LOSSLESS CONTOUR COMPRESSION USING CHAIN-CODE REPRESENTATIONS AND CONTEXT TREE CODING

Ionut Schiopu and Ioan Tabus

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

ABSTRACT

In this paper we experimentally study the context tree coding of contours in two alternative chain-code representations. The contours of the image regions are intersecting, resulting in many contour segments, each contour segment being represented by its starting position and a subsequence of chain-code symbols. A single stream of symbols is obtained, under a given ordering of the segments, by concatenating their subsequences. We present efficient ways for the challenging task of ordering the contour segments. The necessary starting positions and the chain-code symbols are encoded using context tree coding. The coding is performed in a semi-adaptive way, designing at the encoder by dynamical programming the context tree, whose structure is encoded as side information. The symbols are encoded using adaptively collected distributions at each context of the encoded tree. We study for depth map images and fractal images the tree-depth of the optimal context tree and the similarity of the context trees resulted to be optimal for each image.

1. INTRODUCTION

Contours compression is a well-studied topic, needed in several applications, e.g., transmission of image segmentations, object based video coding, depth map image compression, and transmission of digital maps. Choosing the best contour representations plays an important role in the efficiency of the contour compression. In image segmentation applications, the segmentation is stored using the contour map that contains contour segments, which are usually compressed lossless.

In depth image compression the contour compression is used in a lossless [1] or lossy [2] approach where the initial image is represented using two entities: an image segmentation (contour map), that divides the initial image into regions characterized by different properties; and a region value map. At the decoder, both entities are used together with a region filling algorithm to obtain the initial image.

The most used techniques for representing the contour maps are based on the principle of chain coding, using one of the available forms. In [3] vector and raster maps are compressed using the Freeman F8 chain-code, first introduced by Freeman [4]. In [5] various chain codes, as F4, AF4, F8, AF8, and 3OT were compared for representing and lossless compressing by context tree coding the contours of bi-level images, resulting in 3OT and AF4 being the best performers.

In contrast to the previous methods, that are using 1D contexts over the sequence of chain-codes, a method using 2D contexts was shown recently to have the best performance for compressing the depth image contours [6].

1.1. Preview

In this paper we study two chain-code alternative representations for lossless contour compression: the three-orthogonal direction (3OT) representation and the differential Freeman (AF4) representation both used in four-connectivity neighborhood.

Depth map images and fractal images are used in the experiments since these types of images contain areas characterized by a high density zones of contour segments and contour segment intersections, where from one contour point there is not only one way to continue describing the contour, but also two or three other ways. Hence, we studied different methods for ordering the contour segments and creating one string of symbols from the contour map. To signal the starting positions of the contour segments and intersection points of the contour segments, we introduce the matrix of anchor points, Υ.

The last step in the algorithm is to entropy encode the strings of symbols with the *Context Tree Algorithm*. Different context trees are designed by using different tree-depths when encoding a bitstream. The plot codelength versus encoding runtime can be used for selecting the context tree which offers the best trade-off performance-time complexity. The similarity and the robustness of the obtained context trees are studied over several databases.

1.2. Notations

The input of our algorithm is the initial (depth/fractal) image I, a matrix with n_r rows and n_c columns. Any segmentation image can be used as an input as long as a contour segment, Γ_k, separates the regions, Ω_k, characterized by different values in the input image. For each pixel (x, y), an integer $I_{x,y} \in \{0, 1, \ldots, 2^{B-1}\}$ is stored using B bits to define the different regions of the segmentation. In out tests we used $B = 8$ bits.

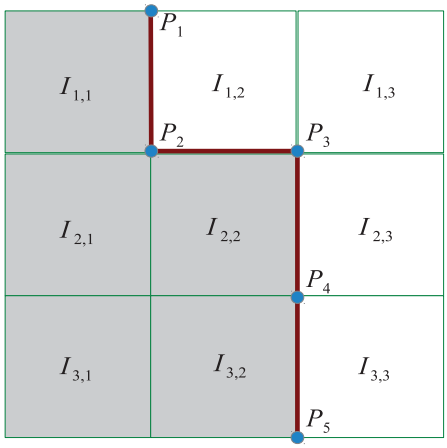


Figure 1. A 3×3 image representation: each pixels of the image is represented by a square marked $I_{i,j}$ in the center, and bordered by four crack-edges. A crack-edge separating two pixels from different regions is called active and is marked in red; the inactive crack-edges are marked in green. Each active crack-edges is bordered by two vertices, denoted P_k and P_{k+1}, and marked in blue.

The contour map of the input image is represented by horizontal and vertical crack-edges. We denoted crack-edge the contour line segment that separates two neighboring pixels. If the pixels belong to two different regions, the crack-edge is called active; if the pixels belong to the same region, the crack-edge is called inactive. A horizontal crack-edge is set between two neighboring pixels from two image rows, while a vertical crack-edge is set between two neighboring pixels from two image columns. The ends of the active crack-edge are called vertices denoted P_k and P_{k+1}.

In the lossless contour compression active crack-edge are encoded using the position of their surrounding vertices. The contour map consists of n_Γ contour segments, and is stored using vectors of consecutive vertices on each contour segment, denoted $\Gamma_k, k = 1, 2, \ldots n_\Gamma$.

Figure 1 shows an example of an image containing two regions: $\Omega_1 = \{I_{1,1}, I_{2,1}, I_{2,2}, I_{3,1}, I_{3,2}\}$ marked in gray, and $\Omega_2 = \{I_{1,2}, I_{1,3}, I_{2,3}, I_{3,3}\}$ marked in white. The contour map separating the two regions contains one contour segment that is stored using the vertex representation as: $\Gamma_1 = [P_1\ P_2\ P_3\ P_4\ P_5]^T$.

Each contour line segment from the contour map is encoded using a chain-code representation: AF4 or 3OT (see Appendix A). The algorithm translates the vector of consecutive vertices Γ_k into a vector of chain-code symbols, denoted S^k_{AF4}, respectively S^k_{3OT} (the first two vertices in each vector, P_1 and P_2 need to be encoded separately).

2. THE ANCHOR POINTS MATRIX

The auxiliary information, regarding specific positions in the contour map, are encoded by signaling the (x, y) positions in a matrix Υ, called here the anchor points matrix. The matrix Υ, with n_r+1 rows and n_c+1 columns, is initially filled with zeros, while the anchor points are stored as $\Upsilon(x, y) = c$, where $c \in \{1, 2, 3\}$ is a symbol that signals the type of information stored in the position. Finally, Υ is vectorized and $vec(\Upsilon)$ is encoded using the Context Tree Algorithm.

2.1. Starting point positions

For encoding a vertex P_{k+2}, a chain-code symbol is generated using the previous two vertices P_k and P_{k+1}. Therefore, a chain-code vector encodes the elements of Γ_k, except the first two vertices P_1 and P_2.

There is no a priori information to encode P_1, that is why this auxiliary information is encoded using the anchor points matrix by signaling the vertex positions using the symbol $c = 1$, and by setting $\Upsilon(P_1) = c$. The position of the vertex P_2 is encoded by a F4 symbol generated using its relative position to P_1.

2.2. Vertex classification

Figure 1 shows that a vertex not on the edge of the image is a crossing point for two horizontal crack-edges and two vertical crack-edges. Hence, the vertices can be classified according to the number of neighboring vertices: two neighbors (majority type); three neighbors, denoted P^3_k; and four neighbors, denoted P^4_k. The chain code representation encodes the first type of vertices using one pass, while for the others a second pass is required.

In a column search for the starting position, P_1, of an contour segment Γ_k, if the current vertex P_1 was never visited, then $P_1 \neq P^3_k$ and $P_1 \neq P^4_k$, since there will always be an unvisited neighboring vertex "up-ward" or "left" to its position. Hence, P_1 is a two neighbors type of vertex, but if Γ_k is not a closed contour segment, the vertex is the starting position of two contour segments that are saved using the vectors: Γ_k and Γ_{k+1}. This "double" starting positions are signaled by $\Upsilon(P_1) = 2$.

When a contour segment is translated into a vector, if a P^3_k vertex position is reached, there are two more options to continue to describe a contour segment. Since we want to have as may "0" chain-code symbols as possible in the vector, we impose the rule: if possible, "go forward", else try the following order list of symbol positions: $\rightarrow, \downarrow, \leftarrow, \uparrow$. After the next vertex is selected, P^3_k has only one unvisited neighboring vertex position, therefore P^3_k will be the first vertex (P_1) or the last vertex (ending point of the contour segment) of Γ_k.

For the P^4_k vertex type there are always two in-out passes, and since we want to have as may "0" chain-code symbols as possible we impose the rule: "go forward" from a P^4_k vertex. When encoding a vector of chain-code symbols, the information about its length is required, but Γ_k always ends in the edge of the image or with a P^3_k vertex. However, reaching the edge of the image is always detected. Encoding the length of the chain-code vector needs a large bitstream, therefor the following rule was set: at the decoder, the last element of the Γ_k is detected at the edge of the image or when a P^3_k vertex, with all of its three neighboring vertices visited, is reached. At the

second pass through P_k^4, its position can be misidentified as a P_k^3 vertex, that is why the P_k^4 vertex position are signaled by $\Upsilon(P_k^4) = 3$.

2.3. Γ_k selection strategies

The chain code representation is encoding contour segments. In an image with a complex contour there are a lot of vertices of P_k^3 and P_k^4 types that introduce auxiliary information encoded with a bits/vertex ratio much higher than a chain-code symbol. Each P_k^3 vertex introduces a starting position which together with the P_k^4 vertex position are encoded by Υ. Hence, the selection strategies for selecting Γ_k from the contour map, plays an important role in contour compression.

A first approach is to search on the edges of the image for the first vertex position P_1 of a contour segment. From the first vertex column the first vertex is selected as P_1 for Γ_1, and go through the contour segment until the current vertex position has no more neighboring active vertices. Below we introduce two strategies to select the contour segments.

The vectors of chain-code symbols are ordered in one vector of chain-code symbols, denoted S_{AF4}, respectively S_{3OT}, in the order of the starting positions P_1.

2.3.1. Anchor point position (APP)

The strategy is based on a repetitive column-wise search for the P_1 vertex positions in the contour map. The matrix Υ is populated with a lot of auxiliary information signaled using the following symbols: "1", "2", and "3".

2.3.2. Relative position (RP)

This strategy has a slightly different perspective: when we go through each contour segment Γ_k, the P_k^3 vertex positions are saved in a vector of possible starting positions, denoted Φ.

After the current contour segment is stored in Γ_k, the starting position P_1, for the next contour segment, is searched in Φ and is set as the first unvisited vertex of type P_k^3. Some of the P_k^3 vertices may already be visited as they could have been the end vertex for $\Gamma_{k-\tau}, \tau \geq 0$, and are removed form Φ together with the already selected elements. Every time $\Phi = \varnothing$, the column-wise search for the next starting position is done until $\Phi \neq \varnothing$, or until there are no more contour segments in the contour map matrix.

The starting positions and the P_k^4 positions are encoded using the matrix Υ and the selected elements from Φ, that are saved in a vector Ψ, whose elements contains the relative position of the starting positions P_1 in $\Gamma_{k-\tau}$.

3. CONTEXT TREE CODING ALGORITHM

The last step of the algorithm is the entropy coding of the vectors of the symbols using the context tree algorithm. More precisely, in this way are encoded: the vectors of chain-code symbols, S_{AF4}, respectively S_{3OT}; the vector of F4 symbols for encoding the position of the vertices P_2; and the anchor points matrix Υ.

The context tree algorithm encodes a string of symbols, denoted v, where each element belongs to a small alphabet having n_{sym} symbols. An important parameter for the algorithm is the tree-depth, d, to which the context tree is allowed to grow, i.e. the maximum context length of previous symbols used to encode the current symbol.

In literature there are a lot of methods on how to determine the optimal context tree, see [7, 8, 9]. The context tree algorithm implemented for this paper contains the following steps:

(T1.) Create the context tree $\mathcal{T}$ with the depth d. Each (parent) node contains n_{sym} leafs (child nodes), i.e. a total of $((n_{sym})^{d+1} - 1)/(n_{sym} - 1)$ nodes.

(T2.) First pass through v. For each node of $\mathcal{T}(d)$, the distribution of the symbols in the alphabet is collected. The values from each node represent the maximum number of appearances, for each symbol, in the context obtained by going through context tree, $\mathcal{T}(d, v)$, obtained for the v, from the current node to the root node.

(T3.) Go through $\mathcal{T}(d, v)$ and set for each node the estimated codelength, using the Laplace predictive probability, computed when the bitstream of symbols is encoded with the obtained distribution. For a better runtime, the Laplace predictive probability is computed using the Gamma function (see Appendix B).

(T4.) Prune $\mathcal{T}(d, v)$ using the rule: cut the child nodes of the current parent node, if the sum of estimated codelength of the children nodes and the estimated codelength of the part the of the bitsream that encodes the child nodes of the context tree, is greater than the estimated codelength of the parent node. We estimated the codelengh for encoding one child node as $CL_{child} = 1$ bit. The pruning process output is the optimal context tree $\mathcal{T}^*(d, v)$.

(T5.) Encode the context tree $\mathcal{T}^*(d, v)$ using a vector of symbols $\{0, 1\}$, by encoding: "0", if the current node is a tree branch (has child nodes); and "1", if the current node is a tree leaf (terminal node).

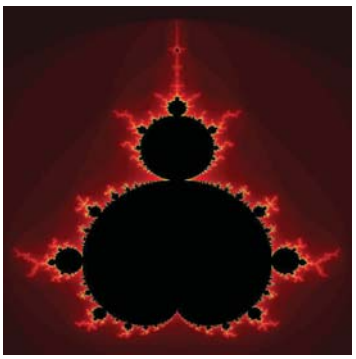
(T6.) Second pass: encode v using $\mathcal{T}^*(d, v)$.

When the input image I_i, from each datasets, is encoded, the information that has the highest codelength is the vector of chain-code symbols. Therefore, in this paper we study the optimal context tree $\mathcal{T}_i^* = \mathcal{T}^*(d, S_{AF4}(I_i))$ (respectively $\mathcal{T}_i^* = \mathcal{T}^*(d, S_{3OT}(I_i))$) obtained for the image I_i. We denote $\mathcal{L}(I_i, \mathcal{T}_i^*)$ the codelength needed to encode the contour of the image I_i using the optimal context tree $\mathcal{T}_i^*$ from the image I_i, and $\mathcal{L}(I_i, \mathcal{T}_j^*)$ the codelength needed to encode the contour of the image I_i using the optimal context tree $\mathcal{T}_j^*$ from the image I_j.

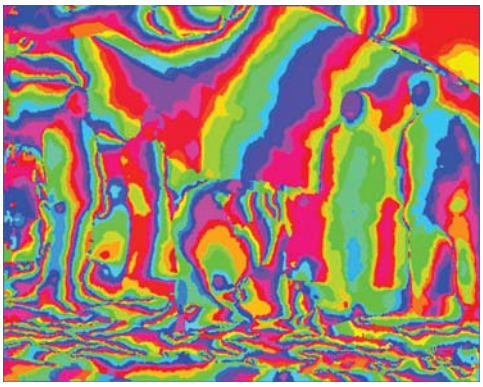
To measure the similarities of the optimal context trees, we introduced the similarity matrix S:

$$S(i, j) = \frac{\mathcal{L}(I_i, \mathcal{T}_j^*) - \mathcal{L}(I_i, \mathcal{T}_i^*)}{|S_{3OT}|}, \tag{1}$$

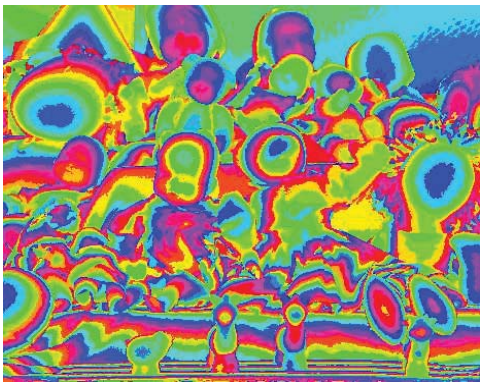
where $|S_{3OT}|$ is the length of the vector S_{3OT}. Furthermore, $S(i, j)$ represents the extra number of bits per each symbol (bps) needed to encode the contour of the image I_i, using a different context tree ($\mathcal{T}_j^*$) then the optimal context tree ($\mathcal{T}_i^*$).



(a) Fractal images: Mandelbrot set

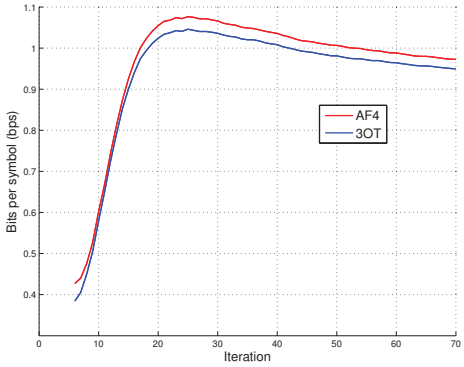


(b) Depth images: breakdancers dataset

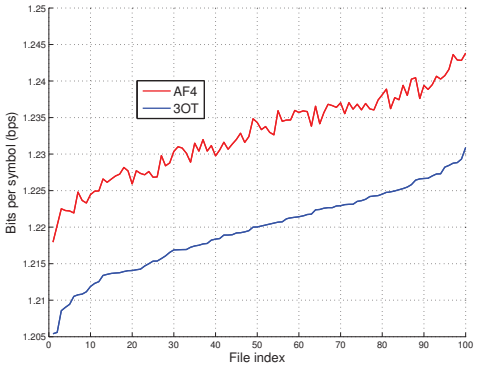


(c) Depth images: Middlebury dataset

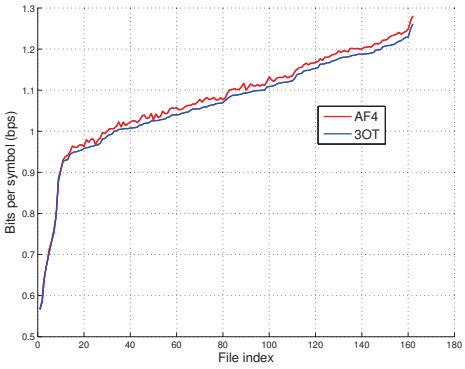
Figure 2. Pseudo-color images for: (a) Mandelbrot (60th iteration); (b) breakdancers (image *depth-cam0-f000*); and (c) Middlebury (*dolls*, third size, left view) dataset.



(a) Fractal images: Mandelbrot set



(a) Depth images: breakdancers dataset



(a) Depth images: Middlebury dataset

Figure 3. Comparison between AF4 and 3OT chain-code representation for: (a) Mandelbrot, (b) breakdancers, and (c) Middlebury dataset.

4. EXPERIMENTAL RESULTS

The experimental results are carried out using two types of images: fractal images and depth images. The contour of the fractal images is generated for the Mandelbrot set [10]. The images are obtained using the initializations:

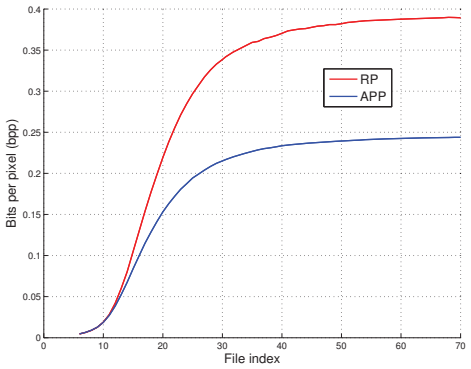
$$x_{min} = -2.1, x_{max} = 0.6, y_{min} = -1.2, y_{max} = 1.2,$$

for a grid of 1000×1000, and are saved from the sixth iteration to the 70th iteration (65 images). Figure 2(a) shows a pseudo-color image for the 60th iteration.

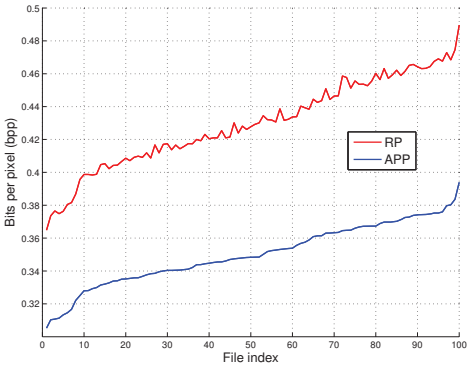
The contour of the depth images from two dataset was also used in our experiments: breakdancers [11], view 1 (cam0), frames $0 - 99$ (100 images), see Figure 2(b) for a pseudo-color image of frame 0; and Middlebury [12] (162 images), see Figure 2(c) for a pseudo-color image of the *dolls* image.

4.1. Chain-code representations

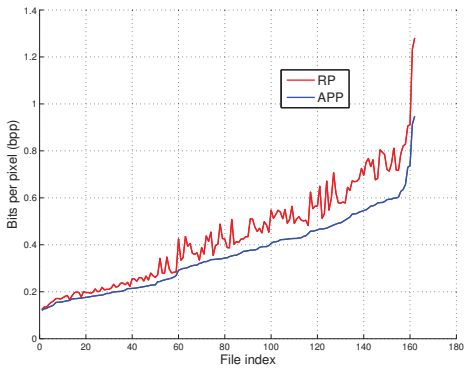
In the first sequence of tests, the AF4 and 3OT representations are studied. In Figure 3 results for the three datasets are presented as plots of bits per symbol (bps) vs. file index (or iteration). The tests are carried out using context tree-depth $d = 25$ and using the APP strategy for generating Γ_k. The bps value is calculated as the sum for the bits required for encoding the optimal context tree



(a) Fractal images: Mandelbrot set

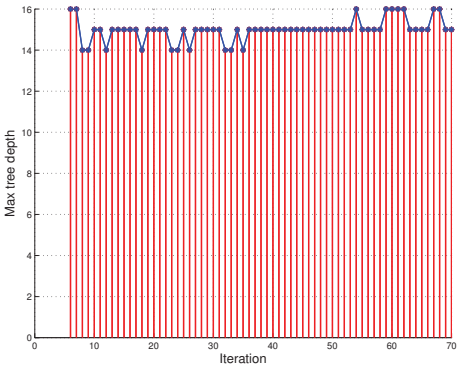


(b) Depth images: breakdancers dataset

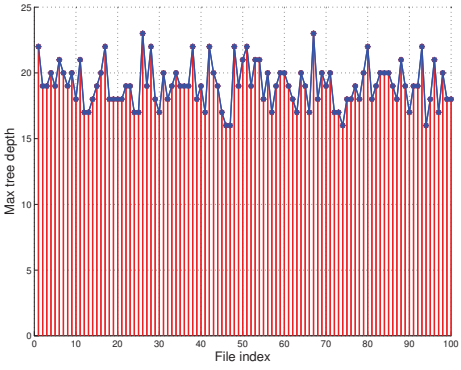


(c) Depth images: Middlebury dataset

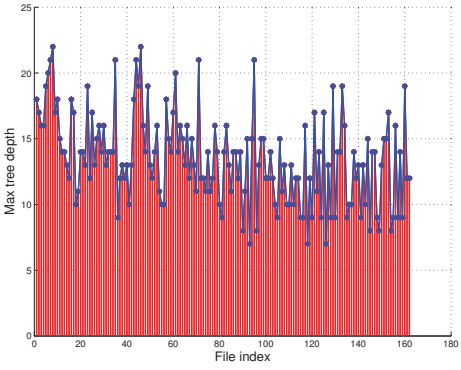
Figure 4. Comparison between APP and RP strategies for generating Γ_k for: (a) Mandelbrot, (b) breakdancers, and (c) Middlebury dataset.



(a) Fractal images: Mandelbrot set



(b) Depth images: breakdancers dataset



(c) Depth images: Middlebury dataset

Figure 5. Tree-depth for each image in the: (a) Mandelbrot; (b) breakdancers; and (c) Middlebury dataset.

$\mathcal{T}^*(25, S_{3OT})$ (resp. $\mathcal{T}^*(25, S_{AF4})$), and the vector of chain-code symbols S_{3OT} (resp. S_{AF4}) using the context tree, divided by the number of symbols, $|S_{3OT}|$ (resp. $|S_{AF4}|$).

Figure 3(a) shows that the $3OT$ representation generally obtains better results than the $AF4$ representation, that is why all other tests are conducted using the $3OT$ chain code representation.

4.2. Γ_k selecting strategies

In the second sequence of tests the two strategies, APP and RP, for generating the vectors of vertices Γ_k are compared. Tests are carried out using the context tree-depth $d = 20$.

In Figure 4 results for the three datasets are presented as plots of bits per pixel (*bpp*) vs. file index (or iteration). The figure shows the codelength obtained by compressing the contour information, since the strings of symbols obtained with the two strategies are different. APP strategy obtains better results and it was used in all other tests.

4.3. Context tree-depth

In the third sequence of test, for selecting an appropriate value for the context tree parameter d, the test are conducted using, $d = 25$, so that the tree-depth of the optimal context tree can be determined. Figure 5 shows that the tree-depth d^* for $\mathcal{T}^*$ varies between 17 and 25.

Using $d = 25$, the initial created balanced context tree

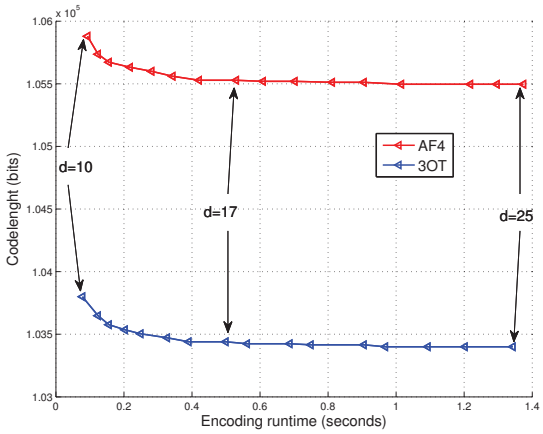


Figure 6. Codelength obtained for different values for the context tree-depth, d, and the encoding runtime for $Bowling2$ (half size, left view), from Middlebury dataset.

has $(3^{26}-1)/2$ nodes, a large enough number of nodes. Therefore, the gain obtained by increment d^*_{max} was studied. Figure 6 shows the codelength value obtained by setting $d = 10, \ldots, 25$ and the encoding runtime for the two chain-code representation. One can see that the runtime price paid for using a large d is very high compared to the codelength gain. Hence, $d^* = 17$ was selected as the appropriate value.

4.4. Similarity of the context trees

In the last sequence of tests we used: the $3OT$ representation, APP strategy, and set $d = 17$. Figure 7(a) shows the surface representation of the similarity matrix S for the Mandelbrot set. The images obtained for the last iterations have the optimal context tree $\mathcal{T}^*$ similar since only a few values are modified in the selected grid (a few pixels are different).

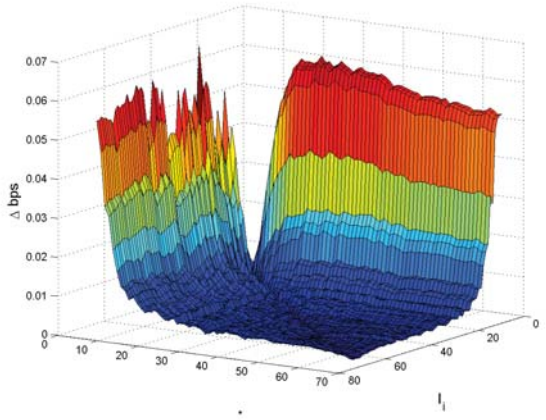
Figures 7(b) shows the surface or the breakdancers dataset. One can see that the sequence of images has similar $\mathcal{T}^*$: the maximum variation is 0.0024 bps, however the optimal is always $\mathcal{T}^*_i$.

Figures 7(c) shows the surface for Middlebury, where the images have different optimal context trees: the maximum variation is 0.0531 bps.

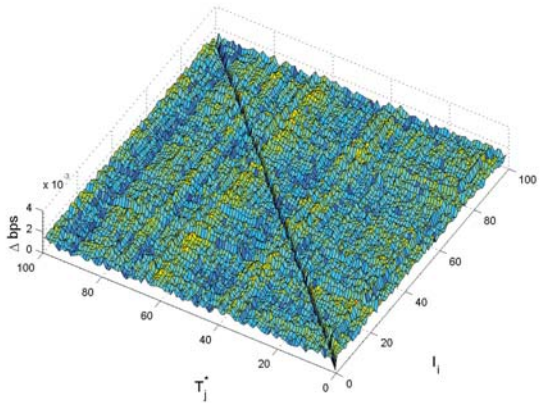
5. CONCLUSION

In this paper we studied the lossless contours compression of depth and fractal images by representing the contour in two alternative chain-code representations. The simulation has shown that the $3OT$ representation obtains generally better results comparing with the $AF4$ representation. Two strategies for selecting and ordering the contour segments were presented and the Anchor Point Position (APP) strategy, based on searching the starting positions P_1 for each Γ_k in a column-wise way, was the best.

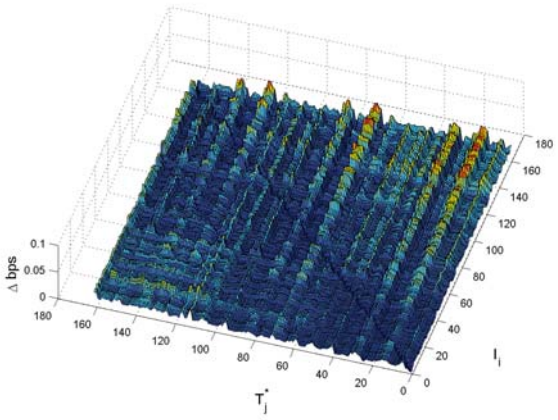
We studied the context tree coding and selected the appropriated context tree-depth as $d = 17$, due to good encoding performance in a small runtime.



(a) Fractal images: Mandelbrot set



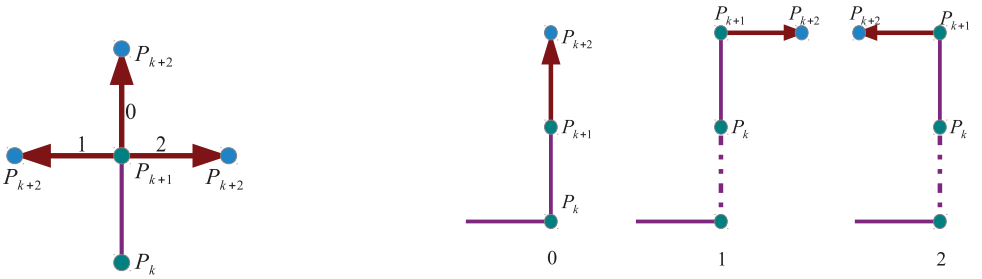
(b) Depth images: breakdancers dataset



(c) Depth images: Middlebury dataset

Figure 7. The similarity matrix of the optimal context trees of the images from: (a) Mandelbrot, (b) breakdancers, and (c) Middlebury datasets.

The similarity of the context trees of the images in different datasets was also studied and the results showed that the optimal context trees of a sequence of images are similar, but only the optimal context tree for each images obtains the best result.



(a) AF4 representation

(b) 3OT representation

Figure 8. Representing the position of the unknown vertex P_{k+2}, marked with blue, using the position of the known vertices P_k, P_{k+1}, marked with turquoise, and using one: (a) AF4 symbol; (b) 3OT symbol. The unknown active crack-edges are marked with red arrows; the known active crack-edge is marked with magenta.

APPENDIX A. Chain-code representations

The chain-code representation is a simple way to encode a contour segment described using a vertex representation $\Gamma_k = [P_1, P_2, \ldots, P_k, P_{k+1}, P_{k+2}, \ldots]$. A chain-code symbol encodes the position of a vertex, P_{k+2}, using its relative position to the previous two vertices, P_k and P_{k+1}, in Γ_k.

F4 and AF4 chain code representation

In the 4-connectivity neighborhood, there are four possibilities to move from one vertex position to another: "upward" ($\uparrow$), "downward" ($\downarrow$), "right" ($\rightarrow$) and "left"($\leftarrow$). Therefore, it was natural to create a representation using an alphabet of these four symbols, called F4 representation. The current position is reached using one of this four symbols, that is why the position of P_{k+1} relative to P_k can be encoded using an alphabet of three symbols since the possibility of going back is excluded, $P_{k+1} \neq P_{k-1}$.

Differential Freeman (AF4) chain-code is an example of code that is based on this idea. The AF4 representation uses the alphabet: "0" - "go forward"; "1" - "go left", and "2" - "go right" (see Figure 8(a)).

3OT chain code representation

The second chain-code representation is the 3-orthogonal direction (3OT) representation. It is based on the three relative changes of orthogonal directions and uses two parameters: orientation and direction. The orientation refers to a horizontal or vertical movement. If the orientation is "horizontal", the direction can be "upward" or "downward", while if the orientation is "vertical", then the direction can be "left" or "right".

The representation has an alphabet of three symbols (see Figure 8(b)): "0" or "go forward", the direction and orientation are not changed; "1" or "change orientation", the orientation is changed, and the direction remains the same; "2" or "go back", both orientation and direction are changing.

APPENDIX B. Laplace predictive probability using Gamma function

Lets assume a sequence of symbols $x_i, i = 1, \ldots, n_x$, with the elements belonging to alphabet $\{s_1, s_2, \ldots, s_{ns}\}$, with the distribution of symbols $(N_1, N_2, \ldots, N_{ns})$, obeying: $\sum_{i=1}^{ns} N_i = n_x$.

The estimated codelength computed using the Laplace predictive probability of the bitstream is written as:

$$CL = -\log_2 \left(\frac{\prod_{i=1}^{N_1}(i) \cdot \ldots \cdot \prod_{i=1}^{N_{ns}}(i)}{ns \cdot \ldots \cdot (ns + n_x - 1)} \right). \quad (2)$$

Gamma function for positive integers is defined as: $\Gamma(n) = (n-1)!$. It is easy to see that now:

$$\begin{aligned} CL &= -\sum_{i=1}^{ns} (\log_2(\Gamma(N_i + 1))) - \log_2(\Gamma(ns)) \\ &\quad + \log_2(\Gamma(n_x + ns)). \end{aligned} \quad (3)$$

The Gamma function can be estimated using Rocktaeschel $\ln(\Gamma(z))$ approximation combined with $\Gamma(z)$ approximation, in the Stirling approximation:

$$\ln(\Gamma(z)) = z\ln(z) - z - \frac{\ln(\frac{z}{2\pi})}{2} + \frac{1}{12z} - \frac{1}{360z^3}. \quad (4)$$

6. REFERENCES

[1] I. Schiopu and I. Tabus, "Depth image lossless compression using mixtures of local predictors inside variability constrained regions," in *Proc. 5th IS-CCSP*, Rome, Italy, May 2012, pp. 1–4.

[2] I. Schiopu and I. Tabus, "Lossy and Near-Lossless compression of depth images using segmentation into constrained regions," in *Proc 20th EUSIPCO*, Bucharest, Romania, August 2012, pp. 1099 – 1103.

[3] A. Akimov, A. Kolesnikov, and P. Franti, "Lossless compression of map contours by context tree modeling of chain codes," *Pattern Recognition*, vol. 40, no. 3, pp. 944 – 952, March 2007.

[4] H. Freeman, "Computer processing of line-drawing images," *ACM Computing Surveys*, vol. 6, no. 1, pp. 57 – 97, March 1974.

[5] I. Tabus and S. Sarbu, "Optimal structure of memory models for lossless compression of binary image contours," in *Proc. ICASSP*, Prague, May 2011, pp. 809–812.

[6] I. Tabus and I. Schiopu amd J. Astola, "Context coding of depth map images under the piecewise-constant image model representation," *IEEE Trans. Image Process.*, vol. PP, no. 99, pp. 1–16, 2013.

[7] B. Martins and S. Forchhammer, "Tree coding of bilevel images," *IEEE Trans. Image Process.*, vol. 7, pp. 517528, April 1998.

[8] J. Rissanen, "A universal data compression system," *IEEE Trans. Inf. Theory*, vol. 29, pp. 656664, September 1983.

[9] M. Weinbergera, J. Rissanen, and R. Arps, "Applications of universal context modeling to lossless compression of gray-scale images," *IEEE Trans. Image Process.*, vol. 4, pp. 575586, April 1996.

[10] H.-O. Peitgen and P. H. Richter, *The beauty of fractals: images of complex dynamical systems*, Springer-Verlag, Berlin, Germnay, 1986.

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

[12] H. Hirschmuller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *Proc. IEEE Comput. Soc. Conf. CVPR*, Minneapolis, MN, USA, June 2007, p. 18.