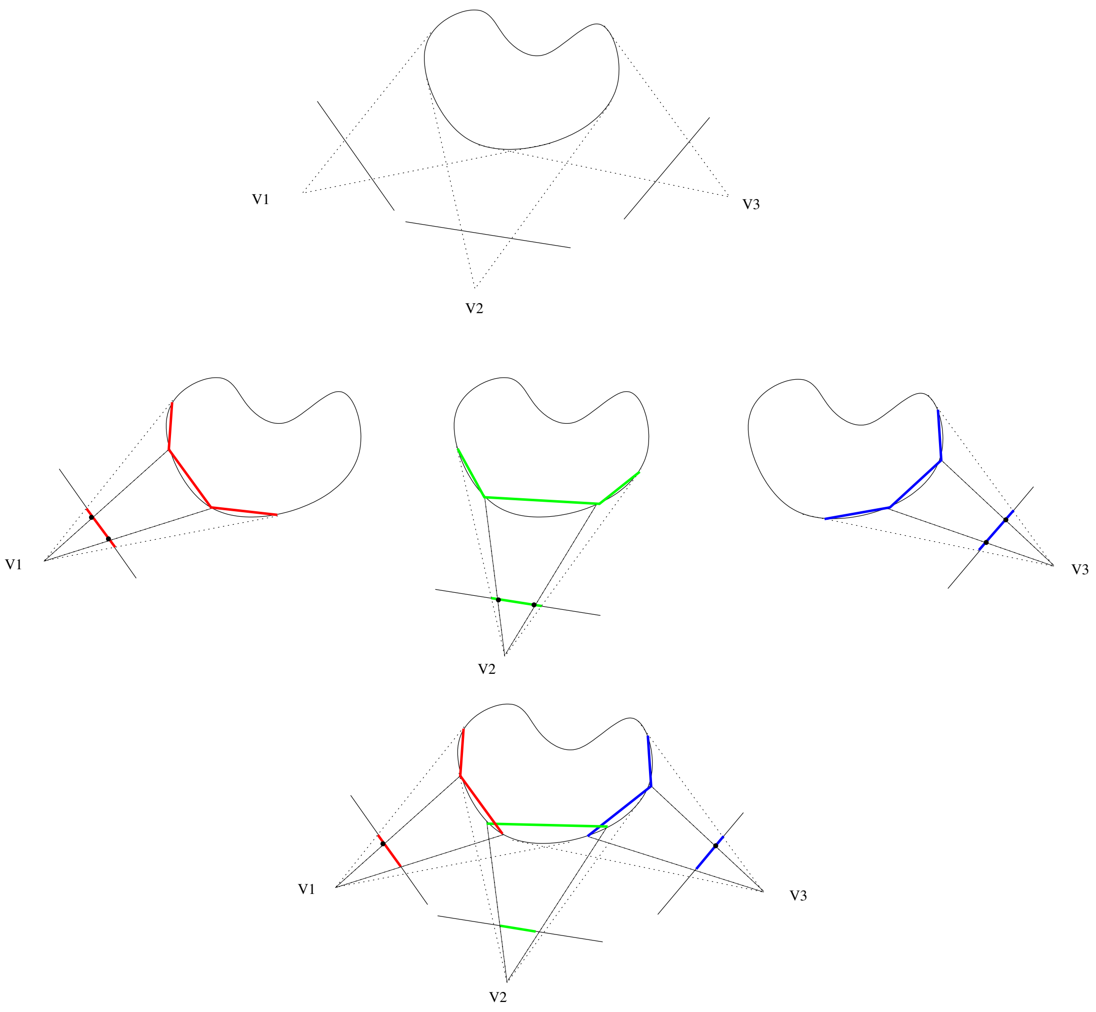


Figure 5. Unreliable Quads : considering 3 viewpoints (top), small quads are situated near depth boundaries in each view (middle), redundancy reduction selects preferably large quads, which are free of matting effects and represent the 3D surface with good resolution. Small quads are used only if necessary (bottom).

The proposed two step process also allows to avoid disturbing artifacts such as corona artifacts. This process is close to the one presented in [8] for building the LDV representation. However, the first difference is that the omitted quads are not only in the background but also in the foreground regions. The second difference is that the representation is built incrementally such that any view that has already been processed contributes to the representation before the next view is processed. Finally, the last difference is that the data is not projected and stored into the reference frame : the texture of the quad is represented as a square block in its original view, thus avoiding texture degradation due to resampling occurring with such projection.

### 2.3.1. Large quads selection

In this step, every quad of size smaller than $T_s$ is omitted, i.e. only large quads are considered as candidates. Let $\Omega$ be the final set of selected quads after large quads selection, and $Q_V$ be the set of quads from quadtree decomposition of view $V$. The idea is to initialize $\Omega$ with the large quads from a reference view $V_{ref}$ and iteratively complete and modify $\Omega$ by selecting some quads in $Q_{V_i}$, $i = 1$ *to* $N$, $i \neq ref$. Let $i$ be the current iteration, and $\Omega_{i-1}$ the set of already selected quads from previous iterations. $\Omega_{i-1}$ is first projected onto view $V_i$. The resulting image contains disocclusion areas which correspond to information from view $V_i$ which

is not present $\Omega_{i-1}$. Quads from $Q_{V_i}$ containing disoccluded pixels are thus added to $\Omega_{i-1}$.

More precisely, the initial set is given by

$$\Omega_0 = \{q \in Q_{V_{ref}}, size(q) > T_s\}$$

and the set is updated at iteration $i$ by

$$\Omega_i = \Omega_{i-1} \cup \{q \in Q_{V_i}, size(q) > T_s \quad \text{and} \quad \exists p \in q | p \notin \mathcal{P}_i(\Omega_{i-1})\}$$

where $\mathcal{P}_i(\Omega_{i-1})$ denotes projection of the current set of selected quads $\Omega_{i-1}$ onto view $V_i$.

The views are iteratively processed from the furthest to the reference view, to the the closest. This scanning order aims to reduce the overall number of quads : for instance an area which is progressively disoccluded will be represented by one large quad from the furthest view, rather than a succession of smaller quads, one from each view.

### 2.3.2. Small quads selection

The previous step gives a set of polygons (polygon soup) containing medium to big size quads, without ghosting artifacts. Even if large disocclusion areas have been filled, some background and foreground areas are still missing. Therefore, the second step consists in iteratively adding the omitted small quads only if necessary.

#### Selection of small quads in the background

Small quads are first added in the background for each view, using the same process as before. From $\Omega$ obtained previously, views are iteratively processed from the furthest to the reference view, to the the closest, and finally the reference view is processed.

This step completes $\Omega$ by filling the background of the scene : no blank area remains when $\Omega$ is projected onto any of the original views. Background areas have been added using large quads if available, and small quads only if necessary, thus drastically reducing ghosting artifacts.

**Selection of small quads in the foreground**

The last process consists in completing the foreground by adding small quads. Here again, the views are iteratively processed from the furthest to the reference view, to the closest, and finally the reference view is processed. Let $i$ be the current iteration. $\Omega$ is projected onto view $V_i$, then for each quad in $V_i$ a depth test is performed to check if the quad is in front of the scene. If so, then the quad is added to $\Omega$. This last process enables to add small quads on foreground object edges.

Note that doing the selection of small quads in two sub-steps allows to limit ghosting artifacts. In the first sub-step, pixels related to a foreground object, but having depth information related to the background can be rejected. Indeed the area where they occur may be already predicted by larger quads of side viewpoints. In second sub-step, a pixel related to a background object, but having depth information related to foreground, can be rejected. This is typically done by performing a photo consistency check: if already proposed reconstruction is of same color information that proposed new small quads, then this means that proposed new small quads has wrong depth information and should be in background.

At the end of the process, the final representation $\Omega$ is a set of polygons (polygon soup), defined as the subset of selected quads in each view.

## 2.4. Adaptive blending for view rendering

The reconstruction of a synthesized view needs to do some merging of the contributions from different viewpoints. Since a soup of polygon is used a direct technique would be to let the 3D rendering engine deal with multiple view contribution thanks to z-buffering technique. However several problems occur:

- colors of different input views may not be homogeneous,
- depth information across views may not be coherent,
- the local color of an object may vary depending on observation direction due to specular properties of materials.

The first point can be avoided by performing color equalization among view. To this extent we first perform a color equalization by fitting a parametric transfer function to reflect global illumination changes across views. That is we search for each color component $c$:

$$\arg\min_{\theta} \sum_{p \in V_{ref}} ||f(\theta, \mathcal{P}_{i \to ref}(V_i[c][p]) - V_{ref}[c][p]|| \quad (1)$$

where $f(\theta, .)$ is a fitting function (typically a continuous piecewise linear function) parameterized by $\theta$, and $V_i[c][p]$ denotes the value of color component $c$ at position $p$ in view $V_i$.

Considering second point z-buffer rendering leads to some kind of "'depth fighting"' across polygons that induce some discontinuities in the rendering. Third point also leads to artificial boundaries in uniform areas. To limit such artifacts a typical technique is to perform blending of different views prediction. In [17] a blending is proposed by generating for each view a synthesized prediction and then average these prediction to obtain the final synthesized view. Unstructured lumigraph [18] allows to do local adaptive weighting of view contribution by attributing weighting factors associated to confidence measures in the respective view contributions (using relative distance information, relative angle observation distance, resolution based criterion, ...). We proposed here a simple technique based on adaptive averaging of view contributions.

- First we compute for each input view the proposed synthesized view and associated depth map projected on intermediate view: $R_i$ and $D_i$,
- we then combine these predictions by local averaging:

$$R(p) = \frac{\sum_i w_i(p) R_i(p)}{\sum_i w_i(p)} \tag{2}$$

where $w_i(p)$ are local weights used to perform local adaptive blending of view contribution. They are defined as follows:

- first for each pixel $p$, we search for minimum depth value: $D_{min} = \min_i D_i$,
- then $w_i(p)$ is derived as follows

$$w_i(p) = \left\{ \begin{array}{ll} f(dist(curr, i) & if(D_i(p) < D_{min}(p) + \delta) \\ 0 & otherwise \end{array} \right. \tag{3}$$

where $\delta$ is used to robustly reject contribution that are occluded by other contributions, $f(dist(curr, i))$ is a decreasing function of the distance between current viewpoint and viewpoint associated to view i. In our experiment we used $f(dist(curr, i)) = \frac{1}{dist(curr,i)}$.

## 3. DEPTH DATA CODING

In this section we consider the coding of the extracted polygon soup representation for one temporal frame.

As presented in section 2.2 polygon soup is established from a quadtree-based splitting technique. Thus an encoding of the polygon soup can benefits from this quadtree structure. First recursively coding splitting information (i.e. using a flag *leaf_flag*) of the nodes of the quadtree allows defining the set of quads considered with their respective size. For a quadtree node *0* is coded to indicate that a node is split, *1* to indicate that a node is a leaf.

When a leaf node is encountered no further signaling for its children nodes is necessary. When considering deactivation of quads to reduce redundancy across views (see section 2.3) an additional flag *active_flag* is coded to signal whether current node contains some active quads or not. Note that since many quads can be deactivated we could have a non-leaf node that signals a non-active set of quads and thus effectively prune the information related to its child nodes.

Once the structure of the quadtree is defined depth information is coded for active leaf nodes of the quadtree. A quad is defined by the depth value of its 4 corners (i.e. no assumption is made on the fact that the quad is of constant depth or connected to its neighbor). Since depth values of quads are highly correlated to depth values of surrounding quads, a predictive coding of depth value is performed. That is for a quad located at $(x, y)$ position, following predictions are established:

- $\hat{z}_{tl} = \overline{zMap}[x, y]$ : depth value of top-left corner is predicted from interpolated depth values established by already coded quads,
- $\hat{z}_{tr} = \hat{z}_{bl} = \bar{z}_{tl}$: depth value of top-right and bottom-left corners is predicted from previously coded corners of current quad,
- $\hat{z}_{br} = \bar{z}_{tr} + \bar{z}_{bl} - \bar{z}_{tl}$: depth value of bottom-right corner is predicted using affine prediction from the three other previously coded corners of current quad.

where $\overline{zMap}[x, y]$ corresponds to the depth map values interpolated from the quads already coded, indexes behind $z$ values reflect corner position ($t/b$: top/bottom, $r/l$: right/left position), $\hat{z}$ corresponds to a prediction value, $\bar{z}$ corresponds to a decoded depth value (i.e. taking into account residue added to prediction).

A prediction residue is then defined to be added to these predictions. Due to high correlation of depth values these residues are often 0. A flag is then introduced to signal 0 value for these residues. If not zero, then ExGolomb code is used to code the residue.

All informations are coded using Context Adaptive Binary Arithmetic Coding (as proposed in [19]). Contexts are established depending on the level of a node in the quadtree to take into account statistical variations among quads size.

When considering deactivation of quads depth value of neighbor nodes may be not defined if neighbor are deactivated. In order to still get a valid prediction in those cases, when a node is deactivated virtual depth values are defined using previously defined predictions and setting residues to 0. This indeed corresponds to do padding of depth information.

## 4. RESULTS

Tests were performed on MVD sequences *Breakdancer* and *Ballet*. They were captured with 8 cameras (resolution 1024 × 768) placed on a horizontal arc spanning about 30°. The depth maps were estimated with a stereo algorithm [4]. [2]

The following experiments were tested with a configuration of 3 views $(V_1, V_3, V_5)$ where the central one is considered as the reference $V_3 = V_{ref}$.

### 4.1. Polygons extraction

Polygon extraction was performed with empirical thresholds $T_d = 10$ *pixels* for the discontinuity criterion, and $T_g = 5$ *pixels* for the geometry criterion. Figure 6 shows the quadtree decomposition for view $V_3 = V_{ref}$. Each depth map has a resolution of $1024 \times 768$ pixels. The average number of quads per view is 26520 for *Breakdancer* and 29876 for

[2] *Thanks to the Interactive Visual Media Group of Microsoft Research for providing the data sets*

*Ballet*. Using these quadrilaterals and corresponding textures, original and intermediate views have been synthesized.

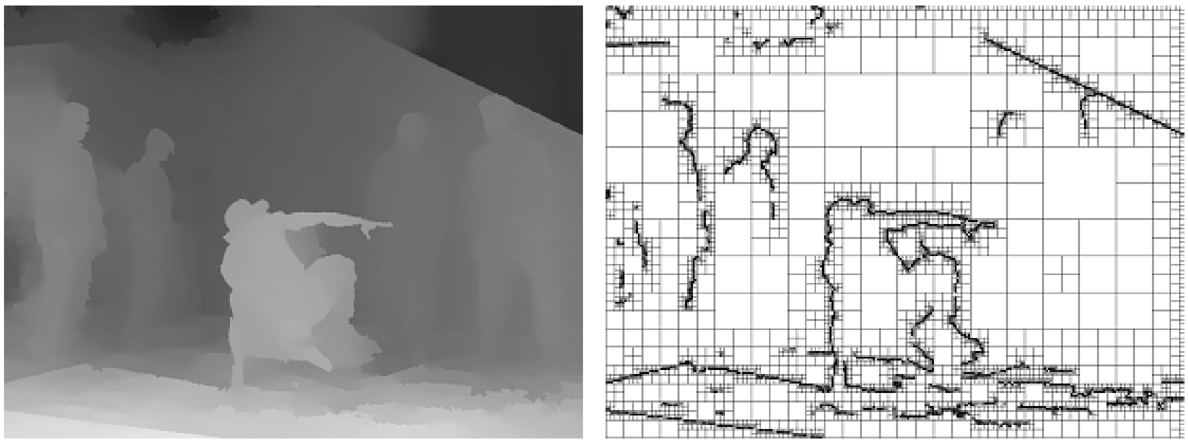


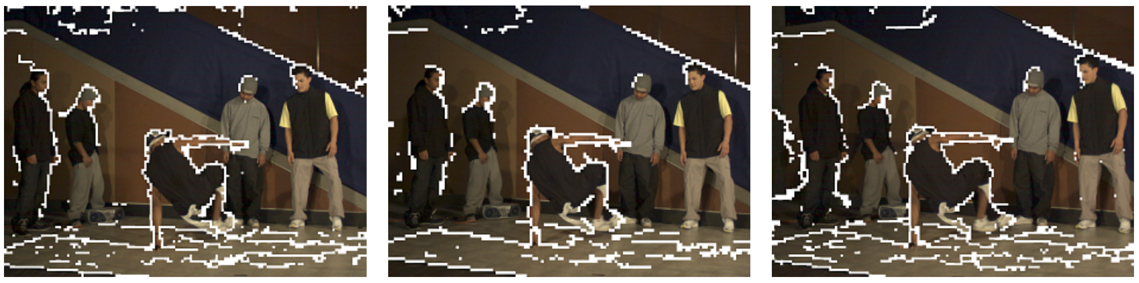
Figure 6. Quadtree decomposition of depth map for view $V_3$ of *Breakdancers*.

### 4.2. Redundancy reduction

Redundancy reduction was performed with empirical threshold $T_s = 4$ (large blocks are at least $4 \times 4$ pixel block).

#### Large quads selection

Figure 7 shows the original views where small quads (of size $< T_s$) have been removed. As expected, small quads appear mostly on object boundaries.



(a) $V_5$ (b) $V_3$ (c) $V_1$

Figure 7. Views 1,3,5 without small quads

Figure 8 shows the large quads selection process for the first iteration. $\Omega$ has been initialized with large quads from view $V_{ref} = V_3$, and thus it contains all quads in image

7(b). $\Omega$ is projected on view $V_1$ (Figure 8(a)). The disocclusion areas can be seen in white. Selected large quads from view $V_1$ are shown on figure (Figure 8(b)). Figure 8(c) shows the resulting representation $\Omega$, projected onto view $V_1$, where many white areas have been filled.

The large white regions in Figure 8(b) show that many redundancies have been eliminated.

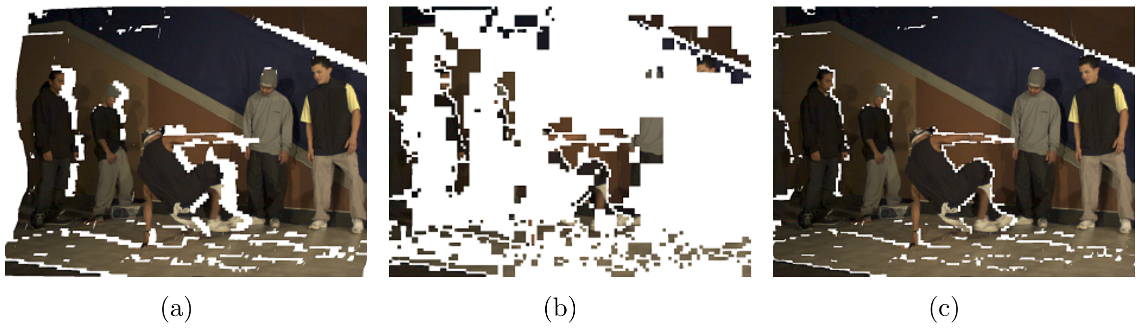


Figure 8. Quad selection for $V_1$ : $\Omega$ projected into $V_1$ (a); selected quads from $V_1$ (b); resulting representation $\Omega$ projected into $V_1$ (c).

### Adding the small quads and ghosting removal

Figure 9 shows the successive steps of quad selection on a zoomed area from $\Omega$ projected on view $V_3$. As expected, large quads leave some blanks areas 9(a), which are filled by small quads added in the background 9(b). After the last step, the useful small quads added to the foreground areas (dancer cap) 9(c).

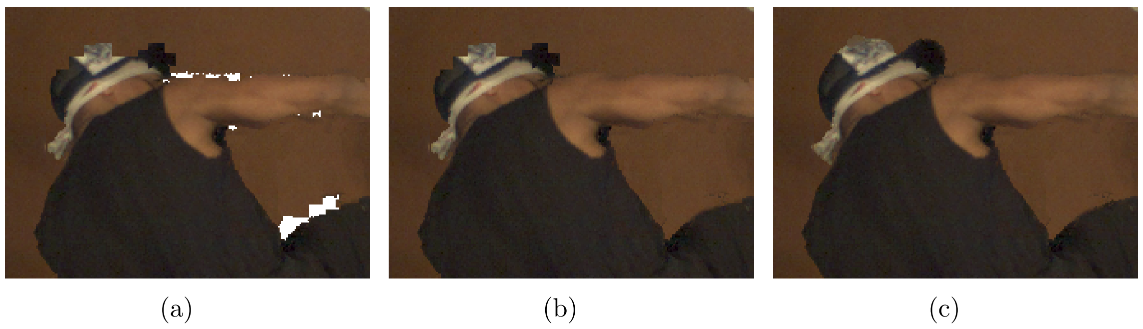


Figure 9. Successive steps of quad selection: $\Omega$ projected into $V_3$ : after large quad selection (a), after adding small quads in the background (b), after adding small quads in the foreground (c).

Figures 10 and 11 show the final obtained representation, i.e. the final set of selected quads in each view. As expected, most of the quads come from the reference view, and quads from the side views provide information on disoccluded areas.

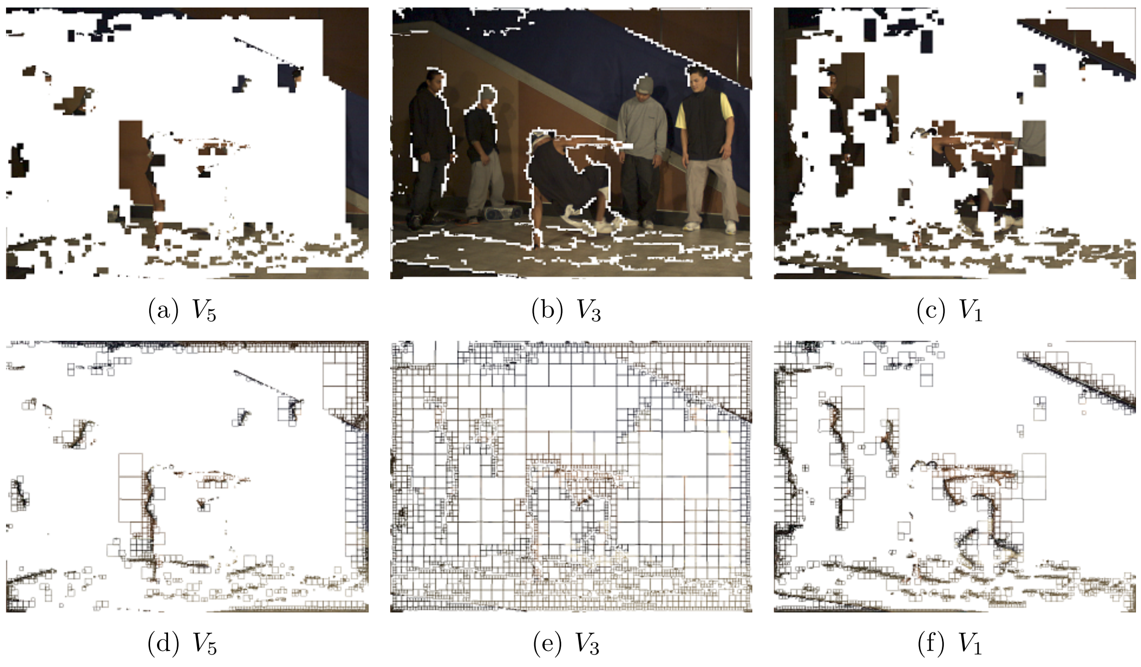


Figure 10. Result of quads selection for *Breakdancers*: final set of selected quads in each view with texture (a) (b) (c) and without texture (d) (e) (f).

Finally, from this reduced set of quads, views $V_5$, $V_3$ and $V_1$ can be synthesized (Figure 12).

The effect of the proposed three step selection process is shown in Figure 13. Picture (a) shows the projection of the final set $\Omega$ with a one step process, i.e. without first removing the small quads. The ghosting artifacts clearly appear on the left side of each character. Corresponding quads are shown in (b). Grey quads and black quads come from different views and overlap in the reconstructed view. Pictures (c) and (d) show the result when using the three step proposed method. As expected, ghosting artifact is drastically reduced and many small quads have been suppressed.

Figure 14 shows similar results for the *Breakdancers* data, zoomed on the dancer's hand in front of the wall.

### 4.3. Rendering techniques

#### 4.3.1. Blending

Figure 15 illustrates results of intermediate viewpoint generation. As can be observed on sub-figure (d) weights considered for various views contribution vary around depth discontinuities. Thanks to adaptive blending transitions between multiple views prediction is smoother and details are better preserved.

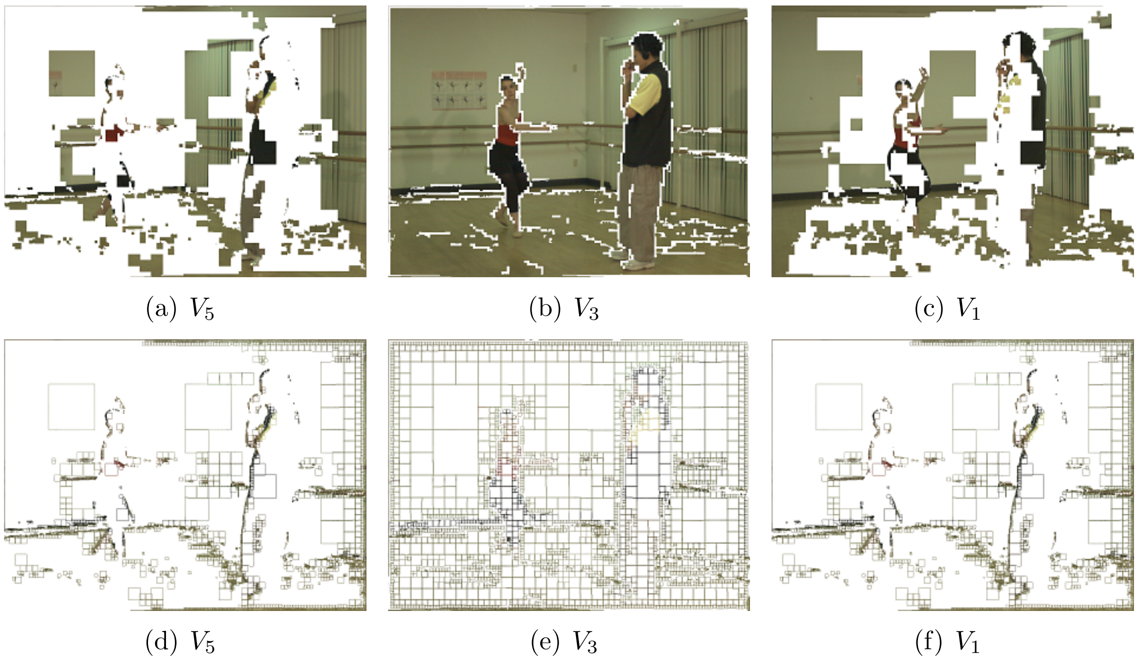


Figure 11. Result of quads selection for *Ballet*: final set of selected quads in each view with texture (a) (b) (c) and without texture (d) (e) (f).

### 4.3.2. Inpainting

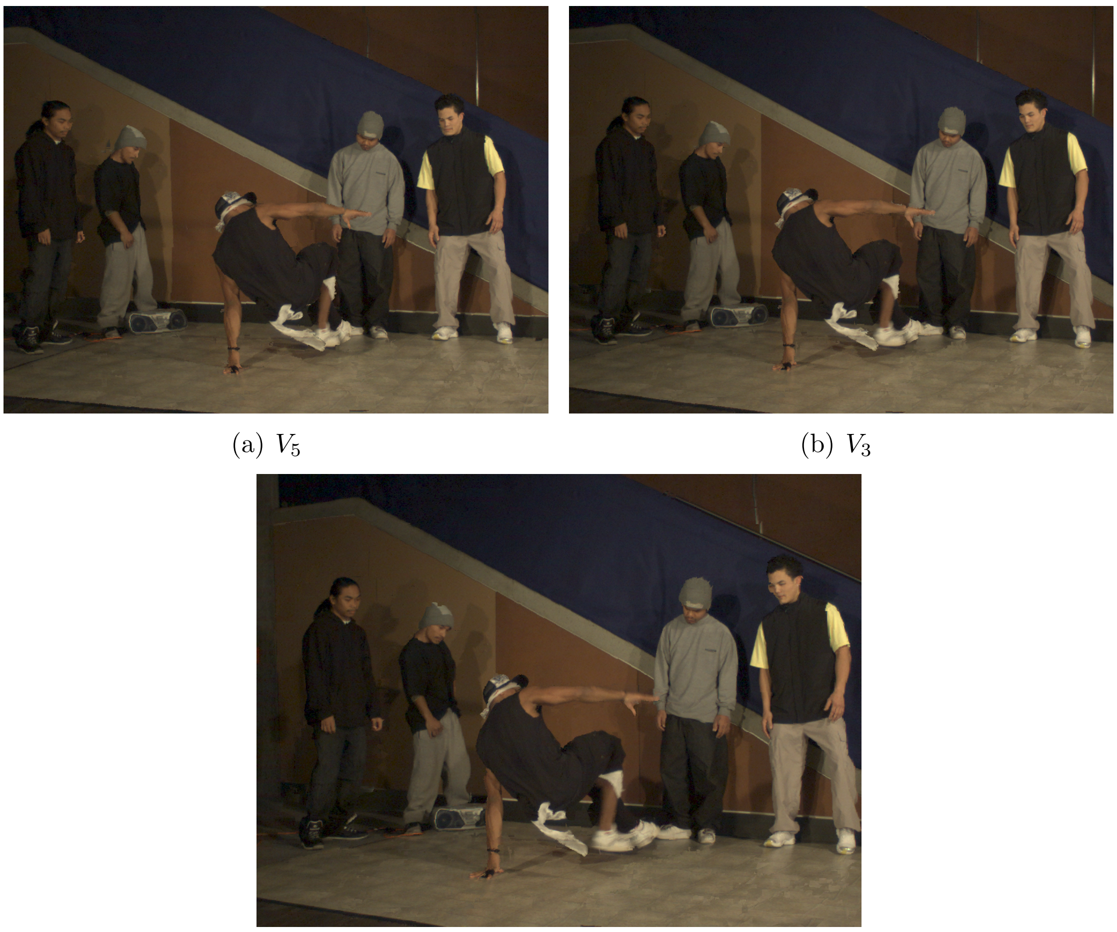
White plots observed in Figure 15 (e) and (f) correspond to non predicted pixels, i.e. pixels that do not appear in any of the views $V_1$, $V_3$ and $V_5$. These pixels can be easily reconstructed using some inpainting techniques such as [16]. The inpainting method used in this article is the Navier-Stokes one, implemented in openCV. Figure 16, illustrates the results of inpainting the white pixels situated around the right arm of the breakdancer.

### 4.3.3. Edge filtering

As explained in [5], in the original views, object boundary samples are a color mixture of foreground and background objects due to initial sampling and filtering during image capturing. However, when rendering intermediate views, the foreground-background boundaries are changed, resulting in unnaturally sharp edges. Therefore, as in [5], foreground objects are low-pass filtered along the edges to provide a natural appearance. This filtering also helps to reduce remaining artifacts along depth discontinuities. An example of a rendering result before and after edge filtering is shown in Figure 16. The cap and the T-shirt of the dancer look unnaturally sharp in front of the background. After filtering, the appearance is more natural and the irregularities along the leg are less visible.

## 4.4. Coding

Table 1 gives a comparison of the number of quads before and after the redundancy reduction (first and second rows) and the number of quads for the reference view $V_{ref}$ (third row). More than 60% of the quads have been removed compared with the full set of 3 views. Compared to a single view there is an increase of 10% to 24% of the number



(a) $V_5$ (b) $V_3$

(c) $V_1$

Figure 12. Result of quads selection: reconstructed views with final representation.

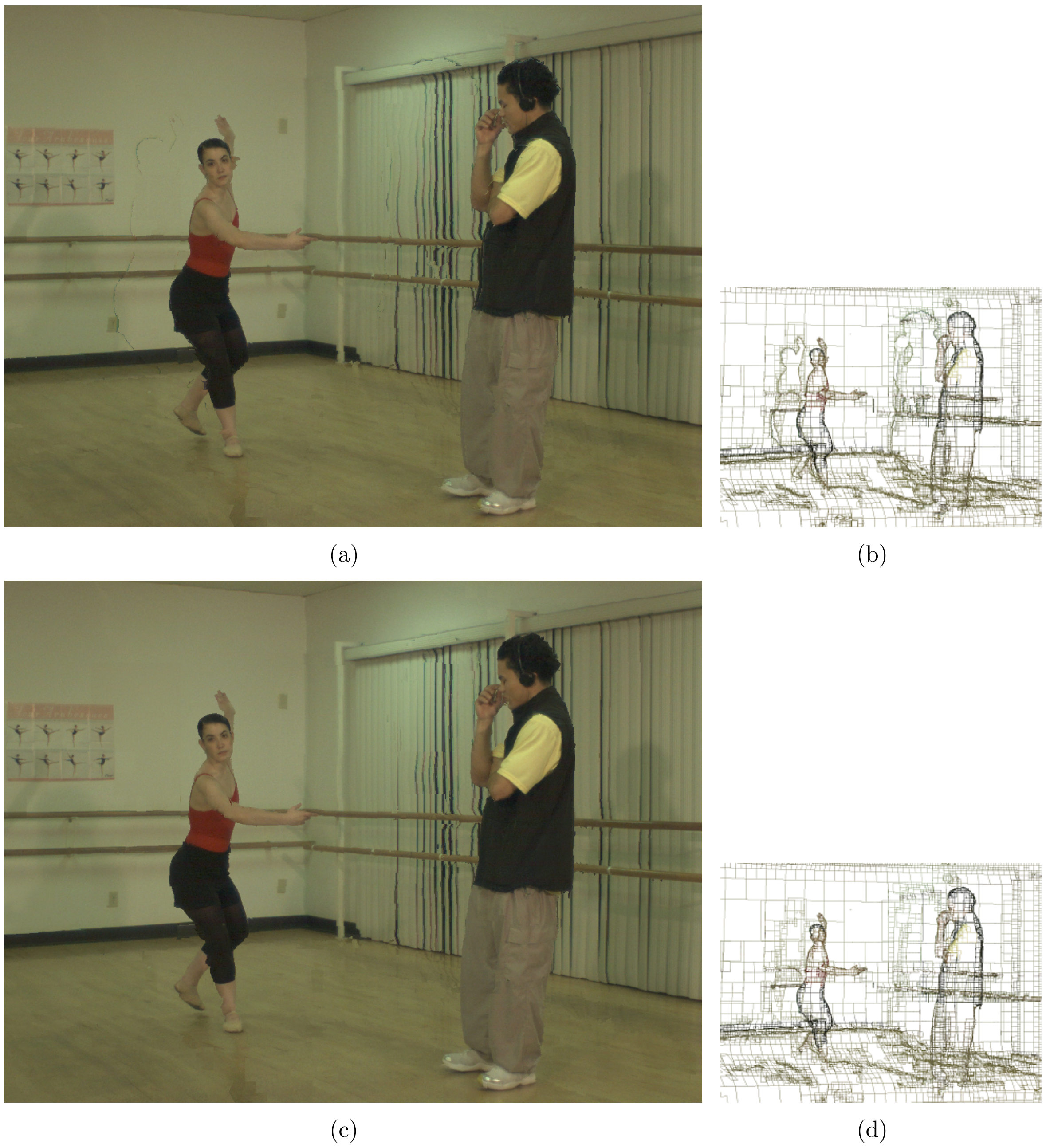


Figure 13. Effect of three step quad selection. (a)Synthesized view $V_5$ without redundancy reduction. (b) Corresponding quad representation. (c) Synthesized view $V_5$ with redundancy reduction. (d) Corresponding quad representation.

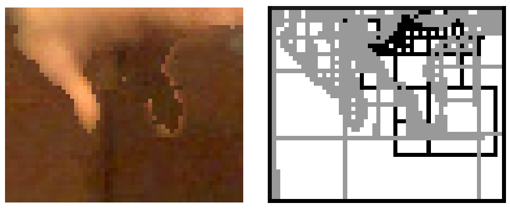
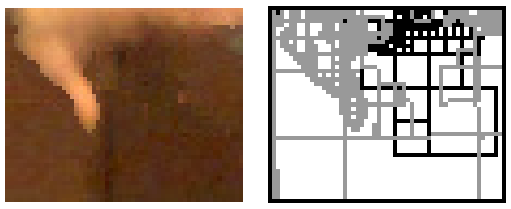
(a)  (b) 

Figure 14. Elimination of ghosting effect, using three step process eliminating small quads.

of quads.

| | *BreakDancer* | *Ballet* | |
|---|---|---|---|
| $V_1$,$V_2$,$V_3$ | 84921 | 88866 | ↓ from -60 to -64% |
| $V_1$,$V_2$,$V_3$ reduced | 30644 | 35171 | |
| 1 view | 27837 | 28291 | ↑ from +10 a +24% |

Table 1
Number of quads before and after the reduction.

Figure 17 illustrates the rate-distortion performance of coding the quadtree for the considered views; results are reported for reduced quadtree and full quadtree for sequences *BreakDancer* and *Ballet*. 56% bit rate reduction is achieved for reference view of *BreakDancer* sequence (46% for *Ballet*) while around 25% bit rate saving is obtained for side views. Higher bit-rate saving is observed on reference view thanks to lower number of small quads in this view. Resulting global rate-distortion is illustrated on figure 18. Around 30% bit rate savings is achieved with respect to coding 3 full quadtrees. Bit rate saving is lower than reduction in number of quads since signaling which quads are active is an additional cost here and that prediction coding of depth values is less efficient for reduced set of quads than for full set of quads.

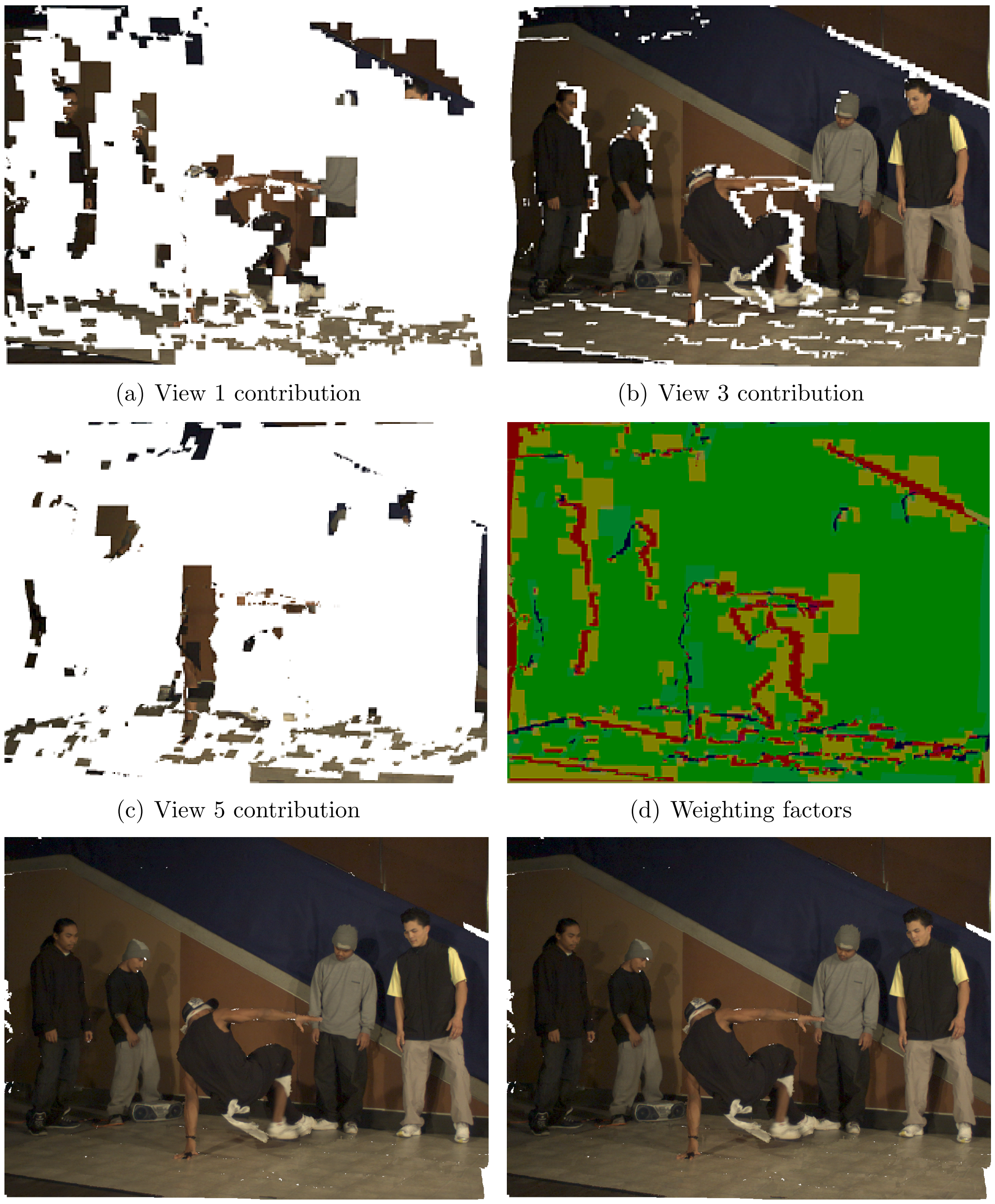
## 4.5. Comparison with an existing MVD approach

The proposed representation and coding scheme is now compared with another existing approach.

### Description of the experiments

The goal is to evaluate the rate-distortion performances of synthesized intermediate views and identify the rendering artifacts caused by the coding and rendering steps. The test data and conditions of experiment were set as follows:

- The depth information of views $V_1$, $V_3$, $V_5$ of the *BreakDancer* sequence was coded. The first 25 frames were used.



(a) View 1 contribution

(b) View 3 contribution

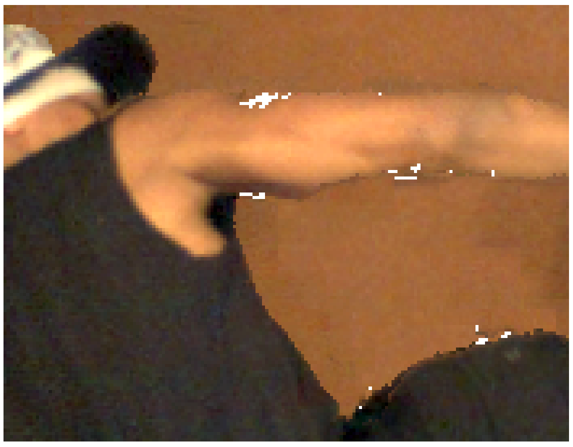
(c) View 5 contribution

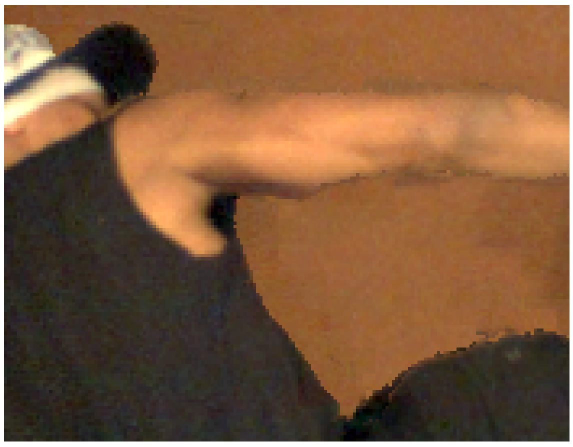
(d) Weighting factors

(e) Rendering with blending

(f) Rendering with z-buffer

Figure 15. Rendering using adaptive blending for the synthesis intermediate viewpoint $V_2$. Contributions of other views are illustrated on sub-figures (a),(b) and (c). Sub-figure (d) illustrates relative weighting used for each view (Red component is the weight associated with $V_1$, Green component is associated with $V_3$ and Blue component is associated with $v_5$). Sub-figure (e) illustrates rendering result with adaptive blending while (f) illustrates z-buffer rendering.


(a) Before inpainting


(b) After inpainting, before edge filtering

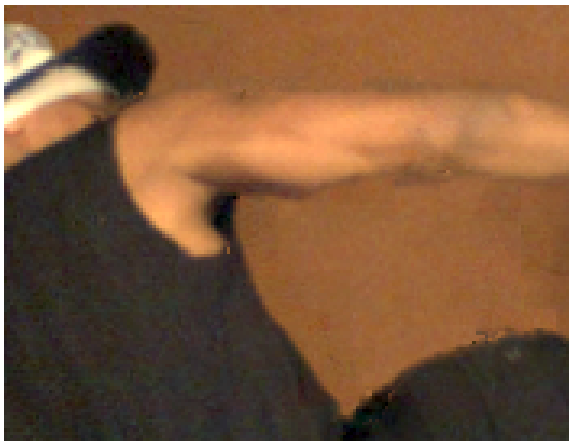
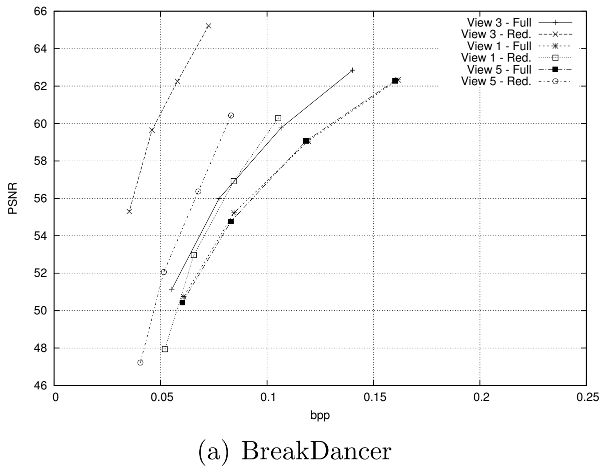

(c) After edge filtering

Figure 16. Illustration of the inpainting and edge filtering method used to fill white pixels and to provide a more natural appearance. Contrast has been enhanced for better visualization.



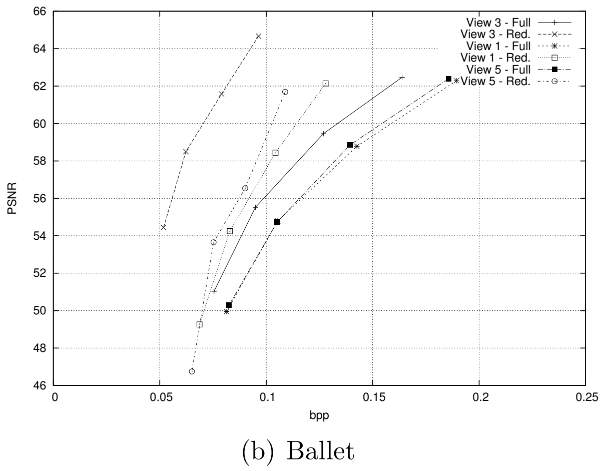
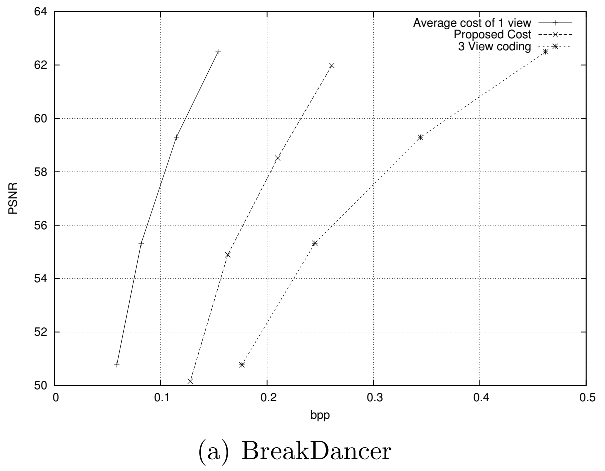


Figure 17. Rate distortion performance of reduced quadtree with respect to full quadtree for views 1, 3 and 5.



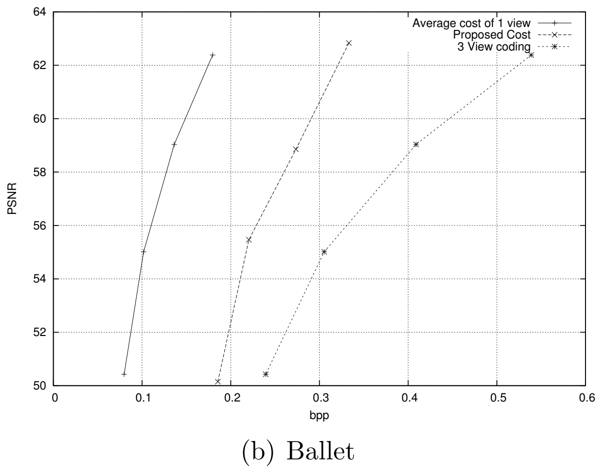


Figure 18. Rate distortion performance of reduced quadtree with respect to full quadtree. Rates are cumulated over views, PSNR is averaging over views.

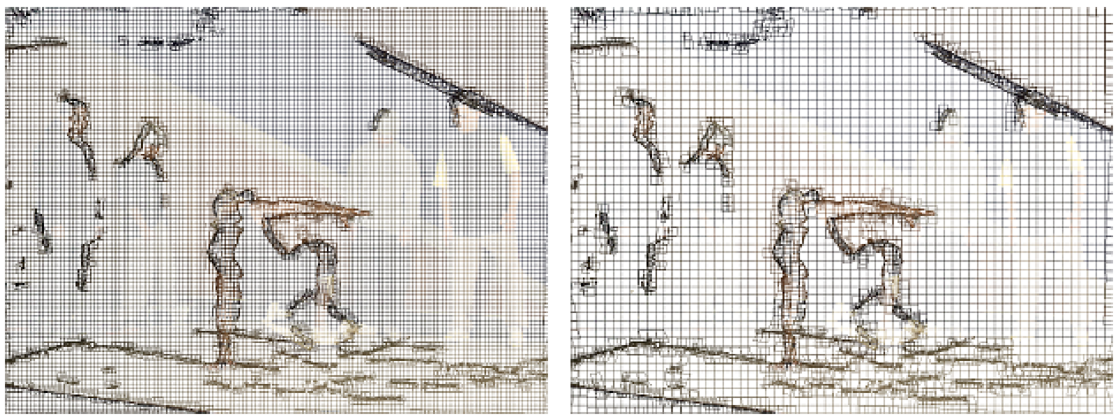
- Corresponding texture images were not coded.
- Intermediate views $V_2$ and $V_4$ were synthesized and compared with the original ones.

The existing approach that was used for comparison is based on the ITU-T H.264 Multi View Coding amendment (aka MVC) [20] for depth coding and the Nagoya university view synthesis software (VSRS) [21] currently studied in the MPEG-3DV group. This group has proposed a similar approach for encoding video plus depth data sets, which is denoted "MVD" in the group contributions. We will follow this denomination and denote the algorithm using MVC-intra+VSRS as "MVD" in the following text and figures. The coding and rendering parameters were set as follows:

- For the MVD approach, only inter-view prediction was performed with the MVC coder (i.e. only INTRA coding) because our proposed coding scheme does not take the temporal dimension into consideration at the moment. The central view V3 was coded as I picture, and the two lateral views V1, V5 were coded as P-pictures with central view as prediction (i.e. PIP inter-view coding structure). Four different bitrates and qualities have been tested by changing the quantization parameter to typical values: $QP = 22, 26, 31, 37$. In the VSRS software, the main settings were activated like half-pel precision; bi-linear filtering; boundary noise removal and view blending and inpainting.
- For the quad-based approach, the coding method described in section 3 was used. Two different levels of complexity of the quadTree were tested. It is controlled by a threshold $QSmax$ fixing the maximum size of the quads. Figure 19 shows the quad representation obtained with $QSmax = 8$ fig.(a) and $QSmax = 16$ fig.(b). In the quad representation, the number of quads of size 1 pixel is about 30% of the total number of quads whereas the surface covered by these quads in the image is very small. In order to reduce the bitrate, two different coding settings were tested: in the first one, all the quads of the representation were coded ($QSmin = 1$), and in the second one, the quads of size 1 pixel were not coded at all ($QSmin = 2$). This was done by deactivating the quads of size 1 pixel using the flag *active_flag* defined in section 3. Then at the rendering step, the lack of these small quads is compensated by the inpainting and edge filtering process (section 4.3.2 and 4.3.3).

The evaluation of the results was performed as follows:

- The quality of the coded/decoded depth information was evaluated with the PSNR objective measure. For the quad-based approach, since each view is reduced and contains only a subset of the quads defining the final representation, the PSNR was computed for the selected quads only. Hence, the empty regions of each view are not taken into account in the PSNR computation.
- The quality of the virtual views was evaluated with both PSNR and spatial PSPNR (S_PSPNR) objective measures. The S_PSPNR measure was computed using the PSPNR tool used in the MPEG-3DV group [22]. The concept of Peak Signal-to-Perceptible-Noise Ratio (PSPNR) was first proposed in 1995 [23,24] in order to obtain a measurement closer to subjective evaluation.



(a) $QSmax = 8$ (b) $QSmax = 16$

Figure 19. Final representation with two different levels of complexity of the quadTree.

- The PSNR and S_PSPNR were computed with respect to the original depth maps,and not with respect to the depth map modeled by the final non-compressed quad representation,as was done in section 4.4 for redundancy reduction evaluation.

**Results**

Figure 20 shows the depth compression results in terms of rate-distortion performance. Note that the range of obtained PSNR values is lower than in figures 17 and 18 because the reference depth is the original depth map and not the one modeled by the non-compressed quad representation. The figure shows that when the quads of size 1 pixel are not coded ($QSMin = 2$) then a bitrate reduction of 40% is obtained compared with coding these small quads ($QSMin = 1$). The figure also shows that the settings ($QSMax = 8$, $QSMin = 2$) outperforms MVC for medium and high bitrates (from 0.042 bpp). At most, a gain of 4dB is obtained at bitrate 0.08 bpp. For low bitrates, MVC outperforms the quad-based representation. This figure clearly shows that better performances are obtained when quads of size 1 pixel are not coded. Moreover, it indicates that for low bitrates the quadTree stucture must be adapted to a coarser approximation. Indeed, the setting $QSMax = 16$ shows that increasing the maximum size of the quads could provide better results at low bitrates.

Figure 21 shows the PSNR results and figure 22 shows the S_PSPNR results of intermediate view synthesis with coded depth data. Similarly to the results for the coded depth on figure 20, the quad-based approach gives higher PSNR and S_PSPNR results at medium and high bitrates. At most, a PSNR gain of 0.33dB and a S_PSPNR gain of 2dB are obtained at 0.08bpp with the settings ($QSMax = 8$, $QSMin = 2$). Comparing $QSMin = 1$ with $QSMin = 2$ the two figures show that the objective quality has not decreased although 40% of the bitrate has been reduced.

Figures 23 and 24 show detail examples of intermediate view $V_2$ at frames number 13 and 17 respectively. Corresponding videos can be watched at this adress [3]. These views

[3]http://www.irisa.fr/temics/staff/colleu/

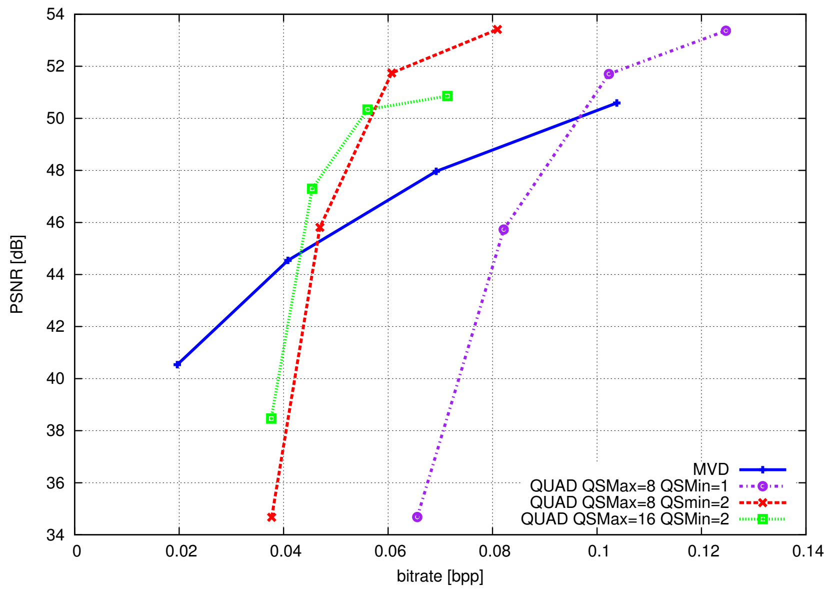


Figure 20. Rate distortion performance for multiview depth compression with MVC and coded multiview quad-based representation. Average bitrate VS average PSNR for 3 views and 25 frames of "Breakdancer" sequence.

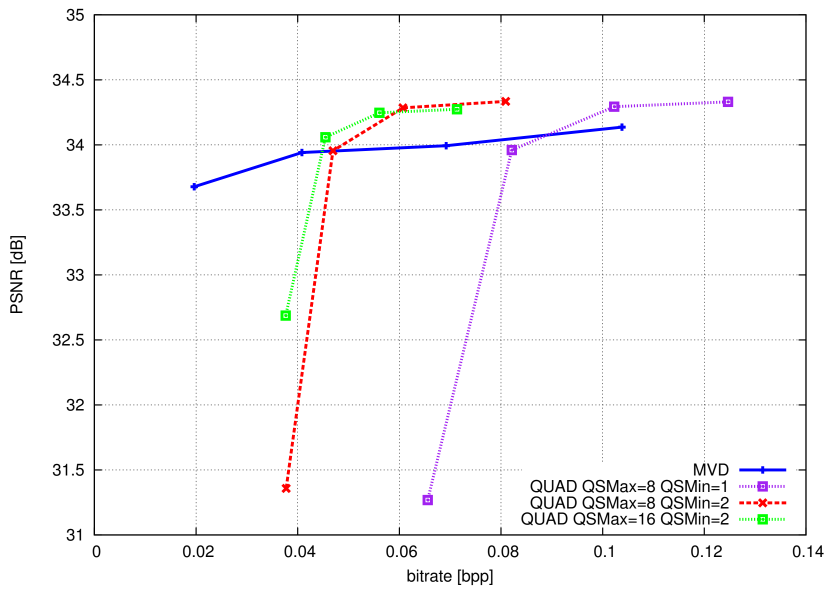


Figure 21. Rate distortion performance for view synthesis using MVD and quad-based approaches. Average bitrate VS average PSNR for 2 intermediate views ($V_2$ and $V_4$) and 25 frames of "Breakdancer" sequence.

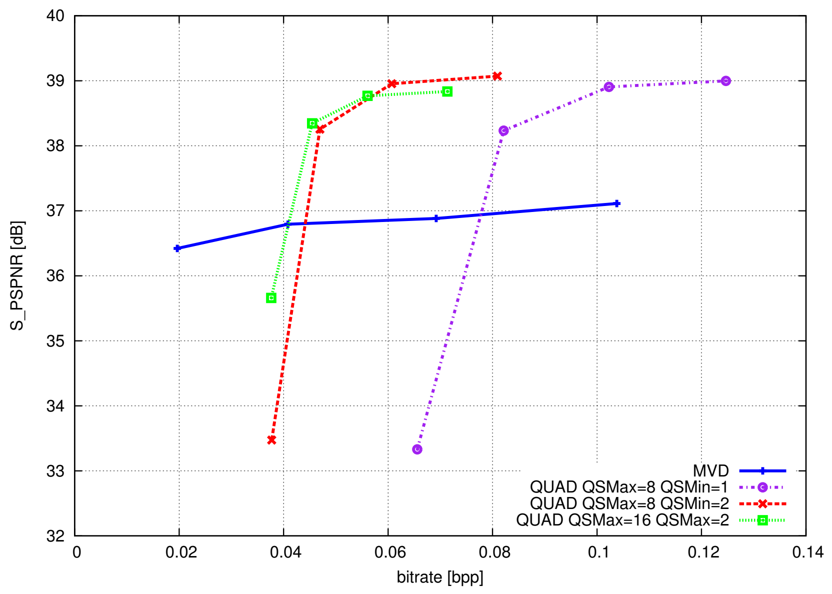


Figure 22. Rate distortion performance for view synthesis using MVD and quad-based approaches. Average bitrate VS average S_PSPNR for 2 intermediate views ($V_2$ and $V_4$) and 25 frames of ”Breakdancer” sequence.

were synthesized with MVD approach at bitrate 0.041 bpp and quad-based approach at bitrate 0.047 bpp ($QSMax = 8$, $QSMin = 2$). The original image is also shown in order to compare artifacts induced by the coding and rendering steps. In both figures, the MVD approach exhibits ghosting artifacts around the dancer. On the contrary, the quad based example does not contain such artifacts. In figure 24, the red square in the MVD approach shows a region that has been inpainted during view synthesis using VSRS software. Note that brown color of the background should appear instead of the blurred white color of the dancer’s cloth. On the other hand, in the quad-based approach, this region does not contain any artifact. In the quad-based example of figure 24, the dancer’s boundaries appear more altered than the MVD approach and original image.

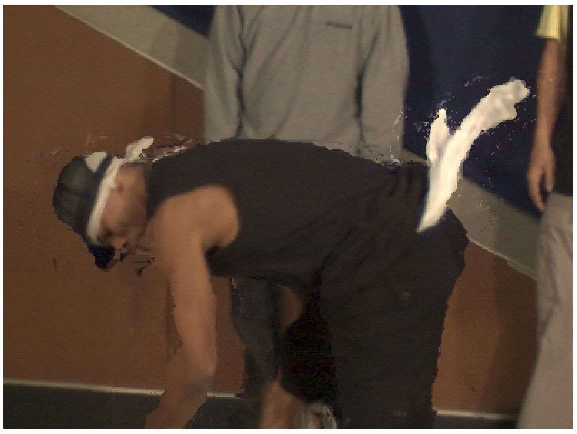
We now give the interpretations of the previous observations:

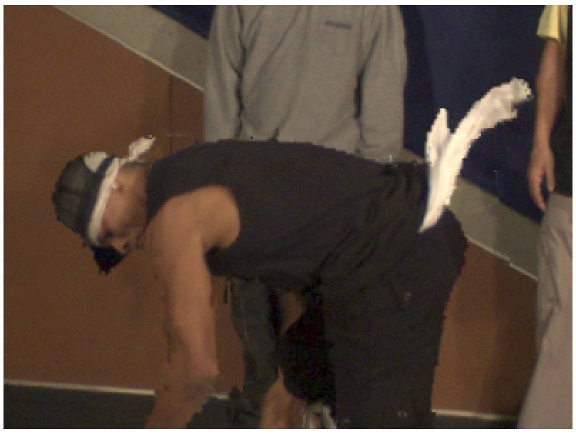
- The ghosting artifact observed with the MVD approach comes from the fact that MVC is not designed to preserve depth discontinuities, and therefore typical ringing artifacts appear in the depth maps and finally create ghosting artifacts in the intermediate view. On the contrary, the quad based example does not contain such artifacts because the quadTree structure allows to code foreground and background regions separately.

- The quad-based approach does not contain the inpainting artifact like the one observed with the MVD approach because this region has been filled by the information of view $V_5$ on the contrary to the MVD approach that used only $V_3$ and $V_1$ for the view synthesis. Using 3 views instead of 2 helps to reduce the size of unknown areas in the intermediate views.

- The dancer's boundaries in figure 24(b) is a bit deformed because of the coding process.

These results show that the proposed quad-based representation enables to reconstruct good quality virtual views. It clearly outperforms the MVD approach at medium and high bitrates in terms of objective quality measures. Although small distortions may appear at the edges, it provides equivalent overall visual quality, with no ghosting artifacts.


(a) MVD


(b) Quad-based

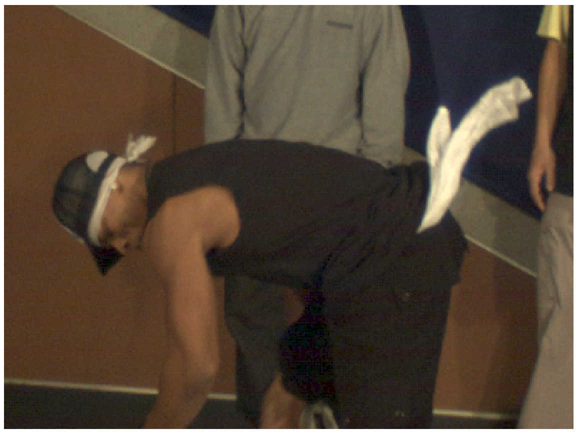
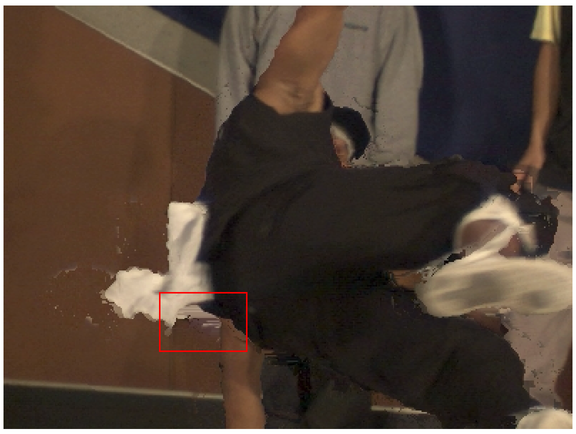
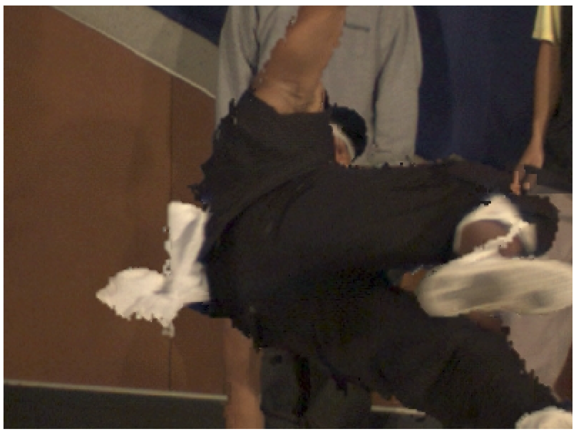

(c) Original

Figure 23. Detail example of the view synthesis result between MVD approach at 0.041 bpp and quad-based approach at 0.047 bpp.

### 4.6. Processing costs

The method has not been implemented with processing costs in mind : thus the representation construction and encoding are offline processes. However decoding and rendering may be achieved in realtime thanks to the graphic pipeline as the representation is a set of 3D textured polygons. Post processing like blending and inpainting may be discarded


(a) MVD


(b) Quad-based

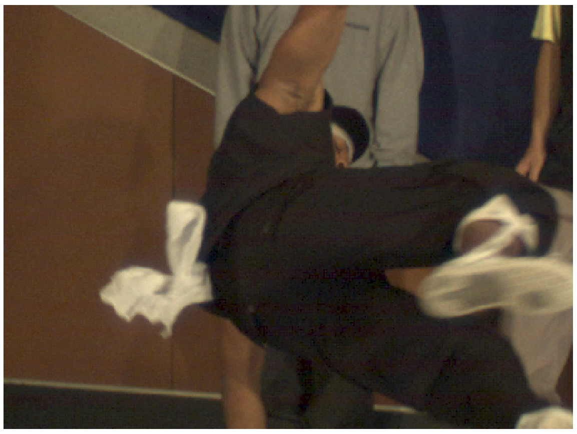

(c) Original

Figure 24. Detail example of the view synthesis result between MVD approach at 0.041 bpp and quad-based approach at 0.047 bpp.

if fast rendering is mandatory. At the moment, rendering (with blending and inpainting) is achieved at 4FPS, using openGL commands executed on the CPU.

## 5. CONCLUSION

This paper presents a polygon soup representation for multiview coding. In a unified manner, it takes into account issues identified in the literature such as data compactness, depth maps compression, and intermediate view synthesis. For each temporal frame, a set of quads is first extracted with a quadtree decomposition of the depth maps. Inter-view redundancies are then reduced based on a selective elimination of quads. The results show that the proposed representation provides a good trade-off between rendering quality and data compactness.

Moreover, the proposed methodology for extracting the representation allows to reduce ghosting artifacts. Finally, an adapted compression technique is proposed that limits coding artifacts. Results on the Breakdancer and Ballet sequences show that this representation for multiview data outperforms a MVD-based approach for medium and high bitrates (above 0.04 bpp), in terms of PSNR, S_PSPNR and visual quality.

The proposed method still suffers from several limitations. One limitation of the representation is that the number of quads depends on the geometry of the scene. Indeed, more quads are needed for a complex scene containing many geometric details than for a simple scene containing many planar surfaces. Therefore, the compression rate decreases as the geometric complexity increases. As shown by the results, our method fails to provide efficient compression for low bitrates: it is then necessary to adapt the precision of the tree-structure according to the bitrate. Specific view-dependent visual effects like reflection due to specular objects are not well modeled in the proposed representation, as only depth cues are used for quad selection. Adding photoconsistency check during quad selection would allow to model view-dependant features. Finally, as in most MVD coding schemes, the quality of the input depth maps directly influences the final rendering quality. Indeed, inconsistencies in the depth maps are not corrected nor detected in our representation, as a result some texture misalignment may appear if the depth maps are not consistent.

Future work will include a study of adaptation of the number of quads. Trade-off between compression efficiency and quality rendering will be specifically investigated (e.g. by adapting $T_d$ and $T_g$ thresholds according to the targeted bit rates). Moreover, the construction of the representation can be improved to better manage depth and texture errors or inconsistencies across views. Lastly, the temporal dimension of the video sequence will be considered to improve performance and to ensure temporal coherence of the proposed representation. More precisely, we plan to predict a polygon soup at time *t+1* thanks to the polygon soup at time *t*. As for classical video compression, prediction vectors could be applied on each quad of the representation. Since quads are 3 dimensional, then 3D prediction vectors would be considered.

## REFERENCES

1. A. Smolic, K. Müller, P. Merkle C.and P. Kauff, and T. Wiegrand, "An overview of available and emerging 3d video formats and depth enhaced stereo as efficient generic

solution," in *Picture Coding Symposium, Chicago, US*, 2009.

2. T. Kobayashi, T. Fujii, T. Kimoto, and M. Tanimoto, "Interpolation of ray-space data by adaptive filtering," in *Three-Dimensional Image Capture and Applications*, 2000, pp. 252–259.
3. C. Fehn, P. Kauff, M. Op de Beeck, F. Ernst, W. IJsselsteijn, M. Pollefeys, L. Van Gool, E. Ofek, and I. Sexton, "An evolutionary and optimised approach on 3d-tv," in *In Proceedings of International Broadcast Conference*, Amsterdam, Netherlands, 2002, pp. 357–365.
4. C.L. Zitnick, S.B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High-quality video view interpolation using a layered representation," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 600–608, 2004.
5. A. Smolic, K. Muller, K. Dix, P. Merkle, P. Kauff, and T. Wiegand, "Intermediate view interpolation based on multiview video plus depth for advanced 3d video systems," in *ICIP*, 2008, pp. 2448–2451.
6. J. Shade, S. Gortler, L. He, and R. Szeliski, "Layered depth images," in *ACM SIGGRAPH*, 1998, pp. 231–242.
7. W.H.A. Bruls, C. Varekamp, R.K. Gunnewiek, B. Barenbrug, and A. Bourge, "Enabling introduction of stereoscopic (3d) video: Formats and compression standards," in *ICIP*, 2007, pp. 89–92.
8. K. Müller, A. Smolic, K. Dix, P. Kauff, and T. Wiegand, "Reliability-based generation and view synthesis in layered depth video," in *MMSP*, 2008, pp. 34–39.
9. P. Gargallo and P. Sturm, "Bayesian 3d modeling from images using multiple depth maps," in *CVPR '05*, Washington, DC, USA, 2005, vol. 2, pp. 885–891.
10. M. Pollefeys, D. Nistér, J. M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S. J. Kim, P. Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewénius, R. Yang, G. Welch, and H. Towles, "Detailed real-time urban 3d reconstruction from video," *Int. J. Comput. Vision*, vol. 78, no. 2-3, pp. 143–167, 2008.
11. P. Merkle, A. Smolic, K. Muller, and T. Wiegand, "Multi-view video plus depth representation and coding," *ICIP*, vol. 1, pp. 201–204, 2007.
12. P. Merkle, Y. Morvan, A. Smolic, D. Farin, K. Muller, P.H.N. de With, and T. Wiegand, "The effect of depth compression on multiview rendering quality," in *3DTV Conference*, 2008.
13. M. Maitre and M.Do, "Shape-adaptive wavelet encoding of depth maps," in *Picture Coding Symposium, Chicago, US*, 2009.
14. S. Yea and A. Vetro, "Multi-layered coding of depth for virtual view synthesis," in *Picture Coding Symposium, Chicago, US*, 2009.
15. J. Evers-Senne, J. Woetzel, and R. Koch, "Modelling and rendering of complex scenes with a multi-camera rig," in *Conference on Visual Media Production (CVMP)*, 2004.
16. K. Oh, S. Yea, and Y. Ho, "Hole filling method using depth based in-painting for view synthesis in free viewpoint television and 3-d video," in *Picture Coding Symposium, Chicago, US*, 2009.
17. M. Tanimoto, "Overview of free viewpoint television," *Signal Processing: Image Communication*, vol. Volume 21, Issue 6, pp. 454–461, 2006.
18. C. Buehler, M. Bosse, L. McMillan, S.J. Gortler, and M.F. Cohen, "Unstructured

lumigraph rendering," in *SIGGRAPH 2001, Computer Graphics Proceedings*, Eugene Fiume, Ed. 2001, pp. 425–432, ACM Press / ACM SIGGRAPH.

19. "ISO/IEC ITU-T recommandation H264 : Advanced Video coding for generic audio-visual services," International Standard, Joint Video Team (JVT) of ISO-IEC MPEG & ITU-T VCEG, May 2003.
20. ISO/IEC JTC1/SC29/WG11, *Text of ISO/IEC 14496-10:200X/FDAM 1 Multiview Video Coding*, Doc. N9978, Hannover, Germany, July 2008.
21. M. Tanimoto, T. Fujii, K. Suzuki, N. Fukushima, and Y. Mori, *Reference Softwares for Depth Estimation and View Synthesis*, ISO/IEC JTC1/SC29/WG11MPEG2008/M15377, April 2008.
22. Yin Zhao and Lu Yu, *Peak Signal-to-Perceptible-Noise Ratio Tool: PSPNR 1.0*, ISO/IEC JTC1/SC29/WG11 MPEG2009/M16584, London 2009.
23. Chun-Hsien Chou and Yun-Chin Li, "A perceptually tuned subband image coder based on the measure of just-noticeable-distortion profile," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 5, no. 6, pp. 467–476, Dec 1995.
24. Chun-Hsien Chou and Chi-Wei Chen, "A perceptually optimized 3-d subband codec for video communication over wireless channels," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 6, no. 2, pp. 143–156, Apr 1996.