ANCHOR POINTS CODING FOR DEPTH MAP COMPRESSION

Ionut Schiopu, Ioan Tabus

Department of Signal Processing, Tampere University of Technology,
P.O.Box 553, FIN-33101 Tampere, FINLAND
ionut.schiopu@tut.fi, ioan.tabus@tut.fi

ABSTRACT

The paper deals with encoding the contours of given regions in an image. All contours are represented as a sequence of contour segments, each such segment being defined by an anchor (starting) point and a string of contour edges, equivalent to a string of chain-code symbols. We propose efficient ways for anchor points selection and contour segments generation by analyzing contour crossing points and imposing rules that help in minimizing the number of anchor points and in obtaining chain-code contour sequences with skewed symbol distribution. When possible, part of the anchor points are efficiently encoded relative to the currently available contour segments at the decoder. The remaining anchor points are represented as ones in a sparse binary matrix. Context tree coding is used for all entities to be encoded. The results for depth map compression are similar (in lossless case) or better (in lossy case) than the existing results.

Index Terms— Lossless and lossy compression, contour compression, anchor points, depth map

1. INTRODUCTION

Depth compression has an important role in the multi-view compression for the 3D Television (3DTV) and Free Viewpoint Television (FTV). In high quality view synthesis, the use of lossless compressed images is important for eliminating the artifacts in depth map image based rendering technologies.

In lossless compression several approaches were proposed: image block splitting and Gray coding of bit planes for binary compression schemes [1], H264/AVC standard modification for depth maps [2], palette images coder with good results for depth maps [3]. In [4], the contour encoder uses a different contour segments generator than the proposed method. The best performance is shown by [5], where the contours are encoded using 2D contexts.

In lossy compression there are numerous approaches: triangular image decomposition by binary tree [6], quad-tree decomposition with a platelet based approach for region filling [7], two image pyramid structures for arc breakpoints and sub-band samples [8], a reduced image resolution and an up-sampling algorithm [9, 10]. Texture and color correlation is studied in [11], where a joint depth/texture coding scheme is used, and in [12], where texture segmentation is used for depth map segmentation. In [13], lossy versions of a depth map are created by either merging or splitting regions, and are compressed lossless.

Our method uses the approach of finding in an image all maximal regions containing 4-connectivity pixels, and encoding the contours and the depth value inside each region. The method is focusing on encoding the contour using an efficient way for generating contour segments, represented using their anchor points and chain-code strings. Section 2 presents the rules for traversing the contour, and the anchor point selection. Section 3 presents deterministic schemes for chain-code symbols changing, and coding scheme for entities to be encoded. Section 4 presents experimental results for lossy and lossless compression. Section 5 draws the conclusions.

2. ANCHOR POINTS CODING (APC)

A depth map is a matrix I of size $n_r \times n_c$, with the integer $I_{x,y}$ representing the depth for each pixel position (x, y). The proposed method finds all connected regions Ω_k of pixels with equal depth, and encodes the contour and the depth value of each region. A region Ω_k containing connected pixels in 4-connectivity can be formally defined as: for $\forall(x, y) \in \Omega_k$ and $\forall(x_i, y_i) \in \{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\}$, if $I_{x,y} = I_{x_i,y_i}$, then $(x_i, y_i) \in \Omega_k$.

The contour map is the union of all contour edges that form the regions contours. One contour edge separates two neighboring pixels belonging to two different regions. The contour map is stored in a graph having vertices placed in a $(n_r + 1) \times (n_c + 1)$ contour grid, where the graph edges represent the contour edges. A vertex is denoted by $P = (i, j)$, where (i, j) are contour grid coordinates. If in the image grid $I_{i,j} \neq I_{i,j+1}$, then in the contour grid a contour edge is introduced between the vertices $(i, j+1)$ and $(i+1, j+1)$. If $I_{i,j} \neq I_{i+1,j}$, then a contour edge is introduced between the vertices $(i+1, j)$ and $(i+1, j+1)$. Two vertices are adjacent if there is a contour edge between them. A contour segment is 'drawn' as a vector $\Gamma_k = [P_1 \quad P_2 \quad \cdots \quad P_{n_{\Gamma_k}}]^T$ of n_{Γ_k} adjacent vertices.

The 3OT chain-code representation [14] codify each Γ_k by a vector $S_k = [s_1 \quad s_2 \quad \cdots \quad s_{n_{\Gamma_k}-2}]^T$, where s_i is a 3OT symbol that encodes: 0 for 'go forward', 1 for 'change orientation', and 2 for 'go back'. A symbol s_i encodes a current vertex P_{i+2} relatively to the previous two vertices: P_i and P_{i+1}. Hence, P_1 and P_2 are needed in order to reconstruct Γ_k. The chain-code vectors are concatenated for coding in a long vector of 3OT symbols $S = [S_1^T \quad S_2^T \quad \cdots \quad S_{n_\Gamma}^T]^T$.

We call here anchor point the first vertex in each Γ_k. The coding of the anchor points is our main focus. Our method offers solutions to generate the set $\Gamma = \{\Gamma_k\}_{k=1}^{n_\Gamma}$, which represents the entire contour map, in such a way that we have a small number, n_Γ, of contour segments (since the anchor points are very expensive to code), and a maximum number of symbols 0 and a minimum number of symbols 2 in S for an efficient coding of 3OT chain-codes.

A summary of the generating procedure for the set Γ and the anchor points selection is as follows:

(a) Search column-wise (see Section 2.2) for the next anchor point (i, j) (a vertex having unvisited adjacent vertices), and mark it in the matrix of anchor points $\Upsilon(i, j) = 1$ (see Section 2.3).

(b) Generate the contour segment Γ_k starting from the anchor point, using the rules form Section 2.1, and ending in a vertex with

Directive T1. *When a P_k^3 vertex is reached, the next vertex is selected so that next encoded chain-code symbol is $s_i \neq 0$.*

Directive T2. *When $P_k^3 = (i, j)$ is reached if $(i-1, j)$ and $(i+1, j)$ are unvisited adjacent vertices, then visit $(i+1, j)$ (if $(i, j-1)$ and $(i, j+1)$ are unvisited, then visit $(i, j+1)$).*

Directive T3. *When a P_k^4 vertex is reached, the next vertex is selected the one that generates a chain-code symbol 0.*

Directive T4. *When a P_k vertex with already three visited adjacent vertices is reached, the remaining vertex, P_j, is adjacent if a chain-code symbol 0 moves from P_k to P_j. If so, the decoder knows that $P_k = P_k^4$ and go visit P_j (without encoding the 0 chain-code), else $P_k = P_k^3$ and Γ_k ends.*

Directive C1. *When $P_k = (i, j)$ is reached, if the next adjacent vertex to visit is $P_{k+1} = (i-1, j)$ or $P_{k+1} = (i, j-1)$, then P_{k+1} is not an anchor point.*

Directive C2. *When P_{k+1} is reached and $P_{k-1} = (i, j)$, if the next adjacent vertex to visit is $P_{k+2} = (i-1, j)$ or $P_{k+2} = (i, j-1)$, then P_{k+1} is not an anchor point.*

Fig. 1. The set of APC directives to be used when reaching a vertex either in a Γ_k or as an anchor point. Directives T1-T4 are used for generating contour segments by traversing from a vertex to one of its selected unvisited adjacent vertex. Directives C1-C2 are used to check if a vertex is not a possible anchor point position.

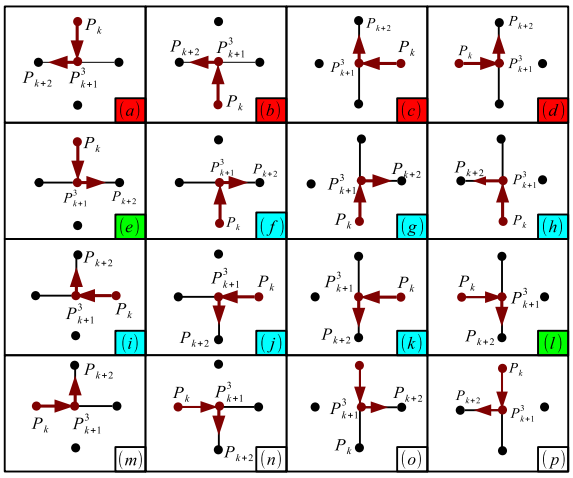


Fig. 2. All possible ways of selecting the next adjacent vertex to visit for a P_k^3 vertex. An arrow shows the next vertex to visit as part of a current Γ_k, a red (black) dot is a visited (unvisited) vertex, and a red (black) line is a visited (unvisited) contour edge.

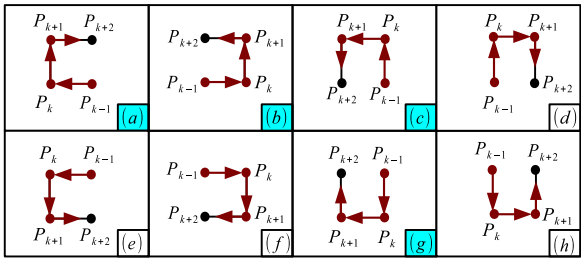


Fig. 3. Cases where a chain-code symbol 2 encodes P_{k+2}. An arrow shows the next adjacent vertex to visit as part of a current Γ_k, a red (black) dot is a visited (unvisited) vertex, and a red (black) line is a visited (unvisited) contour edge.

no unvisited adjacent vertices. While traversing Γ_k, check each vertex if it is a possible anchor point (see Section 2.2), and save the found presumptive anchor points at the end of the list Ψ of possible relative anchor points (see Section 2.3). Mark with 2 in the matrix Υ the remaining vertices in Γ_k (the anchor point is already marked).

(c) While there are still unprocessed positions ℓ in Ψ, if $\Psi(\ell)$ is an anchor point, set the flag of being anchor point, $\Phi(\ell) = 1$, else set $\Phi(\ell) = 0$. Treat any such anchor point as in **(b)**.

(d) Continue with **(a)** until no more anchor points are found.

(e) Encode S, Φ and Υ, in this order.

2.1. Rules for traversing contour segments

A contour segment Γ_k is 'drawn' by the vertices saved when traversing the contour, in the contour segments generation. Details about the anchor points search are present in Section 2.2. The Directives T (see Fig. 1) are used to choose the next adjacent vertex to visit, when there are more options. In the contour graph, the following types of vertices can be found:

(a) P_k^1 has degree one, and is a contour graph boundary: $P_k^1 \in \{(1, j), (n_r + 1, j), (i, 1), (i, n_c + 1)\}$. If the adjacent vertex was visited, then the contour segment ends, else P_k^1 is the anchor point of a contour segment and the adjacent vertex is next to visit.

(b) P_k^2 has degree two. If both adjacent vertices are unvisited, the vertex is a double anchor point (case discussed in Section 2.3), else Γ_k continues to the remaining unvisited adjacent vertex.

(c) P_k^3 has degree three. When a P_k^3 vertex is reached, it can have: no unvisited adjacent vertices, one unvisited adjacent vertex to visit (P_k^3 is an anchor point), or two unvisited adjacent vertices. For the last case, if one of the unvisited adjacent vertices is encoded by $s_i = 0$, then Directive T1 is used to select the next adjacent vertex to visit, else Directive T2 is used due to the constraint of search for anchor points (see Section 2.3). In Fig. 2, all possible ways of selecting an adjacent vertex for a P_k^3 vertex are shown. The consequences of using Directives T1 and T2 are: **(i)** a vertex encoded using $s_i \in \{1, 2\}$ is a possible anchor point for a contour segment; **(ii)** a contour segment with an anchor point $P_1 = P_k^3$ has a known P_2 vertex; **(iii)** the cases shown in Figs. 2(a-d) are impossible.

(d) P_k^4 has degree four, and is the crossing of two contour segments. The first time a P_k^4 vertex is reached, Directive T3 is used to decrease the number of anchor points. The second time a P_k^4 vertex is reached, $s_i = 0$ is used to visit the last adjacent vertex, and since this case is deterministic for the decoder, s_i is not encoded. Directive T4 is used to differentiate a P_k^4 vertex from a P_k^3 vertex, since Directive T1 was used for a P_k^3 vertex.

2.2. Anchor points search

The paper uses the column-wise search for part of the anchor points, since it guarantees that such a found anchor point $P_1 = (i, j)$ has unvisited adjacent vertices $(i, j+1)$ and $(i+1, j)$. The remaining

anchor points are found by checking the list Ψ of possible relative anchor points. The list Ψ contains all the vertices encoded by a symbol $s_i \neq 0$, for which Directives C (see Fig. 1) cannot be used.

Figs. 2(e-l) shows the reason why Directive T2 is used: the second time $P^3_{k+1} = (i, j)$ is reached, the adjacent vertices $(i, j+1)$ and $(i+1, j)$ are visited, and therefore P^3_{k+1} is not an anchor point. The cases from Figs. 2(f-k) are found using Directive C1. Figs. 2(m-p) shows the cases where P^3_{k+1} is a possible anchor point, with a known P_2 position (see Section 2.1).

Fig. 3 shows the cases where a chain-code symbol $s_k = 2$ is generated. The vertex $P_{k+1} = (i, j)$ is not an anchor point in the following cases: **(i)** in Figs. 3(a,c) the vertices $(i, j+1)$ and $(i+1, j)$ are visited the first time P_{k+1} is reached; **(ii)** in Fig. 3(b) P_{k+1} may only be the P^3_{k+1} vertex from Fig. 2(b) (impossible case), or from Fig. 2(h) (not an anchor point); **(iii)** in Fig. 3(g) P_{k+1} may only be the P^3_{k+1} vertex from Fig. 2(c) (impossible case), or from Fig. 2(i) (not an anchor point). These cases are found by Directive C2.

2.3. Anchor points classification

The information about the anchor points is stored in two arrays initially full of zeros: Υ, the matrix of anchor points of size $n_r \times n_c$, and Φ, the binary vector of flags selecting the relative anchor points from the list Ψ. The anchor points are classified as:

(a) Edge anchor point, $P_1 = P^1_k$, where P_2 is determined as follows: if $P_1 = (i, 1)$, then $P_2 = (i, 2)$; if $P_1 = (1, j)$, then $P_2 = (2, j)$. These anchor points are stored by setting $\Upsilon(P^1_k) = 1$.

(b) Double anchor point, $P_1 = P^2_k = (i, j)$, has the vertices $(i, j+1)$ and $(i+1, j)$ unvisited, and any of them can be selected as P_2. We first select $P_2 = (i, j+1)$. If current Γ_k is not 'closed' ($P_1 \neq P_{n_{\Gamma_k}}$), then a next contour segment has $P_1 = P^2_k$ and $P_2 = (i+1, j)$. These anchor points are stored by setting $\Upsilon(P^2_k) = 1$.

(c) Relative anchor point, $P_1 = P^3_k$, it is stored in Ψ at the current index ℓ. If $s_k = 1$ encodes the next vertex in Γ_k and Directive C1 cannot be used (or if $s_k = 2$ and Directive C2 cannot be used), then: **(i)** if P^3_{k+1} is a relative anchor point, set $\Phi(\ell) = 1$; **(ii)** increment ℓ. These anchor points are found using the internal list Ψ and are encoded by the vector Φ. Hence, any encoded vertex P_k is signaled in Υ using the symbol 2 ('ignore position'), if it was not already marked (i.e. $\Upsilon(P_k) \neq 1$.).

First two types are found by the column-wise search, and the last type is found while traversing a contour segment.

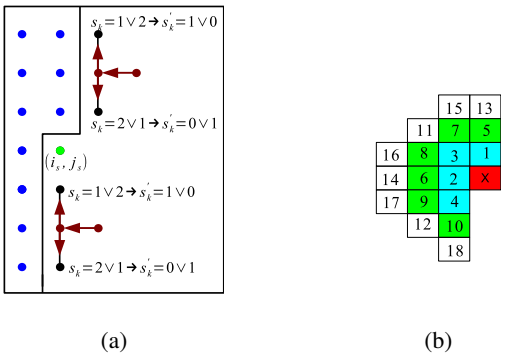
3. DEPTH MAP COMPRESSION

The depth map I is compressed using the set Γ and the depth value for each region. Section 2 showed that APC is coding the set Γ using the arrays: S, Φ, and Υ. The entropy coding of each array is described below, while the depth values are encoded (in about 10% of the compressed file) using *Algorithm Y* from [5].

3.1. Chain-codes entropy coding

The vectors S is encoded using the Context Tree Algorithm [15, 16, 17], with a context tree $\mathcal{T}$, built semi-adaptively, of maximum tree-depth $d_{\mathcal{T}} = 17$. Before coding S, the following deterministic changes are done (the decoder detects and reverses the changes):

(a) Reducing the number of symbols 2. When an anchor point $P_1 = (i_s, j_s)$, encoded using Υ, is found, it guarantees that there is no unvisited contour edge in the previous $j_s - 1$ columns and $i_s - 1$ lines in column j_s of the contour graph. Hence, a vertex $P_k = (i, j)$ with $i < i_s$ and $j = j_s + 1$ (or with $i \geq i_s$ and $j = j_s$) has maximum three unvisited adjacent vertices. When we reach $P_k = (i, j)$ and $P_{k-1} = (i, j+1)$, then possible unvisited adjacent vertices are $P_{k+1} = (i-1, j)$ and $P_{k+1} = (i+1, j)$. In this case $s_k = 0$ is not possible, and we remap the possible symbols 1 and 2 as shown in Fig. 4(a).



(a) (b)

Fig. 4. (a) Reducing the number of symbols $s_k = 2$. The blue dots are vertices checked for anchor points, an arrow indicates the next vertex to be visited, a red (black) dot is a visited (unvisited) vertex, and the green dot is the anchor point. (b) The context used for encoding the matrix Υ, where 'x' is the position to be encoded.

(b) Changing the context tree. The optimal ternary context tree for S is unbalanced: subtree '0' has many leaf nodes, while subtree '2' has a few leaf nodes. When studying the context tree of a general 3OT chain-code vector, we notice that in the context $(s_{k-2} \neq 2,\ s_{k-1} = 2)$ $s_k = 1$ is more frequent then $s_k = 0$, i.e. the probability to encode a symbol 1 is higher than the probability to encode a symbol 0. Hence, $s_k = 1$ is changed into $s'_k = 0$, and $s_k = 0$ into $s'_k = 1$. This introduces also some 'branch interchanges' between '0' and '1', but overall the change has a positive effect.

3.2. Coding the vector of relative anchor points

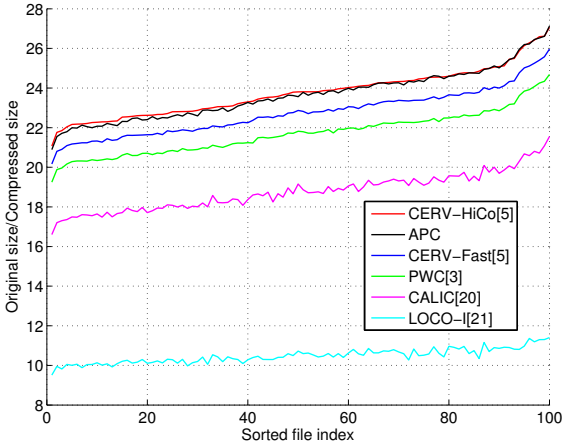
The vector Φ is encoded using the Context Tree algorithm with the tree-depth $d_{\mathcal{T}} = 17$. The tests showed that Φ contains about 10% symbols 1. Because the optimal context tree is unbalanced, i.e. it has only one long branch (branch '0') with a few leaf nodes, the threshold at which the symbol frequencies are halved in the arithmetic coder was set to 511. Φ encodes $3.5 \div 4.9$ bits/anchor point.

3.3. Coding the matrix of anchor points

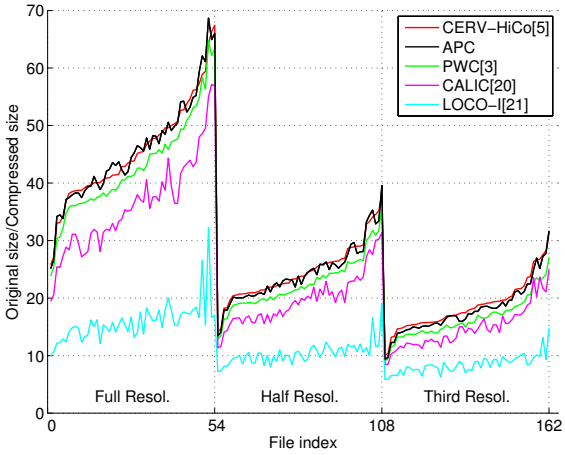
Matrix Υ is encoded the last in column-wise scanning, using a bi-dimensional context of length $d_{\mathcal{T}} = 18$, see Fig. 4(b). After pruning, the optimal context tree has about 7 leaf nodes. The context alphabet has three symbols, while the coding distribution has two symbols (symbol 2 is ignored). At the decoder, if $\Upsilon(i, j) = 1$ is decoded, then the contour segment Γ_k with $P_1 = (i, j)$ is decoded and Υ is updated.Υ encodes $9.9 \div 10.9$ bits/anchor point.

4. EXPERIMENTAL RESULTS

Three datasets are used for simulations: the *Breakdancers* and *Ballet* sequences [18], and the *Middlebury* dataset [19]. All the images are compressed losslessly. The GSOm algorithm from [13] is used to obtain lossy images for each selected image.

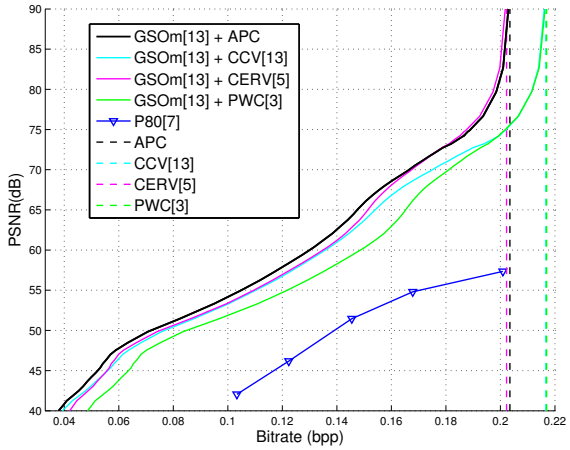


(b) *Breakdancers* sequence.

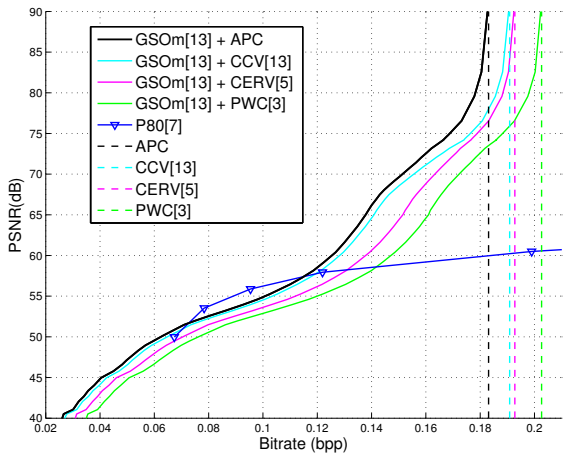


(c) *Middlebury* dataset.

Fig. 5. Lossless compression of depth map images.



(a) *Aloe*, full-size resolution, left view.



(b) *Bowling1*, full-size resolution, left view.

Fig. 6. Lossless compression of lossy images.

Fig. 5 shows the results for lossless compression. APC is compared with the state of the art methods: CERV [5] with two versions CERV-HiCo and CERV-Fast, PWC [3], CALIC [20], and LOCO-I [21] (the JPEG-LS implementation). One can see that APC obtains good results comparing with all the methods, and almost similar results with CERV-HiCo. To compare APC with the other methods, a vector W is introduced. It computes a gain percentage $W_k = \frac{1}{100}\left(\frac{\sum_i Method(I_i)}{\sum_i APC(I_i)} - 1\right)$ for APC over a method for a dataset, where $APC(I_i)$ is the APC size of the compressed file, for the image I_i from a dataset. W contains, in order, the values computed for CERV-HiCo, CERV-Fast, and PWC: for *Breakdances* we have $W = [-0.46 \quad 3.94 \quad 8.95]^T$, i.e. CERV-HiCo has a mean gain of 0.46%, while for example APC has a mean gain of 8.95% over PWC; for *Ballet* we have $W = [-0.59 \quad 3.62 \quad 8.80]^T$; for *Middlebury* $W = [-1.06 \quad 0.96 \quad 7.07]^T$, while for *Middlebury* only full-size resolution $W = [0.13 \quad 2.01 \quad 6.60]^T$.

Fig. 6 shows the results for lossy compression[1]. For two images from *Middlebury*, GSOm is used together with APC, CERV-HiCo [5], PWC [3], and CCV [13] (combining [4] and [5]). The "Proposed_80" algorithm from [8], denoted here $P80$, was also used as a comparison method. APC obtains better results comparing with the other methods, and even if the CERV compresses better than APC the initial image, at about 65 dB APC starts to obtain better results.

[1]For results over the images used in [13] see www.cs.tut.fi/~schiopu/APC

5. CONCLUSIONS

The paper presented a contour coding method by generating sparse arrays for encoding anchor points for chain-code strings. APC has good results compared with other lossless or lossy methods. Comparing APC and CERV we draw the following conclusions:

(a) Similar results are obtained for lossless compression.

(b) For lossy compression, when GSOm is used together with APC, the gains in PSNR, at certain bitrates, for the selected images, are in the order of few dB when compared to GSOm+CERV.

(c) In terms of runtime, a comparison cannot be made since both implementations are not optimized. APC has a smaller runtime then CERV when encoding lossy images, since the number of contour edges is smaller, and Φ tends to become sparser than in lossless case.

APC treats separately the information about anchor points and contour segments, and can be modified for embedded coding.

6. REFERENCES

[1] K.Y. Kim, G.H. Park, and D.Y. Suh, "Bit-plane-based lossless depth-map coding," *Optical engineering*, vol. 49, no. 6, pp. 067403–1–10, 2010.

[2] J. Heo and Y.-S. Ho, "Improved CABAC design in H.264/AVC for lossless depth map coding," in *Proc. IEEE ICME*, Barcelona, Spain, Jul. 2011, pp. 1–4.

[3] P.J. Ausbeck Jr., "The piecewise-constant image model," *Proc. IEEE*, vol. 88, no. 11, pp. 1779 – 1789, Nov. 2000.

[4] I. Schiopu and I. Tabus, "Depth image lossless compression using mixtures of local predictors inside variability constrained regions," in *Proc. IEEE ISCCSP*, Rome, Italy, May 2012, pp. 1–4.

[5] I. Tabus, I. Schiopu, and J. Astola, "Context coding of depth map images under the piecewise constant image model representation," *IEEE Transactions on Image Processing*, vol. 22, no. 11, pp. 4195–4210, Nov. 2013.

[6] G. Carmo, M. Naccari, and F. Pereira, "Binary tree decomposition depth coding for 3D video applications," in *Proc. IEEE ICME*, Barcelona, Spain, Jul. 2011, pp. 1–6.

[7] Y. Morvan, D. Farin, and P.H.N. de With, "Depth-image compression based on an R-D optimized quadtree decomposition for the transmission of multiview images," in *Proc. IEEE ICIP*, SA, TX, USA, Oct. 2007, vol. 5, pp. V–105 – V–108.

[8] R. Mathew, D. Taubman, and P. Zanuttigh, "Scalable coding of depth maps with R-D optimized embedding," *IEEE Transactions on Image Processing*, vol. 22, no. 5, pp. 1982–1995, May 2013.

[9] X. Zhang, J. Zou, X. Gu, and H. Xiong, "A scalable depth coding with arc breakpoints based synthesis in 3-D video," in *Proc. IEEE BMSB*, London, UK, Jun. 2013, pp. 1–5.

[10] Y. Li and L. Sun, "A novel upsampling scheme for depth map compression in 3DTV system," in *Proc. PCS*, Nagoya, Japan, Dec. 2010, pp. 186–189.

[11] K. Samrouth, O. Deforges, Y. Liu, F. Pasteau, M. Khalil, and W. Falou, "Efficient depth map compression exploiting correlation with texture data in multiresolution predictive image coders," in *Proc. IEEE ICMEW*, SJ, CA, USA, 2013, pp. 1–6.

[12] S. Milani, P. Zanuttigh, M. Zamarin, and S. Forchhammer, "Efficient depth map compression exploiting segmented color data," in *Proc. IEEE ICME*, Barcelona, Spain, Jul. 2011, pp. 1–6.

[13] I. Schiopu and I. Tabus, "Lossy depth image compression using greedy rate-distortion slope optimization," *IEEE Signal Processing Letters*, vol. 20, no. 11, pp. 1066–1069, Nov. 2013.

[14] H. Sanchez-Cruz and R. M. Rodriguez-Dagnino, "Compressing bi-level images by means of a 3-bit chain code," *Optical engineering*, vol. 44, no. 9, pp. 1–8, 2005.

[15] B. Martins and S. Forchhammer, "Tree coding of bilevel images," *IEEE Transactions on Image Processing*, vol. 7, pp. 517–528, Apr. 1998.

[16] J. Rissanen, "A universal data compression system," *IEEE Transactions on Information Theory*, vol. 29, pp. 656–664, Sep. 1983.

[17] M. Weinbergera, J. Rissanen, and R. Arps, "Applications of universal context modeling to lossless compression of gray-scale images," *IEEE Transactions on Image Processing*, vol. 4, pp. 575–586, Apr. 1996.

[18] C.L. Zitnick, S.B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High-quality video view interpolation using a layered representation," in *Proc. ACM SIGGRAPH*, LA, CA, USA, Aug. 2004, pp. 600–608.

[19] H. Hirschmuller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *Proc. IEEE CVPR*, Minneapolis, MN, USA, Jun. 2007, pp. 1–7.

[20] X. Wu and N. Memon, "Context-based, adaptive, lossless image coding," *IEEE Transactions on Communications*, vol. 45, no. 4, pp. 437–444, Apr. 1997.

[21] M. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," *IEEE Transactions on Image Processing*, vol. 9, no. 8, pp. 1309–1324, Aug. 2000.