TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Ionut Schiopu

Depth-Map Image Compression Based on Region and Contour Modeling

Julkaisu 1360 • Publication 1360

Tampere 2016

Tampereen teknillinen yliopisto. Julkaisu 1360
Tampere University of Technology. Publication 1360

Ionut Schiopu

Depth-Map Image Compression Based on Region and Contour Modeling

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB109, at Tampere University of Technology, on the 29th of January 2016, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2016

Supervisor:

Prof. Dr. Ioan Tabus,
Department of Signal Processing,
Faculty of Computing and Electrical Engineering,
Tampere University of Technology,
Tampere, FINLAND

Pre-examiners:

Prof. Dr. Adrian Munteanu,
Department of Electronics and Informatics,
Vrije Universiteit Brussel,
Brussel, BELGIUM

Assoc. Prof. Dr. Faouzi Alaya Cheikh,
Faculty of Computer Science and Media Technology,
Gjøvik University College,
Gjøvik, NORWAY

Opponent:

Prof. Dr. Søren Forchhammer,
Department of Photonics Engineering,
Technical University of Denmark,
Lyngby, DENMARK

ISBN 978-952-15-3667-0 (printed)
ISBN 978-952-15-3680-9 (PDF)
ISSN 1459-2045

I dedicate this thesis to my family.

Abstract

In this thesis, the problem of depth-map image compression is treated. The compilation of articles included in the thesis provides methodological contributions in the fields of lossless and lossy compression of depth-map images.

The first group of methods addresses the lossless compression problem. The introduced methods are using the approach of representing the depth-map image in terms of regions and contours. In the depth-map image, a segmentation defines the regions, by grouping pixels having similar properties, and separates them using (region) contours. The depth-map image is encoded by the contours and the auxiliary information needed to reconstruct the depth values in each region.

One way of encoding the contours is to describe them using two matrices of horizontal and vertical contour edges. The matrices are encoded using template context coding where each context tree is optimally pruned. In certain contexts, the contour edges are found deterministically using only the currently available information. Another way of encoding the contours is to describe them as a sequence of contour segments. Each such segment is defined by an anchor (starting) point and a string of contour edges, equivalent to a string of chain-code symbols. Here we propose efficient ways to select and encode the anchor points and to generate contour segments by using a contour crossing point analysis and by imposing rules that help in minimizing the number of anchor points.

The regions are reconstructed at the decoder using predictive coding or the piecewise constant model representation. In the first approach, the large constant regions are found and one depth value is encoded for each such region. For the rest of the image, suitable regions are generated by constraining the local variation of the depth level from one pixel to another. The nonlinear predictors selected specifically for each region are combining the results of several linear predictors, each fitting optimally a subset of pixels belonging to the local neighborhood. In the second approach, the depth value of a given region is encoded using the depth values of the neighboring regions already encoded. The natural smoothness of the depth variation and the mutual exclusiveness of the values in neighboring regions are exploited to efficiently predict and encode the current region's depth value.

The second group of methods is studying the lossy compression problem. In a first contribution, different segmentations are generated by varying the threshold for the depth local variability. A lossy depth-map image is obtained for each segmentation and is encoded based on predictive coding, quantization and context

tree coding. In another contribution, the lossy versions of one image are created either by successively merging the constant regions of the original image, or by iteratively splitting the regions of a template image using horizontal or vertical line segments. Merging and splitting decisions are greedily taken, according to the best slope towards the next point in the rate-distortion curve. An entropy coding algorithm is used to encode each image.

We propose also a progressive coding method for coding the sequence of lossy versions of a depth-map image. The bitstream is encoded so that any lossy version of the original image is generated, starting from a very low resolution up to lossless reconstruction. The partitions of the lossy versions into regions are assumed to be nested so that a higher resolution image is obtained by splitting some regions of a lower resolution image. A current image in the sequence is encoded using the a priori information from a previously encoded image: the anchor points are encoded relative to the already encoded contour points; the depth information of the newly resulting regions is recovered using the depth value of the parent region.

As a final contribution, the dissertation includes a study of the parameterization of planar models. The quantized heights at three-pixel locations are used to compute the optimal plane for each region. The three-pixel locations are selected so that the distortion due to the approximation of the plane over the region is minimized. The planar model and the piecewise constant model are competing in the merging process, where the two regions to be merged are those ensuring the optimal slope in the rate-distortion curve.

Preface

The research presented in this thesis has been carried out at the Department of Signal Processing (SGN) from the Faculty of Computing and Electrical Engineering of Tampere University of Technology (TUT), during the period of time from August 2011 to June 2015.

First and foremost, I wish to express my deepest appreciation and gratitude to my supervisor, Prof. Ioan Tăbuş, for his advice and support towards the elaboration of the thesis. I want to thank him for being a caring person, for believing in me and giving me the opportunity to work under his supervision.

I express my gratitude to Prof. Adrian Munteanu and Prof. Faouzi Alaya Cheikh for reviewing the initial manuscript and for providing constructive comments for a clearer presentation and a better organization of this thesis. I also would like to thank Prof. Søren Forchhammer for accepting the role of the opponent for my public thesis defense.

I gratefully thank all the administrative and teaching staff of the Department of Signal Processing and I owe my warmest thanks to Prof. Jaakko Astola for his generous support. I would like to mention Prof. Bogdan Dumitrescu and Prof. Atanas Gotchev and thank them for influencing my career. Special thanks go to Virve Larmila, Noora Rotola-Pukkila, Ulla Siltaloppi, Pirkko Ruotsalainen and Elina Orava for their assistance and help with administrative matters.

I would like to thank Prof. Dan Ştefănoiu from the Department of Automatic Control and Systems Engineering (ACSE) from my home university, University "Politehnica" of Bucharest (UPB), for introducing the field of Signal Processing to me, for truly believing in me and for being the earliest supporter of my career. I also want to thank Prof. Bogdan Dumitrescu, Prof. Cristian Oara, Prof. Corneliu Popeea, Prof. Boris Jora and Prof. Dumitru Popescu. Your courses were a tremendous learning experience for me, academically and personally.

I am deeply thankful to all my colleagues and friends from the Department of Signal Processing. I would like to thank Alexandru Onose, Florin Ghido, Stefan Uhlmann, Vlad Tăbuş, Defne Us, Petri Helin, Jenni Hukkanen, Septimia Sârbu, for the friendly atmosphere and memorable conversations and discussions. Especially, I would like to thank Alex and Florin for the invaluable scientific discussions that we had at lunch or coffee.

I wish to thank all my friends for all the good moments that they have brought into my life at our Saturday evening parties, Apina meetings, sport events or

other happenings. Thank you Marian and Alex for your friendship and for all the adventures we had together. I want to thank my Romanian friends Aurelian (a.k.a. Puiu), Laurenţiu (a.k.a. Lala) and Vlad for making me feel like home. I want to thank my friends Pattamone (a.k.a. Pat), Bree, Tan, Shriya, Tiina for all the joyful discussions. Thank you Stefan (a.k.a. Mr. German), Ricardo (a.k.a. Ricardinho), Anca and Vladimir, Sebastien, Defne and Paul, my Sunday football friends and all the other friends from all around the world, too many to mention here. Thank you Cristi for joining me in my first visit to Finland in February 2011. Thank you all for giving me the gift of friendship!

The financial support of Tampere Graduate School in Information Science and Engineering (TISE), TUT's Graduate School and The Foundation of Nokia Corporation is gratefully acknowledged.

Last but not least I want to thank my family for their unconditional love and support. I want to express my warmest gratitude to my parents, Aurelia and Marin, with the message: *"Vă mulţumesc pentru dragostea pe care mi-o purtaţi şi pentru suportul moral şi financiar fără de care nimic nu ar fi putut fi realizat"*; to my younger brother Gabriel and to my grandparents for their love. At the same time I want to tender my warm thanks to my girlfriend Cristina for believing in me, supporting me, caring and loving me.

November 2015, Tampere
Ionuţ Şchiopu

Contents

List of publications

The thesis is a compilation of seven publications: two journal articles [P3, P4] and five conference papers [P1, P2, P5, P6, P7]. The publications present methods that compress the depth-map images by first representing the image using different entities, by describing each entity using a sequence of symbols, and then encoding the sequences by taking advantage of their statistical proprieties.

Publications [P1, P3, P5] presents algorithms for lossless compression, while publication [P2, P4, P6, P7] presents algorithms for lossy compression.

[P1] I. Schiopu and I. Tabus. Depth image lossless compression using mixtures of local predictors inside variability constrained regions. In *Proc. International Symposium on Communications, Control, and Signal Precessing (ISCCSP)*, pages 1–4, Rome, Italy, May 2012.

[P2] I. Schiopu and I. Tabus. Lossy and Near-Lossless compression of depth images using segmentation into constrained regions. In *Proc. European Signal Processing Conference (EUSIPCO)*, pages 1099–1103, Bucharest, Romania, August 2012.

[P3] 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 (IEEE TIP)*, 22(11):4195–4210, November 2013.

[P4] I. Schiopu and I. Tabus. Lossy depth image compression using greedy rate-distortion slope optimization. *IEEE Signal Processing Letters (IEEE SPL)*, 20(11):1066–1069, November 2013.

[P5] I. Schiopu and I. Tabus. Anchor points coding for depth map compression. In *Proc. IEEE International Conference on Image Processing (IEEE ICIP)*, pages 5626–5630, Paris, France, October 2014.

[P6] I. Schiopu and I. Tabus. Lossy-to-lossless progressive coding of depth-maps. In *Proc. International Symposium on Signals, Circuits and Systems (ISSCS)*, pages 1–4, Iasi, Romania, July 2015.

[P7] I. Schiopu and I. Tabus. Parametrizations of planar models for region-merging based lossy depth-map compression. In *Proc. 3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*, pages 1–4, Lisbon, Portugal, July 2015.

List of algorithms

List of acronyms

3D Three Dimensions . . . 1
3DTV 3D Television . . . 1
DIBR Depth-Image-Based Rendering . . . 1
CERV Crack-Edge–Region–Value . . . 3
APC Anchor Points Coding . . . 3
GSO Greedy rate-distortion Slope Optimization . . . 3
P-GSO Progressive coding of GSO sequences . . . 3
RD Rate-Distortion . . . 3
LS Least Squares . . . 3
PF Plane Fitting . . . 4
GSOmPF GSOm with Plane Fitting . . . 4
3OT Three-OrThogonal . . . 4
F8 Freeman 8 symbols . . . 8
AF8 Differential Freeman 8 symbols . . . 9
F4 Freeman 4 symbols . . . 9
AF4 Differential Freeman 4 symbols . . . 9
L Laplace . . . 12
KT Krichevsky-Trofimov . . . 12
GR Golomb-Rice . . . 13
AMM Adaptive Markov Model . . . 14
N-AMM N-order Adaptive Markov Model . . . 15
CTM Context Tree Model . . . 15
TCM Template Context Model . . . 15
TCTM Template Context Tree Model . . . 16
DPCM Differential Pulse Coded Modulation . . . 16
MAP Median Adaptive Predictor . . . 17
JPEG Joint Photographic Experts Group . . . 16

Mathematical notations

(x, y)	Position in a matrix found at intersection of row x and column y.
H	The binary matrix of horizontal edges.
P_k	Vertex position in a contour graph.
T	Context tree.
V	The binary matrix of vertical edges.
Y_0	The template image of a GSOs sequence.
Y_τ	Image on the τ position of a GSOs sequence, generated at step τ.
Z_0	Initial depth-map image of a GSOm sequence.
Z_τ	Image on the τ position of a GSOm sequence, generated at step τ.
Γ	The contour of an image.
Γ_k	Contour segment formed of consecutive adjacent vertex positions.
Ω_ℓ	ℓ^{th} region in an image containing a set of pixel positions.
$\{\cdot\}^T$	Transpose operator applied to $\{\cdot\}$.
$\{\cdot\}^{-1}$	Inverse operator applied to $\{\cdot\}$.
$\{\cdot\}^*$	Optimal value of the parameter $\{\cdot\}$.
$\{\cdot\}^{(\tau)}$	Variable associated to image Z_τ.
$\lceil\{\cdot\}\rfloor$	Rounded (to closest integer) value of $\{\cdot\}$.
$\mathbf{x}^n$	A sequence of symbols containing n elements.
$\lvert\{\cdot\}\rvert$	Absolute value of $\{\cdot\}$.
d_ℓ	Truncated average depth value associated to region Ω_ℓ.
n_A	Number of elements found in vector A.
ns_A	Number of symbols in the alphabet found in vector A.
x_j	j^{th} element in the sequence $\mathbf{x}^n$.
$\mathcal{N}_4(x, y)$	Causal neighborhood of the pixel position (x, y) in 4-connectivity.
$\mathcal{T}$	Optimal context tree.
$\mathcal{Y}_n$	Sequence of n lossy images generated by GSOs.

$\mathcal{Z}_n$	Sequence of n lossy images generated by GSOm.
E[{·}]	Expected value of {·}.
sgn({·})	Sign of {·}.

Chapter 1

Introduction

"Begin at the beginning," the King said gravely,
"and go on till you come to the end: then stop."
— Lewis Carroll, *Alice in Wonderland*

1.1 Motivation

A depth-map image is an image that stores information about the distance from the optical center of the camera to the point in space represented in the image. Recently, the acquiring techniques for this type of images have improved a lot, and many types of specialized sensors are now available on the market. Even more, the depth-map estimation techniques based on computer vision tools were also improved and are now presenting much better results. Since the Three Dimensions (3D) experience is captivating the users, a wide range of applications that use depth-map images have been developed. Computer vision, gaming industry, movie industry, mobile phone industry, 3D Free Viewpoint Video, or 3D Television (3DTV) are only a few examples of research areas, where the applications use multiple cameras to capture views of the scene from different viewpoints. Every application aims to provide a very good 3D experience to the user, therefore a lot of research has been done to improve the quality of the acquired depth-map images. This research was focused on developing techniques which can compress the depth-map images either without any information loss, called lossless (image) compression, or at a good enough quality by accepting some information loss, called lossy compression.

The 3D effect is perceived by a human, who receives two color images, one for each eye. The synthesis of the color images, necessary for rendering a certain viewing angle of the scene, can be achieved using the Depth-Image-Based Rendering (DIBR) technique, starting from two or several color images and one or more depth-map images. A depth-map image is more redundant than a color image and can be compressed losslessly at a lower bitrate. The depth-map compression became an active research field in the recent years, and many new techniques have been proposed. Below we list briefly a few main approaches that were proposed

in the last several years. A depth-map image may be compressed by applying different decorrelating transforms [21, 39, 101]; by decomposing the image using a tree triangular decomposition [18]; by down-sampling the image and compressing a smaller size image [64]; by conserving the edges found in the image using *wedgelets* [23] or *platelets* [96]; by representing the image using pyramidal structures [40]; by applying state of the art bit-plane compressors (e.g. JBIG) [36, 101]; by transmitting to the decoder a segmentation of the image and encoding different entities to reconstruct the image [24, 57]; or by modifying video standards (e.g. H.264, HEVC) to compress a depth-map video sequence [49, 100, 102, 103].

In this dissertation, we choose to represent the depth-map image using a designed segmentation that divides the image into regions having pixels with similar proprieties. The developed algorithms are reconstructing the initial depth-map image at the decoder from an encoded segmentation and some auxiliary information, with which we are able either to recover the information in each region and to obtain exactly the same image, or to reconstruct the regions with a controlled distortion and to obtain a lossy version of the initial image.

1.2 General overview of the algorithms

The algorithms developed for this thesis are divided into two groups according to their characteristic to compress an image with or without information loss: lossless compression algorithms and lossy compression algorithms. A general overview of the algorithms, from the chronological point of view, is presented below by mentioning a few details about the ideas used in each algorithm.

In a first published article [82] (detailed in the author's Master Thesis [77]), we first introduced an algorithm containing most of the ingredients that we use in our compression schemes. That paper was not included in this thesis, but is a precursor of the seven publications [P1]-[P7] included in this thesis. The main idea of the algorithm is to design segmentations suitable for prediction, which are transmitted to the decoder using region contours by codifying each contour using chain-code representations. The regions are reconstructed using prediction, where the smallest details in each region are encoded by the prediction residuals.

In the first publication [P1] of this thesis, we further developed the concepts from [82]. We first improved the prediction inside each region by introducing a mixture of local predictors, and by searching on both directions, column-wise and row-wise, to find the variant where the codelength of the residual prediction errors is the smallest. Secondly, we improved the contour compression by introducing five options for encoding the contour segments, from which the best option is selected.

The second publication [P2] introduces a lossy compression algorithm. The main idea of the algorithm is to generate a set of segmentations. Each segmentation creates a lossy version of the initial image, where the distortion is controlled by a selected threshold. The algorithm starts from an over-segmentation, where the regions contain pixels having the same depth value. The segmentations are generated by merging neighboring regions, in an order determined by their cardinality.

Two regions are merged if the variation of the depth values inside the region does not exceed a threshold. In the final stage, each lossy image is compressed losslessly using prediction techniques, residual quantization, and chain-code representation.

Our main algorithm for lossless compression is dubbed Crack-Edge–Region–Value (CERV) and is presented in the third publication [P3]. CERV uses the initial partition into constant regions of the image, where each region contains pixels having the same depth value, and improves the compression of both region contours and depth values when compared with the state of the art algorithms. In the first stage, denoted *Algorithm C*, the algorithm collects distributions at template contexts, finds the optimal context tree, and then uses the tree in the compression of crack-edges (or contour edges), which are the 'atomic' elements used to represent the region contours. In the second stage, denoted *Algorithm Y*, the algorithm encodes the regions depth value using the list of depth values of the already encoded neighboring regions.

In the fourth publication [P4], we focus on developing an image segmentation algorithm, where the segmentations are designed for lossy compression. Each created lossy image corresponds to a certain distortion, and can be compressed using an entropy coder (e.g. CERV). Sequences of segmentations are obtained either by merging regions or by splitting regions. In the region merging process, we select the pair of regions to be merged as the pair that obtains, after the merging, the lossy image with the best estimated slope in a Rate-Distortion (RD) plot. In the region splitting process, a template is selected and the procedure is reversed, by splitting regions, and encoding more efficiently the contours using horizontal and vertical lines. The algorithm, denoted Greedy rate-distortion Slope Optimization (GSO), uses the piecewise constant model to reconstruct the regions.

In the fifth publication [P5], we studied the problem of finding the optimal solution for generating and encoding contour segments. A contour segment is described using an anchor point, a direction point, and a sequence of three-orthogonal (3OT) chain-code symbols. The problems tackled by the algorithm are the generation of the contour segments, the traversing of contour intersections, and the coding of the anchor points in such a way that the contour codelength is as small as possible. The algorithm was denoted Anchor Points Coding (APC), and is our one-dimensional coding solution of contours, having very similar results with CERV, our bi-dimensional coding solution.

In the sixth publication [P6], we focused on designing an algorithm that can provide the progressive coding of a sequence of images generated by the region merging phase of GSO, the algorithm being called Progressive coding of GSO sequences (P-GSO). In [P6], the goal was to develop a progressive coding algorithm that can obtain good results over a wide range of rates without paying a high price for the progressive functionality, and so that the performance is not degraded too much when compared with non-progressive methods.

Finally, in the seventh publication [P7], we studied the parameterization of the planar model for lossy image compression, and we choose the heights (in the optimal Least Squares (LS) plane) of three pixel positions, as parameters for the planar model. Seven methods are compared to find a way of choosing the three

optimal positions to improve the baseline results, where the constant model is used. The developed algorithm was denoted Plane Fitting (PF), and its results have further improved the constant model results because of the use of a more complex model, that introduces a lower distortion inside each region, and due to the efficiency of *Algorithm D*, which is used to compress the plane parameters. The GSO algorithm was extended to the GSOm with Plane Fitting (GSOmPF) algorithm by introducing a competition between the constant and planar models in the region merging decision so that better segmentations are generated for the lossless compressors.

Three other articles were published on related topics and are not included in this compilation of publications. In [88], the contours of fractal [28] and depth-map images are encoded using a combination of active horizontal and vertical line contours, that separate the current pixel from the northern and western neighboring pixel. In [86], CERV is used in a two-phase compression algorithm for histological images. In [83], a preliminary study of the contour intersections is presented, and the similarities of the optimal context trees are analyzed.

1.3 Outline of the thesis

The thesis is organized as follows. In Chapter 2 we describe the general concepts (common in most of our methods), the basic coding methods, and the statistical methods, i.e. the building blocks used in the development of our algorithms.

In Chapter 3 we describe the algorithms developed for lossless compression. We start by first analyzing the recent state of the art lossless compression algorithms and then discuss the algorithms developed for transmitting to decoder the image segmentation, by encoding its regions contours. Two ideas were used in our algorithms: the contour is encoded either by sequences of vertex positions codified by the Three-OrThogonal (3OT) representation [P1, P5], or by contour edges using template contexts [P3]. To reconstruct losslessly the image, our algorithms contain a second stage, where we developed methods for encoding the constant model parameters used by the region reconstruction procedure. Here, two ideas were tested: predictive coding using a mixture of predictors [P1], and the use of the list of depth values of the neighboring regions for 'guessing' the symbols [P3].

In Chapter 4 we describe the lossy compression of depth-map images. We start by analyzing the recent state of the art lossy compression algorithms. The algorithm from [P2] based on variability constrained segmentations is described first, and is followed by a detailed analysis for the GSO algorithm [P4] using region merging and region splitting. The progressive coding of GSO sequences [P6] is presented next, and is using all available a priori information. Finally, the parameterization of the planar model is studied using seven alternative methods in [P7].

In Chapter 5 we present the compression results of the proposed algorithms for the available datasets of depth-map images. In Chapter 6 we present the original contributions introduced by the seven publications selected for this compilation of articles and we draw the final conclusions.

Chapter 2

Basic Principles

"Secret agent 00111 is back at the Casino again, playing a game of a chance, while the fate of mankind hangs in the balance."
— Solomon Wolf Golomb, *Run-Length Encoding*

In this chapter, we describe the basic principles used in the development of our algorithms. In Section 2.1 we discuss about depth-map images. In Section 2.2 we describe the way we choose to represent the depth-map images and the sequences of symbols used to encode the image. The basic principles of entropy coding are mentioned in Section 2.3, while in Section 2.4 we introduce the statistical models used to encode different sequences of symbols generated by the image representation.

2.1 Depth-Map images

Figure 2.1 shows the pinhole camera model, where the point in space with the coordinates $(\mathbf{X}, \mathbf{Y}, \mathbf{D})$ is projected on the image plane at the coordinates $(\mathbf{x}, \mathbf{y})$. The matrix $D(x, y)$ represents the depth-map of the points in the scene, recorded at the integer coordinates $x = 1, 2, \ldots, n_r$, $y = 1, 2, \ldots, n_c$ in the image plane, where n_r and n_c are the number of rows and columns respectively.

When a depth sensor is used to acquire the depth-map image, the data are recorded in a matrix I, called *intensity image* [107], with the relation between I and D being:

$$D(x, y) = \frac{1}{\frac{I(x,y)}{Q_l}\left(\frac{1}{D_m} - \frac{1}{D_M}\right) + \frac{1}{D_M}}, \tag{2.1}$$

where D_M is the maximum depth and D_m is the minimum depth acquired by the sensor, and $Q_l = 2^b - 1$ is the maximum quantization level used. In our tests the intensity images are saved using $b = 8$ bits.

When the depth is estimated by *stereo matching* [71, 84] from two color images taken from two different viewpoints, the output of the stereo matching is the

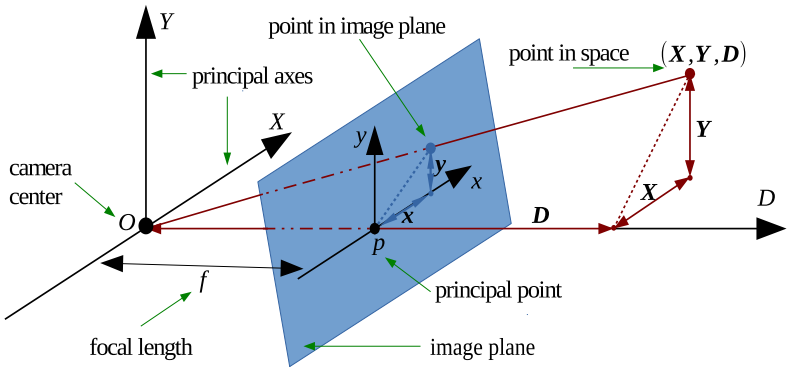


Figure 2.1: Pinhole camera geometry. The projection on the image plane of a point in space. $OXYD$ are the principal axes and O is the camera center. pxy are the image plane coordinates and p is the principal point. f is the camera's focal length. A point in space with the coordinates $(\mathbf{X}, \mathbf{Y}, \mathbf{D})$ is projected on the image plane at the coordinates $(\mathbf{x}, \mathbf{y})$.

disparity image B. Each disparity value $B(x, y)$ at position (x, y) is inversely proportional to the depth $D(x, y)$. An example of a pair of depth and color images, corresponding to one viewing position of the scene, is presented in Figure 2.2.

Irrespective of the methods for computing the depth, we use in this dissertation the generic matrix Z that we choose to call as *depth-map* image, where Z can be either the intensity image I or the disparity image B.

If a 3D view is synthesized using the color images of two views, the matrix Z is known as a *disparity* image [27, 91] and stores the distance using disparity values. The applications of depth-map images are covering many areas of computer vision where the point cloud represents the geometry of the observed scenes and further tasks can be, e.g., object detection and recognition.

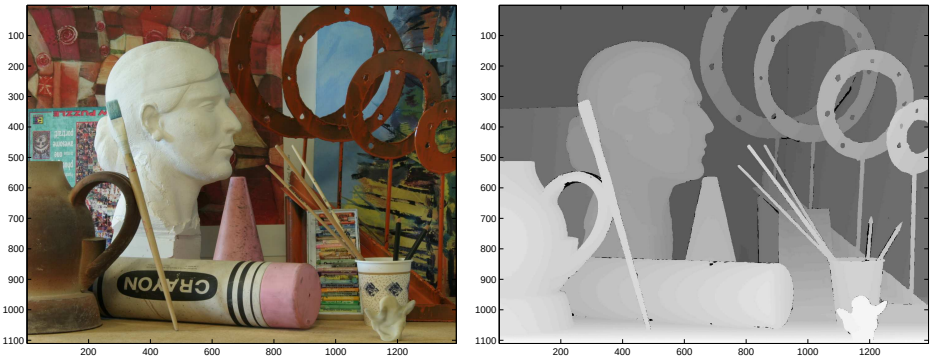
Additionally, the depth-map image can be used for view synthesis in the important application of 3DTV [34], where new views can be synthesized starting from the color images of two given views, and using the information from the depth-map image.

2.2 Image representation

In our approach, the input matrix Z is represented using two types of information:

- An image *segmentation*, that divides the image into regions.
- A set of *depth values*, used to reconstruct each region.

A region may contain one pixel or a collection of pixels, and each region may contain pixels having the same depth value or similar depth values, depending on the type of segmentation used to represent the image. Figure 2.3 shows all



(a) (b)

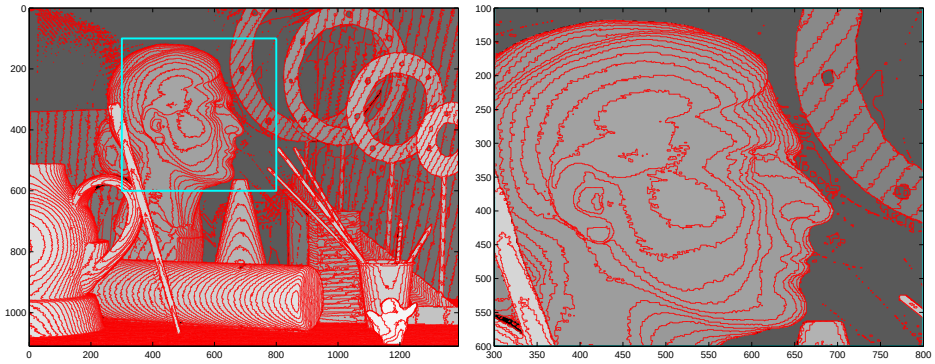
Figure 2.2: The pair of images corresponding to one viewing position of *Art* image (full-size, viewing point 'disparity1') from *Middlebury* dataset. The pair is composed of: **(a)** one color image, and **(b)** one depth-map image (disparity image).

the contours found in the initial image *Art*, where each region contains pixels having the same depth value. In our approach for lossy compression, we developed segmentation algorithms that are selecting a subset of contours out of the initial set of contours by either creating regions with certain proprieties [P1], or by ranking the contours [P4]. For lossless compression, all the contours found in the initial image (see Figure 2.3) are encoded.

The region pixels are connected in 4-connectivity, which means that a pixel position (x, y) has four neighbor pixel positions: $(x+1, y)$, $(x-1, y)$, $(x, y+1)$, and $(x, y-1)$. Let us denote $\Omega = \{(x_i, y_i)\}_{i=1,2,\ldots,m}$, a generic region of Z, containing m pixels, and let us denote d, the depth value used to represent Ω, which can be the depth value of every pixel in Ω or the average value of the pixels in Ω. If the matrix Z is divided into n_Ω regions, then the ℓ^{th} region of the segmentation is Ω_ℓ and is having m_ℓ pixels with depth d_ℓ.

2.2.1 Contour representations

A segmentation is represented using the regions contours. Let us denote as *contour map* the union of all *contour edges* (or *crack-edges*) that form the regions contours, where a 'contour edge' is the atomic element used to represent the regions contours. A contour edge is used to separate two neighboring pixels in the contour map. If the contour edge is active, then the two pixels are belonging to two different regions. If the contour edge is inactive, then the two pixels are belonging to the same region. One way of encoding the contour, represented using contour edges, is to encode all the contour edges, active and inactive, that can be found in the image (see Section 3.2.1). For a very simple segmentation, where only a few contour edges are set active, a large codelength is used to inform the decoder about the positions of the inactive contour edges.



(a) (b)

Figure 2.3: Example of the initial image partition, where each region contains pixels having the same depth value. The contours separating the regions are marked with red. **(a)** The image *Art* (full size, viewing point 'disparity1') from the *Middlebury* dataset; **(b)** The zoom in the cyan rectangular from (a).

Another strategy for encoding the contour is to create its one dimensional representation by first dividing it into sequences of contour edges, called here contour segments, and then to codify each contour segment as a sequence of symbols that describes the way it is 'drawn' on the contour map. To draw a contour edge, the decoder needs to know the positions of its two ends, which are called here *vertices*. Since the contour of the regions is continuous, in a sequence of edges, we need to encode the positions of both vertices of the first edge, and the position of one vertex for each of the rest of the edges. Hence, a contour segment, formed of $n-1$ neighbor edges, can be represented using a sequence of n adjacent vertices, e.g. $[P_1 \quad P_2 \quad \cdots \quad P_n]^T$. The sequence of vertices can be encoded using a chain-code representation, which informs the decoder how to draw the contour edges, i.e. by starting from a current vertex and continuing with one of its adjacent vertices, where the selection of the next adjacent vertex is codified by a chain-code symbol until the end of the contour segment.

Chain-code representations

Many types of chain-code representations were developed for different purposes [10, 11, 74]. In [87], five chain-code representations are studied while compressing the contours of binary images. If we choose an 8-connectivity for the pixels (i.e. a pixel has eight neighbor pixels), then the contour can be codified using one of the following chain-code representations:

- *Freeman 8 symbols (F8).* Each of the 8 adjacent vertices is codified by a symbol according to their position relative to a current vertex position. The set of symbols used is $\{0, 1, \ldots, 7\}$ and corresponds to a variation with a $\frac{\pi}{4}$ angle around the origin, which is the current position.

- *Differential Freeman 8 symbols (AF8).* Here, the next adjacent vertex in the sequence is codified by angular rotation, where the direction of movement is rotated with an angle of multiple $\frac{\pi}{4}$. The distribution of the symbols is closer to the exponential distribution.

If the 4-connectivity is selected to create a region (i.e. a pixel has four neighbors), then the contour can be codified using one of the following chain-code representations:

- *Freeman 4 symbols (F4).* It is similar to F8, but uses a set of four symbols $\{0, 1, 3, 4\}$, corresponding to a $\frac{\pi}{2}$ angle variation around the unit circle.

- *Differential Freeman 4 symbols (AF4).* It is similar to F4, but after moving from one vertex to an adjacent vertex, there are only three more adjacent vertices as possible options to move forward, and each option is encoded by a symbol in the set $\{0, 1, 2\}$.

- *Three-orthogonal (3OT).* The symbols in the representation are set using the position of the adjacent vertices relative to the current vertex, and using a long term memory of the movement when traversing a contour segment (see Figure 2.4). This representation was chosen in this dissertation and is described in more detail in the next subsection.

3OT chain-code representation

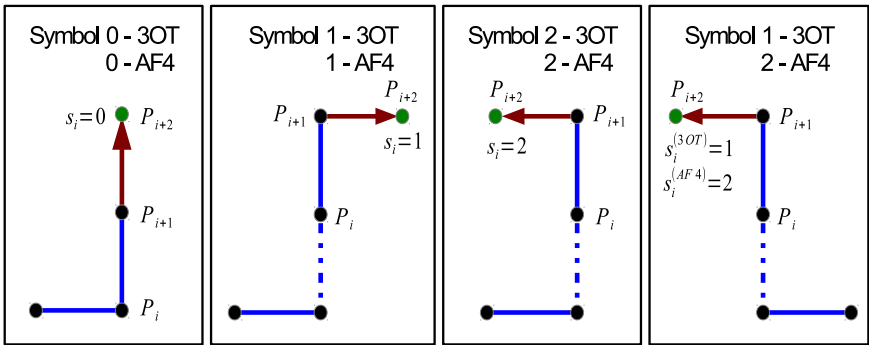
The 3OT representation is encoding the current vertex, say P_{i+2}, relatively to the previous two vertices in the sequence, P_i and P_{i+1}, using a symbol $s_i \in \{0, 1, 2\}$ (see Figure 2.4). 3OT codifies a sequence of vertices starting from the third vertex in the sequence, until the last vertex in the sequence. Hence, the first two vertices P_1 and P_2 must be coded using a different strategy, and the remaining sequence of vertices, $[P_3 \quad P_4 \quad \cdots \quad P_n]^T$, are codified by a corresponding vector $[s_1 \quad s_2 \quad \cdots \quad s_{n-2}]^T$, where s_i is a 3OT symbol.

The 3OT representation is describing the advance in the description of a contour segment, from one vertex to one of the three remaining adjacent vertices, using a symbol with one of the following meanings:

(0) A symbol *'0'* is describing the advance from the current vertex to the vertex found on the position for which the previous and next contour edges form a straight line. The symbol has also the significance of 'going forward'.

(1) A symbol *'1'* is describing the advance from the current vertex to the vertex found on the position for which the direction of movement is the same as previously and the orientation changes: *horizontal* $\leftrightarrow$ *vertical*.

(2) A symbol *'2'* is describing the advance from the current vertex to the vertex found on the position for which the direction of movement and the orientation are changing. The symbol has the significance of 'going back'.



(a) (b) (c) (d)

Figure 2.4: Examples of vertices codified by a 3OT or by a AF4 symbol. In the 3OT representation the unknown vertex positions, P_{i+2}, is codified using the last two known vertex positions, P_{i+1} and P_i, and the long term memory of the movement, where the memory is updated while traversing the contour segment. In the AF4 representation the unknown vertex positions, P_{i+2}, is codified only using the last two known vertex positions P_{i+1} and P_i. Known contour edges are marked with blue lines, and known vertices are marked with black dots. The arrow shows the next adjacent vertex to visit, P_{i+2}, which is marked with green dots. Previous movements done while traversing the contour are marked with dotted lines.

The AF4 and 3OT representations are compared in [83], where the contour of fractal and depth-map images are compressed. The results have shown that the 3OT representation is more suitable to represent the contours, because 3OT generates a sequence of symbols that has more redundancy, and that contain a distribution with a lot of symbols 0 and 1, and only a few symbols 2. Figures 2.4.(c,d) are showing how the use of the long term memory in the 3OT representation is decreasing the number of symbols 2 and increasing the number of symbols 1 compared to the AF4 representation.

2.2.2 Region reconstruction

The last stage of our compression scheme is the regions reconstruction stage. In the previous stage, the decoder received the information about the contour and used it to define the partition of the image into regions using the set of regions $\{\Omega_\ell\}_{\ell=1,2,\ldots,n_\Omega}$. To finish the representation of the image, the decoder is reconstructing each region in the image by setting a depth value to each pixel.

In this dissertation, we used two ways for encoding the information needed to reconstruct the regions. One way is to develop a predictor (see Section 2.4.3) and to encode the prediction error computed for each pixel in the region. The prediction techniques use the causal neighborhood to estimate the depth of each pixel, and therefore the segmentation plays an important role and it is designed

so that prediction errors are as small as possible. Another way to reconstruct the regions is to use a model to fill each region (e.g. constant or planar model). For the constant model, one depth value is encoded for each region and is set to each pixel. For the planar model (see Section 2.4.4), three parameters are encoded and are used together with the pixels coordinates to compute a plane and to set each pixel with a depth value.

2.3 Entropy coding

Entropy coding is an efficient way of compressing a sequence of symbols, taken from a known alphabet. In its simplest form (as in the optimal Huffman coding [33]), each symbol in the alphabet is mapped to a codeword by a reversible mapping so that the sequence is reconstructed losslessly from the encoded sequence of codewords.

Let us consider a sequence of symbols $\mathbf{x}^n = x_1 x_2 \dots x_n$, containing n elements, where each element x_j is a symbol in the finite ($k \ll n$) alphabet $\{s_i | i = 1, 2, \dots, k\}$ of k symbols and the counts of symbols of $\mathbf{x}^n$ is $D = [n_1 \ n_2 \ \dots \ n_k]^T$, where n_i is the frequency of symbol s_i. The empirical probability distribution from $\mathbf{x}^n$ is $P = [p_1 \ p_2 \ \dots \ p_k]^T$, where $p_i = \frac{n_i}{n}$ is the empirical probability of occurrence for symbol s_i. P contains the probability for each symbol in the alphabet to be the next element x_{n+1} in the sequence.

In 1948, Claude Elwood Shannon, who is known as "the father of information theory", published his famous paper *"A Mathematical Theory of Communication"* [85], where he introduced the *Shannon Lossless Coding Theorem*. His source coding theorem states that for a sequence of independent and identically distributed random variables, having the probability distribution P, the *entropy* H is the lowest bound on the average number of bits per symbol, with which the sequence can be compressed. The entropy is computed as

$$H = -\sum_{i=1}^{k} p_i \log_2 p_i. \tag{2.2}$$

The sequence $\mathbf{x}^n$, composed of n symbols, may be encoded losslessly using in average $n \cdot H$ bits.

2.3.1 Estimators of probability distribution

In data compression, the sequence $\mathbf{x}^n$ can be encoded by updating adaptively the probabilities of the symbols. In the initial stage, the same probability $p_i = \frac{1}{k}$ is associated to each symbol in the alphabet. The associated probability distribution is updated in the stage where element x_j is encoded, to be used in the next stage where the element x_j is known, by associating a new set of probabilities to each symbol in the alphabet. There are many types of probability estimators used in literature, which are usually classified according to the proprieties of the sequence.

The most used type of estimators is the *add-constant predictor family* [14], which is written as

$$p_i(x_{n+1} = s_i) = \frac{n_{s_i} + c}{n + kc},\ i = 1, 2, \ldots, k, \tag{2.3}$$

where $p_i(x_{n+1} = s_i)$ is the probability associated to the symbols s_i, i.e. the probability that the next symbol in the sequence is symbol s_i; n is the number of elements in the sequence that were already encoded; k is the length of the symbols alphabet; and c is a predictor's constant. When c takes some specific values, famous estimators are obtained:

(a) If $c = 1$, then the *add-one* estimator is obtained, known also as the Laplace (L) estimator [47], and (2.3) is rewritten as

$$p_i(x_{n+1} = s_i) = \frac{n_{s_i} + 1}{n + k},\ i = 1, 2, \ldots, k. \tag{2.4}$$

(b) If $c = \frac{1}{2}$, then the *add-half* estimator is obtained, known also as the Krichevsky-Trofimov (KT) estimator [42], and (2.3) is rewritten as

$$p_i(x_{n+1} = s_i) = \frac{n_{s_i} + \frac{1}{2}}{n + \frac{k}{2}},\ i = 1, 2, \ldots, k. \tag{2.5}$$

However, sometimes different values for the predictor's constant c offer better results, e.g. $c = 0.42$ was used in [51].

2.3.2 Arithmetic coding

Arithmetic coding [46, 97] is a form of entropy encoding, that can obtain a codelength close to the optimal codelength computed by the entropy using the probability distribution.

It encodes the entire string of data by creating a string of code that represents a fractional value found in the interval [0, 1). The algorithm is encoding one symbol at a time. It partitions at each iteration a smaller interval from the initial interval [0, 1), where each partition has the intervals proportional to the values in the current probability distribution. The interval corresponding to the current encoded symbol is retained as the new interval. Therefore, the algorithm is dealing with smaller intervals at each iteration, and the generated code string is selecting the encoded symbol in each of the nested intervals. The string of data is recovered by using the code string to partition and retain at each iteration the nested subinterval in a procedure that the encoder used to generate the code.

Let us consider now encoding the sequence $\mathbf{x}^n$ using the arithmetic coder. Let us suppose its alphabet is $\{s_1, s_2, \ldots, s_k\}$ and the Laplace (L) estimator is used to update the probabilities. Before anything is transmitted, every symbol has the same probability $p(s_1) = p(s_2) = \ldots = p(s_k) = \frac{1}{k}$ and the initial interval is portioned into k intervals: symbol s_1 has associate the first interval $[0,\ p(s_1))$,

symbol s_2 has associate the next interval $p(s_1)+[0,\ p(s_2))$, and so on until symbol s_k has associate the last interval $\sum_{i=1}^{k-1} p(s_i) + [0,\ p(s_k)) = [1 - p(s_k),\ 1)$. After receiving the first element in the sequence x_1, e.g. $x_1 = s_\ell$, the encoder uses the ideal codelength $\mathcal{L}(p(x_1)) = \mathcal{L}(p(s_\ell)) = -\log_2(p(s_\ell))$ $[bits]$ to inform the decoder that $x_1 = s_\ell$ and narrows the initial interval to the interval associated to the symbol s_ℓ by selecting $\sum_{i=1}^{\ell-1} p(s_i) + [0,\ p(s_\ell))$ as the new interval. The Laplace estimator (2.4) is used to update the probabilities of the symbols and the current interval is portioned again into k intervals using the new probability distribution. In the second iteration, the encoder uses $\mathcal{L}(p(x_2))$ bits to transmit to the decoder the element x_2, the interval is further narrowed to the interval associated with the symbol used to represent x_2, and the probability distribution is updated using the Laplace estimator. This procedure continues until the last element in the sequence is encoded using $\mathcal{L}(p(x_n))$ bits. It can be proven that, the sequence $\mathbf{x}^n$ is encoded using approximately the codelength

$$\mathcal{L}(P_L(\mathbf{x}^n)) = -\log_2\left(\frac{(k-1)!}{(n+k-1)!}\prod_{i=1}^{k} n_i!\right). \tag{2.6}$$

2.3.3 Golomb-Rice coding

A different strategy, for encoding the sequence of symbols $\mathbf{x}^n$, is to codify the symbols in the original alphabet $\{s_i\}_{i=1,2,\ldots,k}$ using a set of *codewords*, called a *code*. For each symbol (or string of symbols) the code associates usually a variable-length codeword that is formed of symbols 0 and 1.

The most used codes are the prefix codes, which have the propriety that, in the set of codewords, a codeword is never a prefix (initial segment) of any other codeword in the set. The Huffman code is an optimal prefix code that creates a set of codewords such that the average codelength is minimized. The Huffman algorithm was developed in 1952 by David A. Huffman [33]. However, if the alphabet is infinite, we cannot apply directly the Huffman algorithm.

When the infinite alphabet has a geometric distribution, the Golomb-Rice (GR) code [25, 69] is the optimal prefix code. Each integer symbol $s_i \equiv i$ is represented using the quotient's codeword and the remainder's codeword, when the symbol s_i is divided by a parameter M. For a fast implementation, the parameter M is selected as a power of 2, i.e. $M = 2^{k_{GR}}$. The quotient q_i and the remainder r_i are computed as $q_i = \lfloor \frac{s_i}{M} \rfloor$, and $r_i = s_i \% M$ (meaning r_i equals s_i mod M). The quotient's codeword is generated by unary coding, writing a q_i-length string of bits set as 0, followed by one bit 1, to mark the end of the string. The remainder's codeword is generated by writing r_i in the binary format. The codelength needed to encode $\mathbf{x}^n$ can be estimated using the parameter M. The GR algorithm first searches for the optimal parameter k^*_{GR} that obtains the minimum codelength. The first value encoded by GR is k^*_{GR}, and is followed by n pairs of quotient and remainder codewords. The decoder first obtains k^*_{GR}, computes $M = 2^{k^*_{GR}}$, and then uses M to decode the sequence $\mathbf{x}^n$.

2.4 Statistical models for prediction and coding

The entities used in the representation presented in Section 2.2 are each codified by a sequence of symbols. However, before encoding each sequence using the arithmetic coding, we are using a statistical model for achieving a good compression. A context model selects for each element x_j in the sequence $\mathbf{x}^n$ a context in which x_j is usually found. For one-dimensional signals, the context is created using a window of recently encoded symbols. For bi-dimensional signals (e.g. matrices), the coding is done line by line (or similarly column by column), and the context is created using the already encoded lines. In this case the context is usually called a *Template Context* [51] and it contains the elements located at the pixel positions selected, in order, according to the ℓ_1 or ℓ_2 norm computed between the current pixel position and the selected pixel position.

2.4.1 One-dimensional models

A one-dimensional model uses the previous N symbols in the sequence to create a set of contexts, where N is the order of the model. A zero-order model will then have $k^0 = 1$ contexts for which the probability distribution is computed. This model encodes the sequence $\mathbf{x}^n$ using a probability distribution computed by one of the estimators presented in Section 2.3.1. For an 1-order model, k^1 contexts are created (since there are k symbols in the alphabet), and in each one of them a probability estimator is updating the probability distributions. Hence, for an N-order model, k^N contexts are created.

A graphical representation of the contexts is a tree, called *context tree*, denoted by T. Each node of the tree represents a context for which a probability distribution is computed. In each node, there are k branches labeled with the k symbols in the alphabet. A context tree having the tree depth N is the graphical representation of all the models from the first order until the N^{th} order, since at every tree depth ℓ, between 1 and N, we have the contexts of the ℓ-order model.

Adaptive Markov Model

The Adaptive Markov Model (AMM) uses the k^N contexts of the N-order model, where k is the length of the alphabet. Its contexts are represented by all tree nodes at depth N, the reason why the adaptive Markov model is sometimes called the Fully Balanced Context Tree Model.

Let us consider that the N-order adaptive Markov Model is used to encode the sequence $\mathbf{x}^n$, and the probability distributions are computed using the Laplace estimator. The context of each element in the sequence is obtained using the previous N elements in the sequence. For the element x_j we obtain the context $C_j = x_{j-1}x_{j-2}\ldots x_{j-N}$. The probability distribution used to compress x_j is the one corresponding to context C_j, i.e. $p(x_j|C_j = x_{j-1}x_{j-2}\ldots x_{j-N})$. In the graphical representation, this corresponds to traversing of the context tree T from the root node to the leaf that represents the context C_j. The traversing is done by selecting, at each node of the tree, the branch labeled with the current symbol

in the context. For example, from the root node of the context tree T we select the branch labeled $s_i = x_{j-1}$, then in the new node we select the branch labeled $s_i = x_{j-2}$, until the last branch, labeled $s_i = x_{j-N}$, is selected.

Let us denote the distribution in the context C_j as $D_j = [n_1^j \; n_2^j \; \ldots \; n_k^j]^T$, where n_i^j is the frequency of symbol s_i, $j = 1, 2, \ldots, k^N$ is the context index, and $i = 1, 2, \ldots, k$ is the symbol index. We can then encode the sequence $\mathbf{x}^n$ using the N-order Adaptive Markov Model (N-AMM) with Laplace (L) estimator. Thus, eq. (2.6) is used for each context C_j, $j = 1, 2, \ldots, k^N$, and the model's codelength is computed as

$$\mathcal{L}(P_{AMM,L}(\mathbf{x}^n, N)) = -\sum_{j=1}^{k^N} \log_2 \left(\frac{(k-1)!}{\left(\sum_{i=1}^{k} n_i^j + k - 1\right)!} \prod_{i=1}^{k} n_i^j! \right). \tag{2.7}$$

Context Tree Models

The Context Tree Model (CTM) is a version of AMM where the context tree is pruned before it is used in coding. The coding procedure can be summarized by the following steps. Firstly, a context tree T having the maximum tree depth N is generated. Secondly, the codlength for each node in the tree is computed using the empirical counts of the node using (2.7). Thirdly, T is pruned and the optimal context tree denoted $\mathcal{T}$ is obtained. Finally, the sequence $\mathbf{x}^n$ is encoded using contexts with variable length, given by the optimal context tree $\mathcal{T}$. For CTM, the encoder is transmitting to the decoder also the shape of the optimal tree, e.g. by encoding a symbol '*1*' if the labeled branch of a node is cut, and a symbol '*0*' if it remains part of the tree.

There are many algorithms developed for tree pruning and for obtaining the optimal context tree [51, 70, 95]. In our algorithms, the pruning of the context tree T is done using the following rule: *A labeled branch is cut if the sum of the codelengths of the k child contexts, having context length $\ell + 1$, and the estimated codelength for encoding the k branches is greater than the codelength of the father context, having length ℓ.*

2.4.2 Bi-dimensional models

Template Context Models

A *Template Context* is a context used to encode the elements of a matrix. It is formed for a current pixel position by selecting certain positions from the current and previous lines and columns in the matrix. Figure 2.5 shows an example of a template context. The positions of the elements are labeled, in order, starting from the closest to the current pixel, going further away from it, and ending with the 18^{th} position that is having one of the highest ℓ_2 norm. The AMM with Laplace that uses the template context is called here Template Context Model (TCM).

If we are encoding the symbols in the matrix starting from the top left corner and advance line by line, the template context shown in Figure 2.5 can be used by TCM, with the model order set as the maximum position labeled in the context.

			18	14	17			
		12	10	6	9	11		
	16	8	4	2	3	7	15	
	13	5	1	X				

Figure 2.5: Example of a template context, where the order of the context elements is set using the ℓ_2 norm and the row-wise coding is used. The pixel which is currently encoded is marked with symbol X. The symbols $1, 2, \ldots, 18$ are marking the position of the 18 elements, which are forming the template context.

The order in which the context elements are selected can influence the results. The template context in Figure 2.5 can be used first as possible positions, and different orders can be tested before an order is chosen for the final fixed template.

Template Context Tree Models

The Template Context Tree Model (TCTM) is adding a pruning stage to TCM, where the template context tree is pruned. The contour coding algorithm from [P3], which is described in Section 3.2.2, uses this type of model. There, because a binary alphabet is used, the template context that has a particular mask can predict the value of the next element in the matrix without other auxiliary information. The main reason for this is that the region contours are continuous.

2.4.3 Predictive coding

One of the simplest ways to compress an image is to use a predictive method. The prediction techniques, also known as lossless Differential Pulse Coded Modulation (DPCM), are using a predictor to estimate the value of a current pixel based on neighboring pixel values. It is easy to guess that, in some given image region, a pixel may have a similar value with its neighbors. The image is usually scanned line by line, and for each pixel a predicted (estimated) value is computed using the pixels from the causal neighborhood of the current pixel. The prediction errors are then computed as the difference between the predictor's estimated values and the pixel's real values. In lossy compression, some part of the information is lost by applying quantization, while in lossless compression the prediction errors are encoded as they are, using a statistical model.

The improvements in predictive coding are usually done by first transmitting to the decoder the most important image edges, by encoding the segmentation and then applying prediction inside the regions. Because sometimes the segmentation is too expensive to be encoded, the image is divided into blocks of different sizes [37], or a quad-tree decomposition is used [13]. Another way of improving the results is by selecting a predictor from a set of given predictors. This strategy is used by Joint Photographic Experts Group (JPEG), where the set contains seven predictors [93].

One of the most used adaptive predictor is the *Martucci predictor* [52], known also as the Median Adaptive Predictor (MAP). In our case, the predictor is estimating the depth value $z = Z(x, y)$, of the current pixel position (x, y) in the image Z, using the depth values of the northern (N), western (W) and northwestern (NW) pixel positions: $(x-1, y)$, $(x, y-1)$, $(x-1, y-1)$, in the causal neighborhood of the current pixel. If we denote $z_N = Z(x-1, y)$, $z_W = Z(x, y-1)$, $z_{NW} = Z(x-1, y-1)$, then MAP estimates z as

$$\hat{z}_{MAP} = median\{z_N, z_W, z_N + z_W - z_{NW}\}, \tag{2.8}$$

and the prediction error can be computed as $\epsilon = z - \hat{z}_{MAP}$.

2.4.4 Planar model

A different strategy for encoding a depth-map image, is to find a segmentation that divides the image into regions, where each region may contain an object. The search to find an object based segmentation has a high algorithmic complexity. In the lossy compression case, the optimization function of the algorithm depends on the codelength needed to transmit the segmentation and the distortion introduced in each region, while in the lossless compression case, the optimization function depends on the codelength needed to transmit auxiliary information to reconstruct the region exactly as initially.

The region reconstruction stage of our algorithms is based on the (piecewise) constant model, but a parameterization of the planar model is studied in the last publication [P7]. For the constant model only one parameter is used to reconstruct each region, by setting each pixel in the region to be equal to the encoded parameter, which can represent either the depth of each pixel in the region or the average of the pixel's depth. In the planar model case, a region is estimated using a plane, which usually has the parametric form $z = ax + by + c$ and where the three parameters (a, b, c) are first estimated and then encoded.

Let us consider a region $\Omega_\ell = \{(x_i, y_i)\}_{i=1,2,\ldots,n}$, having n pixels and the pixel's depth values denoted $z_i = Z(x_i, y_i)$, $i = 1, 2, \ldots, n$. The optimal plane that offers the best estimation of the region is obtained by the Least Squares (LS) algorithm. The LS solution minimizes the sum of squared modeling errors $\sum_{i=1}^{n} \varepsilon_i^2$, where the errors ε_i are defined by

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ & \vdots & \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}. \tag{2.9}$$

Hence, the LS parameters, $\underline{\theta}^* = \begin{bmatrix} a^* & b^* & c^* \end{bmatrix}^T$, are obtained. At the decoder, $\underline{\theta}^*$ is used to compute the floating point values z_i^* as:

$$z_i^* = \begin{bmatrix} x_i & y_i & 1 \end{bmatrix} \underline{\theta}^* = z_i - \varepsilon_i^*, \; i = 1, 2, \ldots, n. \tag{2.10}$$

where ε_i^* are the optimal modeling errors. The initial region can be then reconstructed as follows: at the pixel location (x_i, y_i), the value $\hat{z}_i$ is computed by rounding z_i^* as

$$\hat{z}_i = \lfloor z_i^* \rceil = z_i^* - \Delta_i^*,\ i = 1, 2, \ldots, n, \tag{2.11}$$

where $\Delta_i^* \in [-0.5,\ 0.5]$ are the rounding errors. From (2.10) and (2.11) results that the value z_i is reconstructed as $\hat{z}_i = z_i - (\varepsilon_i^* + \Delta_i^*)$.

In the lossless case, besides the plane parameters $\underline{\theta}^*$, the difference

$$z_i - \hat{z}_i = \varepsilon_i^* + \Delta_i^*,\ i = 1, 2, \ldots, n, \tag{2.12}$$

must be encoded for a perfect reconstruction.

In the lossy case, the planar model is introducing a distortion in the region Ω_ℓ by encoding only the optimal plane parameters $\underline{\theta}^*$. The distortion can be measured by Mean Squared Error (MSE), which is computed as

$$MSE_p^{(LS)}(\Omega_\ell) = \frac{1}{n}\sum_{i=1}^{n}(z_i - \hat{z}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(\varepsilon_i^* + \Delta_i^*)^2. \tag{2.13}$$

The distortion introduced in the region Ω_ℓ by the constant model with the parameter $d_\ell = \frac{1}{n}\sum_{i=1}^{n} z_i$ can be computed as

$$MSE_c(\Omega_\ell) = \frac{1}{n}\sum_{i=1}^{n}(z_i - \lfloor d_\ell \rceil)^2. \tag{2.14}$$

Chapter 3

Lossless Compression of Depth-Map Images

"The cure for boredom is curiosity.
There is no cure for curiosity."
— unknown author

The chapter starts by briefly presenting, in the first section, the state of the art coders in the research field of lossless compression of depth-map images. Three methods [P1, P3, P5] were developed for this topic and they contain two stages: contour compression and region reconstruction. Two contour compression algorithms are introduced in the second section and two region reconstruction algorithms in the third section. In the last section, we summarize the coders by selecting an algorithm from each stage.

3.1 State of the art coders

In the lossless compression field, there are several algorithms that were designed for depth-map image compressing. In one approach, instead of encoding (conditionally) each pixel's depth value, the authors are applying different transforms to the binary representation of the depth values. The integers in the depth-map image are then represented and encoded as a sequence of bit-planes. The large constant patches (with the same symbol) will also be present in the bit-planes. The transforms intend to lead to bit-planes with even larger constant patches. In the last stage of the methods, binary masks or entire bit-planes are encoded as binary images by a specialized entropy coder. For example, in one of the methods the authors first use the Gray code [26] (also known as reflected binary code) to do a transformation of the initial bit-plane representation of the image. In [39], the image is divided into blocks and the depth value of each pixel is converted to a gray code. The algorithm checks if the blocks of size 16×16 are full of symbols '0' or symbols '1', and encodes them using a binary switch. Since not all the blocks

are filled with only one symbol, the masks of the remaining blocks are encoded using the MPEG-4 Part 2 [15], known as the Visual Binary Shape coding scheme of the Moving Picture Experts Group (MPEG) [63]. The same idea of transforming the bit-planes is used also in [101]. This time the converted binary planes are encoded by the Joint Bi-level Image Experts Group (JBIG) standard for bi-level images. Moreover, the algorithm is extended to conditionally encode the left-right pairs of depth-map images from two viewpoints of the scene. The use of the JBIG standard offers to this method an important advantage compared to the previous ones. This group of methods is easy to implement and relies on the performance of the chosen entropy coder, but they were outperformed by various methods that exploited the specific characteristics of the depth-map image.

The algorithm in [12] is called Piecewise-Constant image model (PWC). The PWC algorithm was designed for the compression of palette images and was obtained by further developing the contour coding algorithm from [90]. In publication [P3], the experimental results show that PWC is a good solution for compressing depth-map images. Another palette coding method can be found in [68]. The reason why a method designed for palette image compression has good results in depth-map compression is that it is using a context coding algorithm which is able to detect and encode very efficiently the object boundaries and smooth areas inside the depth-map image.

Recently, many published algorithms use the idea of modifying some of the traditional lossless image coders to take advantage of the proprieties that depth-map images have. The H.264/AVC [9] and MPEG-4 Part 10, Advanced Video Coding (MPEG-4 AVC) are the video compression standards that are commonly used for video compression. For example, in [29] and [30], the authors modified the H.264/AVC standard to improve the results for depth-map compression. However, H.264 is mainly used in the lossy compression of video depth-map sequences.

The generic lossless image coders may also be used for compressing depth-map images. JPEG-LS is the JPEG standard for lossless and near-lossless compression of continuous-tone images and its core algorithm is called LOssless COmpression for Images (LOCO-I) [93, 94]. The algorithm consists of two main stages that are called modeling and encoding. Prediction, residual modeling and residual context-based coding are the main concepts used by the algorithm. LOCO-I achieves low complexity using the assumption that the computed prediction residuals are following *a two-sided geometric distribution*, and from the use of the *Golomb-like codes* in the coding stage, shown to be an optimal solution for coding geometric distributions. The programs are publicly available and their executable files can be downloaded from [45]. The advantages that LOCO-I offers are: it is easy to use and it is available online; has a low complexity and a small runtime. However, its compression performance can be easily outperformed by using more complex methods.

The Context-based Adaptive Lossless Image Coder (CALIC) [99] is one of the best generic lossless image coder that obtains high lossless compression for continuous-tone images. Modeling contexts are used by CALIC to condition the residuals of a non-linear predictor and to make the predictor able to adapt to

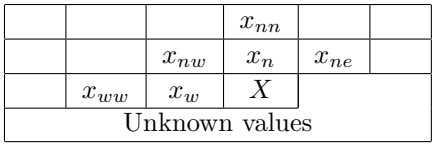


Figure 3.1: The causal neighborhood used by the Gradient Adjusted Predictor from CALIC. The current pixel position, marked by X, is predicted using the value of six neighboring pixels, marked by x_n, x_w, x_{nw}, x_{ne}, x_{nn}, x_{ww}.

different source statistics. The non-linear predictor is called Gradient Adjusted Predictor (GAP) and uses a causal neighborhood of six neighboring pixels (see Figure 3.1) to detect three types of edges (sharp, normal and weak) on both horizontal and vertical directions. In the adaptation process, the algorithm estimates the expectation of the prediction residuals conditioned on a large number of contexts rather than estimating a large number of conditional error distributions. CALIC is achieving a low time and space complexities thanks to the efficient techniques for forming and quantizing modeling contexts. The CALIC executable files are available online [98]. The large number of modeling contexts and the efficient coding of the predicted residuals makes possible to detect and encode the sharp edges and the smooth areas of the depth-map image. The small runtime and the very good compression performance are making CALIC one of the best options for depth-map image compression.

In our test we used as state of the art encoders the PWC, LOCO-I and CALIC algorithms. Although LOCO-I is the fastest, PWC and CALIC have the best compression performance, where PWC usually has a small advantage. In the following two sections we describe a set of algorithms for each of the two stages of our approach: contour compression stage and region reconstruction stage. In the final section we propose different encoders by selecting an algorithm from the set of each stage.

3.2 Algorithms for contour compression

The segmentation of a depth-map image is represented in our algorithms using the *contour map*, which separates the regions from each other. In this dissertation, we use a graph to store the contour map. The graph is having the vertices placed in a $(n_r+1)\times(n_c+1)$ *contour grid*, one size bigger than the image grid of size $n_r \times n_c$ of the depth-map image Z. A graph *vertex* is denoted by $P=(x,y)$, where (x,y) are contour grid coordinates. The smallest element used to represent the contour map is called *contour edge* and it separates two neighbor pixels belonging to two different regions. A contour edge is represented in the graph using a graph edge. The graph is created using the following rule that sets the contour grid coordinates using the image grid coordinates. If for two neighbor pixels $Z(x,y)$ and $Z(x,y+1)$ (located on the same row x in the image grid) we have $Z(x,y) \neq Z(x,y+1)$,

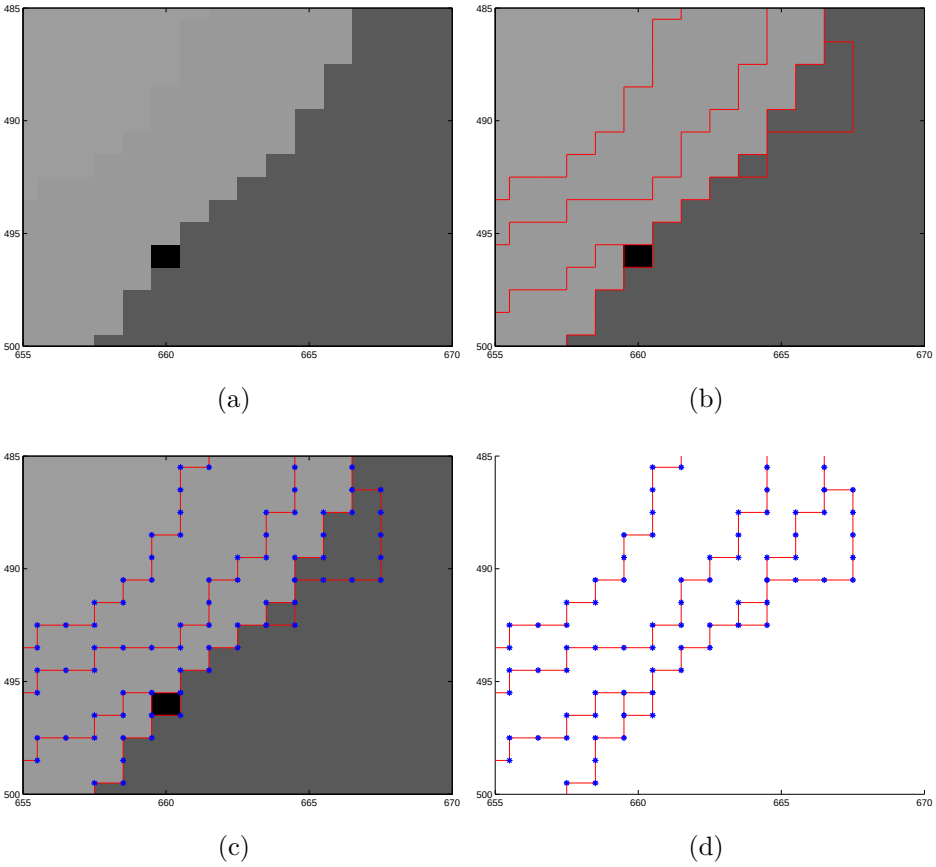


Figure 3.2: Representation of the contour of a depth-map image. The graph edges (or contour edges) are marked by red lines. The graph vertices are marked with blue asterisks. **(a)** The zoom in the depth-map image *Art* (full size, viewing point 'disparity1') from the *Middlebury* dataset. **(b)** The contour map of the image from (a) overlaid on the image. **(c)** The graph edges and vertices that form the contour map from (b). **(d)** The graph used to store the contour map from (b).

then we draw a graph edge (as the active contour edge) between the vertices $(x, y+1)$ and $(x+1, y+1)$ in the contour grid. Similarly, if for two neighbor pixels $Z(x, y)$ and $Z(x+1, y)$ (located on the same column y in the image grid) we have $Z(x, y) \neq Z(x+1, y)$, then we draw a graph edge (as the active contour edge) between the vertices $(x+1, y)$ and $(x+1, y+1)$ in the contour grid. Figure 3.2 shows the representation of the contour of a depth-map image. In Figure 3.2.(a) we present the initial depth-map image. The contour map of the image is presented in Figure 3.2.(b), and is represented in Figure 3.2.(c) using graph edges and graph vertices. The graph used to store the contour map is presented in Figure 3.2.(d).

There are two equivalent ways of encoding the segmentation of the image: by encoding the positions of the contour edges or by encoding the positions of the

vertices. The encoding of the contour edges requires the coding of a matrix of size $(n_r+1)\times(n_c+1)$, where the (active) contour edges that separates two regions are signaled using one symbol (e.g. '1') and the (inactive) contour edges that separates the pixels in each region are signaled using another symbol (e.g. '0'). If the contour edges are further represented using their ends, the vertices, the contour map can be encoded by transmitting to the decoder the vertex positions. Sequences of adjacent vertices are obtained by traversing the contour and collecting the contour edges between any two consecutive vertex positions.

3.2.1 Encoding vertex positions

In publications [P1] and [P5], we choose to encode the contour using the vertex representation. The first step in the algorithm is to 'divide' the contour map into contour segments. Let us consider that the contour map of Z is 'drawn' using a set of n_Γ contour segments, $\{\Gamma_k\}_{k=1,2,\ldots,n_\Gamma}$. Each contour segment, Γ_k, is represented by a sequence of n_{Γ_k} adjacent vertices, that are saved in the vector $\Gamma_k = [P_1\ P_2\ \cdots\ P_{n_{\Gamma_k}}]^T$. A contour segment is generated by first selecting the vertex P_1, called *anchor point*, and by traversing the contour map and saving the found adjacent vertex position in Γ_k.

Here, we choose to use the 3OT representation to codify Γ_k. However, the 3OT representation is encoding the position of one vertex relatively to the position of the previous two vertices in the sequence (see Section 2.2.1), and hence the last $n_{\Gamma_k}-2$ vertices from Γ_k, i.e. $[P_3\ P_4\ \cdots\ P_{n_{\Gamma_k}}]^T$, are codified by a vector of 3OT symbols, $S_k = [s_1\ s_2\ \cdots\ s_{n_{\Gamma_k}-2}]^T$. In each contour segment the first two vertices, P_1 and P_2, must be encoded using a different strategy. The vertex P_2 is called *direction point*, and is encoded relative to the position of the vertex P_1, the anchor point. However, when encoding P_1 there is no other a priori information that can be used to reduce the encoded codelentgh. This is the reason why the anchor points are the most 'expensive' information that needs to be transmitted to the decoder, i.e. they require a high codelength for coding. The number of anchor points can be reduced using a strategy that is selecting a minimum number of contour segments based on a contour generation procedure, and by setting a set of rules for traversing the contour map intersections.

Finally, the contour map that will be 'drawn' using the set of n_Γ contour segments, $\{\Gamma_k\}_{k=1,2,\ldots,n_\Gamma}$, is codified by n_Γ anchor points, n_Γ direction points, and a vector of 3OT symbols, denoted S, that is created by concatenating all the 3OT vectors, $S = [S_1^T\ S_2^T\ \cdots\ S_{n_\Gamma}^T]^T$. In each of the two publications, [P1] and [P5], a different strategy was used to reduce the codelength of the image contours. In the first strategy [P1], the codelength of vector S is reduced, since it was usually found to represent between 70% and 80% of the total output bitstream. That is why in [P1] we choose to select adaptively the statistical model for vector S and pick the one having the smallest codelength. In the second strategy [P5], we are searching for the optimal selection of the anchor points and contour segments, by analyzing contour crossing points and imposing rules to help us reduce the number of anchor points. The two strategies are presented in detail next.

Adaptive Selection of Statistical Models

The algorithm presented in [P1] has the following characteristics. The anchor points are found using a column-wise search inside the contour map (see Figure 3.2), and are saved in a matrix Υ by setting for each anchor point $P_1 = (x, y)$ a value $\Upsilon(x, y) = 1$. The matrix Υ was initialized with values '0' and is encoded using AMM with Laplace estimator. Each direction point, P_2, is codified by the F4 representation (see Section 2.2.1) using its relative position to the anchor point P_1. All direction points are collected in a vector that is encoded using the AMM with Laplace estimator.

The vector of concatenated 3OT symbols is encoded using the best statistical model selected adaptively. In [P1], we choose to use also two additional representations for S. In one representation, we codify S by using two vectors $S_{0,x}$ and $S_{1,2}$ that are each having binary symbols where: $S_{0,x}$ codifies the position of the 3OT symbol '0' using the symbol '0', and the position of both 3OT symbols '1' and '2' using the symbol '1'; $S_{1,2}$ codifies the distinction between the 3OT symbols '1' and '2' for each symbol '1' in $S_{0,x}$. The second representation is based on the classification of contour segments. Some contour segments are forming a contour loop because, in some cases, a region has only one neighboring region, i.e. is inside another region. The contour vectors obtained for this type of contours are concatenated in a vector S_1, while the others are concatenated in a vector S_2.

Hence, vector S is encoded by choosing the best option between the following:

(a) apply the AMM with Laplace estimator to S;
(b) apply the CTM with Laplace estimator to S;
(c) apply the AMM with Laplace estimator to S_1 and S_2;
(d) apply the CTM with Laplace estimator to $S_{0,x}$ and $S_{1,2}$;
(e) apply the AMM with Laplace estimator to $S_{0,x}$ and $S_{1,2}$.

Codelength estimation for each of the five options is used in model selection.

The Anchor Point Coding algorithm

In publication [P5], we improved the coding of the contours in a different way. The main idea of the algorithm is to analyze the contour crossing points and to decide which vertices are possible anchor points and which are not possible anchor points. Using this analysis, we can reduce the number of possible anchor points by imposing a set of rules when traversing the contour by checking whether a vertex is a possible anchor point or not.

A vertex used to draw the contour can have a maximum of four neighboring vertices, resulting in four types of vertices having the degree one, two, three or four, which are analyzed below.

(degree one) A vertex that has the degree one is denoted P_k^1, and has one adjacent vertex. This special type of vertex can be found only on the boundary of the contour graph, i.e. $P_k^1 \in \{(1, j), (n_r + 1, j), (i, 1), (i, n_c + 1)\}$. Because it has only one adjacent vertex, then a contour segment Γ_k can either start from P_k^1, or end with P_k^1. If the contour segment starts from P_k^1, then P_k^1 is its anchor point and its adjacent vertex is the direction point, P_2. Note that the position of

P_2 is found by the decoder without transmitting any additional information, since there is only one option to continue the description of the contour segment. Let us denote this found type of anchor point as *edge anchor point*, since it can only be found on the image outer edges.

(degree two) A vertex that has the degree two is denoted P_k^2, and has two adjacent vertices. It is easy to imagine that when we are traversing the contour and a P_k^2 vertex is reached, then one of its adjacent vertices was already visited and is the previous vertex in the contour segment Γ_k. Hence, there is one option to continue the description of the contour segment, i.e. by choosing the remaining adjacent vertex as the next vertex in the contour segment. However, if we are forced to choose P_k^2 as an anchor point, then the second vertex in the contour segment can be any of its two adjacent vertices. Let us denote this found type of anchor point as *double anchor point*. To be able to find an anchor point, we need first to choose a scanning method. The simplest way in which we can find anchor points is to search in each column for a vertex that still has adjacent vertices. Returning to our vertex type analysis, if a vertex $P_k^2 = (i, j)$ is selected as an anchor point, then the positions $(i, j + 1)$ and $(i + 1, j)$ are always unvisited because of the way we are searching for the anchor points, i.e. column-wise. Any of the two adjacent vertices can be selected as direction point, and hence we impose the rule that $P_2 = (i, j + 1)$ is always selected. Note that some contour segments are forming a loop, i.e. $P_1 = P_{n_{\Gamma_k}}$, and the second adjacent vertex was visited within the current contour segment. Otherwise, if the contour segment is not forming a loop, i.e. $P_1 \neq P_{n_{\Gamma_k}}$, then P_k^2 is the anchor point of the next contour segment Γ_{k+1} and the vertex $P_2 = (i + 1, j)$ is selected as direction point. This is the reason why these anchor points are called 'double'. The position of the vertex P_2 is again found by the decoder without transmitting any other information.

(degree three) A vertex that has the degree three is denoted P_k^3, and has three adjacent vertices. When a P_k^3 vertex is reached while traversing the contour map, the vertex may have one of two types, depending on whether it was previously visited or not. If it is the second time when the P_k^3 vertex is visited while traversing the contour map (note that in this case P_k^3 is not an anchor point), then it is the last vertex in the contour segment. However, if it is the first time the P_k^3 vertex is visited while traversing the contour map, then it has two unvisited adjacent vertices. If the contour segment would end here, then the procedure of segment generation would favor the generation of short contour segments. Since we already experimentally confirmed that for each contour segment an anchor point is encoded with a high codelength, one of the adjacent vertices must be selected to continue the description of the contour segment. Here we impose the rule: if one of the two unvisited adjacent vertices is codified by a 3OT symbol $s_i = 0$, then we use the Directive T1 to select the next adjacent vertex to visit, else we use the Directive T2 to select the next adjacent vertex to visit. The Directive T1 is introduced to differentiate a P_k^3 vertex from a P_k^4 vertex (case described in the next paragraph). Hence, a P_k^3 vertex is always codified by a 3OT symbol $s_i \in \{1, 2\}$, which offers the possibility to find an anchor point position for the next contour segment between the vertices of the previously encoded contour segment. (Note

Directive T1. *When a P_k^3 vertex is reached, the next adjacent vertex is selected as the one that is codified by a 3OT symbol $s_i \neq 0$.*

Directive T2. *When a vertex $P_k^3 = (i, j)$ is reached, if $(i-1, j)$ and $(i+1, j)$ are unvisited adjacent vertices, then select the next adjacent vertex as $(i+1, j)$ (if $(i, j-1)$ and $(i, j+1)$ are unvisited adjacent vertices, then select the next adjacent vertex as $(i, j+1)$).*

Directive T3. *When a P_k^4 vertex is reached, the next vertex is selected as the one that is codified by a 3OT symbol $s_i = 0$.*

Directive T4. *When a P_k vertex with already three visited adjacent vertices is reached, the remaining vertex, P_j, is adjacent if a 3OT symbol 0 codifies P_j. If so, the decoder knows that $P_k = P_k^4$ and goes visiting P_j (without encoding the 3OT symbol 0), 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} cannot be a possible anchor point.*

Directive C2. *When P_{k+1} is reached and the previous vertex in the sequence is $P_{k-1} = (i, j)$, 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} cannot be a possible anchor point.*

Figure 3.3: The set of APC directives from [P5]. The Directives T1-T4 are used for generating contour segments by traversing from a vertex to one of its selected unvisited adjacent vertex. The Directives C1-C2 are used to check if a vertex is not a possible anchor point position.

that the 3OT symbols $s_i = 1$ and $s_i = 2$ have the lowest frequencies among the three symbols.) Let us denote this found type of anchor point as *relative anchor point*, since its position is determined relative to a contour segment. The Directive T2 is introduced to eliminate the possibility for a P_k^3 vertex to be found as anchor point by the column-wise search. Figure 3.4.(e-p) shows all possible ways of selecting the next adjacent vertex to visit for a vertex P_k^3 having degree three, while Figure 3.4.(a-d) shows the impossible cases.

(degree four) A vertex that has the degree four is denoted P_k^4, and has all four vertices adjacent. A P_k^4 vertex is the crossing point of two contour segments, and it would be inefficient to select it as an anchor point. Therefore, to visit all its adjacent vertices the P_k^4 vertex is traversed two times. At the first visit, P_k^4 has three remaining adjacent vertices unvisited, i.e. three options to choose from, and the Directive T3 is used to select one of them. At the second visit, the decoder can detect the position of the last remaining adjacent vertex without any other information, since it is codified by a 3OT symbol '0'. Note that the Directives T1 and T4 are used to distinguish a P_k^4 vertex from a P_k^3 vertex.

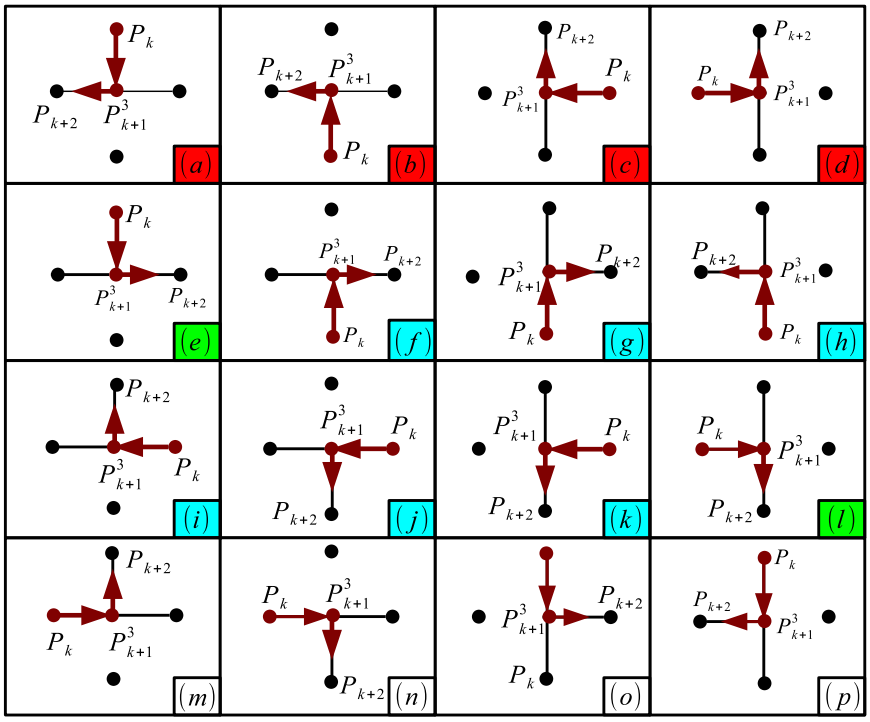


Figure 3.4: All possible ways of selecting the next adjacent vertex to visit for a P_k^3 vertex. The arrows show the traversing of the contour and the way a contour segment is generated, a red (black) dot is a visited (unvisited) vertex, and a red (black) line is a visited (unvisited) contour edge. The sub-figure color labels are marking with: (red) the impossible cases due to the Directive T2; (blue) the cases found by Directive C1, where P_{k+1} is not an anchor point; (green) the case where P_{k+1} is not an anchor point; (white) the cases where P_{k+1} may be an anchor point.

The above analysis establishes that the edge and double anchor points are found by a column-wise search, while the relative anchor points are found by checking, using Directives C1 and C2, if each traversed vertex is or not an anchor point. The algorithm creates a list, denoted Ψ, of vertices that were encoded by a symbol $s_i \neq 0$, for which the hypotheses of Directives C are not satisfied (see Figure 3.3).

The Directive C1 was introduced to be used in the search of anchor points, among the P_k^3 vertices, by removing from Ψ the vertices found in the cases shown by Figures 3.4.(f-k). The list Ψ is further shortened by the use of Directives T1 and T2, the only possible remaining cases are shown by Figures 3.4.(m-p).

The Directive C2 was introduced to search for anchor points in the cases where a 3OT symbol '0' is generated. We notice that a vertex $P_{k+1} = (i, j)$ cannot be an anchor point in the following cases: **(i)** in Figure 3.5.(a) the vertices $P_{k+2} =$

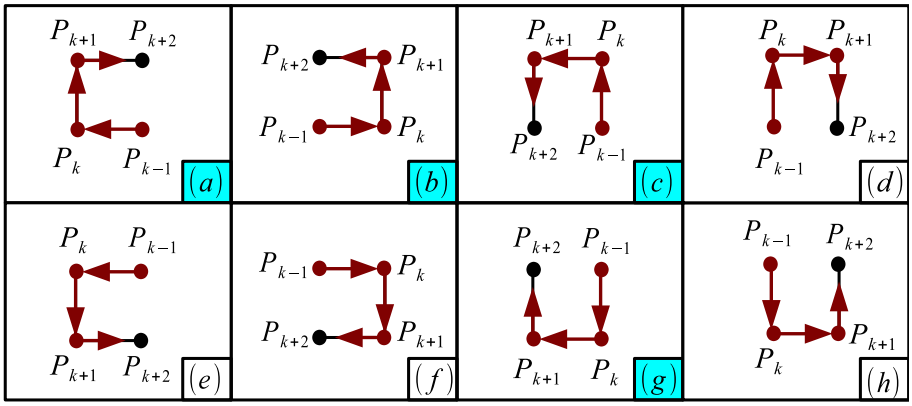


Figure 3.5: Cases where a 3OT symbol '2' encodes P_{k+2}. The arrows show the traversing of the contour, a red (black) dot is a visited (unvisited) vertex, and a red (black) line is a visited (unvisited) contour edge. The sub-figure color labels are marking with: (blue) the cases found by the Directive C2, where P_{k+1} is not an anchor point; (white) the cases where P_{k+1} may be an anchor point.

$(i, j+1)$ and $P_k = (i+1, j)$ (and in Figure 3.5.(c) the vertices $P_k = (i, j+1)$ and $P_{k+2} = (i+1, j)$) were visited when we first traversed P_{k+1}; **(ii)** in Figure 3.5.(b) P_{k+1} may only be the corresponding P^3_{k+1} vertex found in Figures 3.4.(b, h), cases that are either impossible cases, or cases where P_{k+1} cannot be an anchor point; **(iii)** similarly, in Figure 3.5.(g) P_{k+1} may only be the corresponding P^3_{k+1} vertex found in Figures 3.4.(c, i), cases for which the Directive C2 hypothesis are not satisfied.

The anchor points are stored by two arrays (initially full of zeros): Υ, a matrix of anchor points of size $n_r \times n_c$, and Φ, a vector of flags that are selecting the relative anchor points in Ψ. The edge and double anchor points are stored in Υ by setting $\Upsilon(P^1_k) = 1$, and respectively $\Upsilon(P^2_k) = 1$. The relative anchor points, P^3_k, are stored in Ψ at an incremented index ℓ. When the hypotheses of both C1 and C2 directives are not satisfied, then if P^3_{k+1} is a relative anchor point, then we set $\Phi(\ell) = 1$ (else the element $\Phi(\ell)$ remains set 0). These anchor points are found using the internal list Ψ, and they 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. $\Upsilon(P_k) \neq 1$).

The last step in the contour coding algorithm is the *entropy coding* step. The image contour is now represented only using three entities: S, Φ, and Υ, each entity being encoded by applying the CTM with Laplace estimator. In Algorithm 3.1 we present a summary of the Anchor Points Coding (APC) algorithm that is using the analysis presented above. Some specific implementation details regarding the entropy coding step can be found in Appendix A.1.

Algorithm 3.1: **Anchor Points Coding (APC)**

The algorithm encodes the contour of an image, where the contours are represented using a contour graph and the position of vertices in the contour graph.

(1) Search column-wise in the contour graph for a vertex with at least one unvisited adjacent vertex (an anchor point). Mark the found vertex $P_1 = (i, j)$ as $\Upsilon(i, j) = 1$.

(2) If P_1 is an edge anchor point, select its only adjacent vertex as the direction point. If $P_1 = (i, j)$ is a double anchor point, then select $P_2 = (i, j+1)$ as direction point, and if P_1 is selected again as anchor point, then select $(i+1, j)$ as direction point.

(3) Generate the current contour segment, Γ_k, by either selecting the only unvisited adjacent vertex as the next vertex to visit, or by selecting one of the unvisited adjacent vertices using Directives T1-T4 as the next vertex to visit.

(4) Check each vertex from the vector Γ_k, of vertices that draws the current contour segment, if it is a possible anchor point using the Directives C1 and C2. Save the position of the found presumptive anchor points in the list Ψ of possible relative anchor points, and mark in Υ, using the symbol '2', the remaining vertices in Γ_k.

(5) While the current index ℓ points to an incremented location in Ψ:

  (5.1) If $\Psi(\ell)$ is an anchor point, then set $\Phi(\ell) = 1$, else set $\Phi(\ell) = 0$.

  (5.2) Determine the position of the direction point from the way the anchor point was traversed the first time using Directives T1-T2.

  (5.2) Continue generating the current contour segment Γ_k as in step (4).

(6) Continue with step (1) until no more anchor points are found.

(7) Codify using the 3OT representation each Γ_k (starting from 3^{rd} position), and generate a vector S_k. Concatenate all the vectors S_k into $S = [S_1^T \; S_2^T \; \cdots \; S_{n_\Gamma}^T]^T$.

(8) Encode each entity S, Φ, and Υ (in this order) using CTM with Laplace estimator.

3.2.2 Encoding contour edges

In publication [P3], we introduce a different strategy for compressing the contours, where we choose to represent the contour using contour edges (called in [P3] crack-edges). The contour edges (crack-edges) are stored using two binary matrices, each of size $n_r \times n_c$: one matrix to store the vertical contour edges, denoted V; and one matrix to store the horizontal contour edges, denoted H. Using these two matrices we are able to achieve a much simpler connection between the image grid and the contour edge grid. In each location of the vertical binary matrix a vertical edge is stored, that is set as follows: if the pixel positions $(x, y-1)$ and (x, y), in the initial image Z, are belonging to two different regions, i.e. if $Z(x, y-1) \neq Z(x, y)$, then the vertical edge is set as active by $V(x, y) = 1$, else it is set as inactive by $V(x, y) = 0$. Similarly, in each location of the horizontal binary matrix a horizontal edge is stored, where $H(x, y) = 1$ if $Z(x-1, y) \neq Z(x, y)$ and $H(x, y) = 0$ otherwise. Figure 3.6 shows an example of the way active contour edges are set in the two binary matrices.

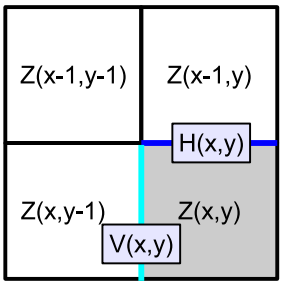
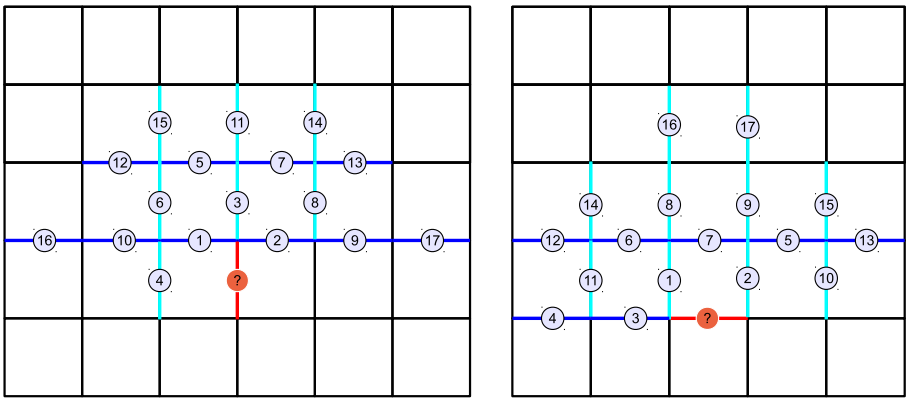


Figure 3.6: The representation of contour edges using two binary matrices. Black lines are marking the inactive contour edges. A cyan line is marking the active vertical edge, $V(x, y) = 1$, between $Z(x, y)$ and $Z(x, y-1)$. A blue line is marking the active horizontal edge, $H(x, y) = 1$, between $Z(x, y)$ and $Z(x-1, y)$.

The image contour is transmitted to the decoder by encoding the two binary matrices H and V. Although the information regarding the crack-edges is divided into two matrices, the coding of each vertical and horizontal edge is done using both binary matrices. Template Context Tree Model (TCTM) is the statistical model used to encode the representation, where two different templates are introduced. The template shown in Figure 3.7.(a) is used to encode a vertical edge, where the previous two lines in the two binary matrices are used to create the template context using 10 horizontal edges from H and 7 vertical edges from V. Let us denote T^v the context tree obtained using this template. The template shown in Figure 3.7.(b) is used to encode a horizontal edge. Similarly, the previous two lines in the two binary matrices are used to create the template context using 7 horizontal edges from H and 10 vertical edges from V. Let us denote T^h the context tree obtained using this template. Note that in both cases the length of the context is 17. The order, in which the edges are collected in the template context, was fixed after testing different orders for a set of five different depth-map images. Each image has an optimal order selection that could be found by a greedy method. We choose not to search for it because its method is increasing the algorithm's complexity and the obtained decrease in bitrate is very small.

The two binary matrices, V and H, are scanned row-wise by first traversing the row x from H, and then traversing row x from V. The symbols distributions, at each node in each context tree T^v and T^h, are collected in the first pass. The two context trees are then pruned, and the optimal context trees $\mathcal{T}^v$ and $\mathcal{T}^h$ are obtained. In the second pass, the contour edges saved in the two binary matrices are encoded using the two optimal trees [51, 70]. To lower the complexity of the algorithm and to decrease the runtime, in [P3], we developed two versions for the contour compression algorithm. We summarized the High Complexity (HiCo) version (described above) in an algorithm denoted Algorithm C (Alg. C), presented in Algorithm 3.2. In the second version of the algorithm, called Fast, the complexity is decreased by using only one pass to encode each binary matrix. Hence,



(a) (b)

Figure 3.7: The context template used to encode: **(a)** a vertical edge; **(b)** a horizontal edge. The cyan lines are marking the position of the vertical edges added in the templates. The blue lines are marking the positions of the horizontal edges added in the templates. The numbers in the circles are denoting the contour edges order in the template. Both images represent the image Z, with the contour edges marked as in Figure 3.6.

in FAST the balanced trees T^v and T^h are used to encode the contour edges, and the context template length (and tree depth), is decreased from 17 to 15.

The regions in the image must be separated by a continuous contour; otherwise we cannot distinguish two neighbor regions, which give the contour edge the propriety that each of its ends is connected with at least one other contour edge end. Exceptions appear only for the contour edges at the image boundary. This propriety is used in [P3] to improve the compression by finding deterministic cases where the contour edge can be predicted from its neighboring edges. Unfortunately, only for the vertical edges we found deterministic cases. In Figure 3.7.(a), if all the contour edges in the positions 1, 2, and 3 are inactive, then the unknown vertical edge marked with the red line is also inactive. A simple explanation is that, because all the contour edges in the three positions are inactive, all four pixels, between which these contour edges are set, are in the same region, and hence the last unknown edge out of the four is always inactive. If only one contour edge is active and has any position 1, 2, or 3, then the unknown vertical edge, marked with the red line in Figure 3.7.(a), is always active. The propriety described above is used here, since the unknown vertical edge has at the upper end a connection with one active contour edge, it must be active so that the contour can be continuous.

The non-stationarity propriety of the context distributions was used in publication [P3], where during the count collection, we try to down-weigh some counts in each context by halving the values of the symbols distribution. Note that the context that has all the contour edges inactive is the only context that has a sta-

Algorithm 3.2: **Algorithm C, contour compression stage in CERV**

The algorithm represents the image contour using contour edges, and encodes two binary matrices, V and H, using Template Context Tree Model (TCTM).

(0) Initialize the context trees T^v and T^h up to the maximum tree depth of 17.

(1) Do one pass through the binary matrices V and H and collect: the counts in the context tree T^v using the context template in Figure 3.7.(a), and the counts in the context tree T^h using the context template in Figure 3.7.(b).

(2) Prune the balanced trees, T^v and T^h, using dynamic programming, and encode the structure of the new optimal context trees, $\mathcal{T}^v$ and $\mathcal{T}^h$, using a Breadth-First Search (BFS) procedure.

(3) Do one pass through the binary matrices V and H. If the current contour edge is not in a deterministic case, then encode the contour edges using $\mathcal{T}^v$ and $\mathcal{T}^h$ (already available at the decoder); else set the corresponding deterministic contour edge.

tionary distribution. The concept is described in more details in Appendix A.2.

For the image ART (full size, left view) of size 1390×1110 the CERV algorithm obtains a result of 0.2065 bpp, i.e. a compression ratio of 38.7420 (see Section 5). The contours are compressed using Algorithm 3.2 in 0.1918 bpp, i.e. 92.87% of the final bitrate. The optimal context trees $\mathcal{T}^v$ and $\mathcal{T}^h$ have together a total of 1857 branches that are encoded using 227 bytes or 0.0013 bpp, i.e. 0.61% out of the contours bitrate.

3.3 Algorithms for region reconstruction

In the previous section, we presented the first stage of a lossless coder, the contour compression. In the second stage, for each region obtained from the segmentation, the encoder transmits to the decoder the information needed to reconstruct it. If all the pixels in the region Ω_ℓ have the same depth value d_ℓ, then d_ℓ is the only information needed to reconstruct the image. If the pixels in the region have different depth values, the linear predictive coding is used to obtain a prediction for each value, and then the computed prediction error is encoded. The two cases are described in the next subsections.

3.3.1 Predictive coding using mixtures of local predictors

Let us consider a region $\Omega_\ell = \{(x_i, y_i)\}_{i=1,2,\ldots,n}$ formed of n pixels. The predictive coding technique is predicting the depth value, $z_i = Z(x_i, y_i)$, found in a current pixel position (x_i, y_i), using the depth values $Z(x_t, y_t)$ of the pixels found on the positions $(x_t, y_t) \in \Omega_\ell$ in the causal neighborhood $\mathcal{N}_P(x_i, y_i)$, of the current pixel (x_i, y_i). Figure 3.8.(a) shows the pixel labeling for defining the causal neighborhood $\mathcal{N}_P(x_i, y_i)$ used to predict the pixel value at the position labeled "**?**". Two

d	c	e
a	**?**	
f		

(a) Pixel labeling for causal neighborhood in vertical and horizontal scanning

Mixture predictor index $n = 2^{b_3} + \ldots + 2^{b_0}$	Elementary predictors	
	row-wise scanning	column-wise scanning
$b_0 = 1$	$z_0 = c + a - d$	$z_0 = c + a - d$
$b_1 = 1$	$z_1 = a - c + e$	$z_1 = c - a + f$
$b_2 = 1$	$z_2 = a + \frac{e-d}{2}$	$z_2 = c + \frac{f-d}{2}$
$b_3 = 1$	$z_3 = [c \quad a \quad d \quad e]\ \mathbf{w}_r$	$z_3 = [c \quad a \quad d \quad f]\ \mathbf{w}_c$

(b) Mixtures of local predictors

Figure 3.8: **(a)** The causal neighborhood $\mathcal{N}_P(x_i, y_i)$ used to predict the pixel position (x_i, y_i), labeled '**?**'. The neighbor pixels labeled a, c, d and e are forming the causal neighbors in row-wise scanning, and those labeled a, c, d and f are forming it in the column-wise scanning. **(b)** The set of elementary predictors, z_0, z_1, z_2 and z_3, used in mixtures of predictors forming. Mixture index $n = 2^{b_3} + \ldots + 2^{b_0}$ is defining the set of elementary predictors $\mathcal{P}_n(\mathcal{N}(x_i, y_i)) = \{a, c, z_k | b_k = 1, \forall k \in \{0, 1, 2, 3\}\}$. The predictors z_0, z_1, z_2 are the best linear fits over the causal neighborhood, while z_3 is the best linear fit over the whole region.

scanning orders are usually used to traverse the pixels in the region mask of Ω_ℓ. Two causal neighborhoods are defined: a, c, d, and e for the *row-wise scanning*; and a, c, d and f for the *column-wise scanning*.

In [P1], both scanning orders are tested for each region of the segmentation, and one bit is used to encode the selection of a scanning order. For each found scanning order an optimal predictor, with mixture index n^*, is selected among $n \in \{1, \ldots, 15\}$ mixture predictors. For each mixture predictor, $\mathcal{P}_n(\mathcal{N}(x_i, y_i))$, a codelength estimation is computed and used to select the optimal predictor. A mixture predictor is formed from the set of four elementary predictors z_0, z_1, z_2 and z_3 (see Figure 3.8.(b)). Their selection is done by the binary representation of the mixture index, $n = 2^{b_3} + \ldots + 2^{b_0}$, as follows: if the bit b_k is set, $\forall k \in \{0, 1, 2, 3\}$, we have $b_k = 1$, then the elementary predictor z_k is added to the set of elementary predictors, defined as

$$\mathcal{P}_n(\mathcal{N}(x_t, y_t)) = \{a, c, z_k | b_k = 1, \forall k \in \{0, 1, 2, 3\}\}. \tag{3.1}$$

Finally, for the depth z_i found on the pixel position (x_i, y_i), its prediction $\hat{z}_i$ is

$$\hat{z}_i = median\{\mathcal{P}_n(\mathcal{N}(x_i, y_i))\}, \tag{3.2}$$

while the prediction error is computed as

$$\epsilon(i) = z_i - \hat{z}_i, \ \ i = 1, 2, \ldots, n. \tag{3.3}$$

Let us consider a region Ω_ℓ. For each of the row-wise and column-wise scanning, we compute the codelength estimation for each of the 15 mixture predictors having the mixture index $n = 1, 2, \ldots, 15$. We use the codelengths to select the optimal mixture index n^*. For example, if $n^* = 9$, then the binary representation of 9 is 1001, i.e. we have $b_0 = 1, b_1 = 0, b_2 = 0, b_3 = 1$. The mixture predictor is then set as $\mathcal{P}_n(\mathcal{N}(x_t, y_t)) = \{a, c, z_0, z_3\}$.

The last elementary predictor in the set, z_3, is designed using all the pixels in the region Ω_ℓ. The vectors of weights $\mathbf{w}_r$ and $\mathbf{w}_c$ are obtained by solving an LS problem. For the row-wise search, let us denote ψ_r the vector of depth values in the region Ω_ℓ. For the i^{th} element in the vector ψ_r, its four neighbors, with values $[c\ a\ d\ e]$, are set as i^{th} row in a matrix Φ_r. The LS parameters are obtained as $\mathbf{w}_r = (\Phi_r^T \Phi_r)^{-1} \Phi_r^T \psi_r$. Similarly, we obtain $\mathbf{w}_c = (\Phi_c^T \Phi_c)^{-1} \Phi_c^T \psi_c$ for the column-wise search.

Entropy coding of the prediction errors is done using the AMM with Laplace estimator. The prediction error vector is centered before coding, and an escape mode is introduced to encode large error values.

3.3.2 Constant model coding using neighbors list

A segmentation that divides an image into constant regions is an example of *over-segmentation*. Its regions are called constant because all the pixels inside a region have the same depth value, and the region can be reconstructed by encoding one constant model parameter. Let us consider that Z is divided into a set of n_Ω constant regions, $\Omega = \{\Omega_1, \Omega_2, \ldots, \Omega_{n_\Omega}\}$, where each region Ω_ℓ contains the depth value d_ℓ. Hence, Z is reconstructed losslessly if the set of depth values $\mathcal{D} = \{d_1, d_2, \ldots, d_{n_\Omega}\}$ is transmitted to the decoder after the segmentation. The over-segmentation of a depth-map image has the depth value of each constant region 'close' to the depth values of its neighbors, because of the type of information stored in Z. Let us consider that each region Ω_ℓ has K_ℓ neighboring regions, that are collected in a list $\mathcal{N}_\ell = \{\Omega_{\ell,1}, \Omega_{\ell,2}, \ldots, \Omega_{\ell,K_\ell}\}$. The depth value d_ℓ is 'close' to the list of depth values, $\mathcal{D}_\ell = \{d_{\ell,1}, d_{\ell,2}, \ldots, d_{\ell,K_\ell}\}$, corresponding to the list $\mathcal{N}_\ell$. Let us denote D_ℓ the vector that stores the list of k_ℓ unique values from $\mathcal{D}_\ell$, that are known at the moment when d_ℓ is encoded, where $D_\ell = [d_{\ell,1}\ d_{\ell,2}\ \cdots\ d_{\ell,k_\ell}]$.

In [P3], the set $\mathcal{D}$ is encoded by an algorithm that encodes each value d_ℓ using its list of likely values, denoted $\mathcal{L}_\ell$. The list $\mathcal{L}_\ell$ is constructed using the vector D_ℓ. If k_ℓ, the number of elements in D_ℓ, is $k_\ell > 2$, then the algorithm is using a clustering algorithm to perform the grouping of the values in D_ℓ around centers, that are selected sequentially using a threshold Δ. The clustering algorithm is using $\Delta = 5$, and is generating the centers as follows: **(a)** select as the center of the first cluster the first value in D_ℓ, i.e. $d_{\ell,1}$; **(b)** pass through D_ℓ and mark all elements in D_ℓ that are within Δ distance from the center $d_{\ell,1}$; **(c)** recompute the center of the cluster as the rounded mean of all values in the cluster and denote the center Q_i; **(d)** if the list D_ℓ is not empty go to (a). Let us denote n_Q the number of clusters obtained, where each cluster is having its center denoted $Q_i,\ i = 1, 2, \ldots, n_Q$.

Algorithm 3.3: **Algorithm Y, region reconstruction stage in CERV**

The algorithm encodes the information for region reconstruction based on a constant model. It is assumed that the image contours are already available at the decoder.

(0) Use region contours to find the set of constant regions, $\Omega = \{\Omega_1, \Omega_2, \ldots, \Omega_{n_\Omega}\}$, and collect depth value d_ℓ of each Ω_ℓ in the depth values set, $\mathcal{D} = \{d_1, d_2, \ldots, d_{n_\Omega}\}$.

(1) Initialize the set of variables:

(1.1) The context counters $N^I_{j,i_C} = \delta_C$, where $i_C = 1, 2, \ldots, 5$ is the context index, $j = 1, 2, \ldots, 2\Delta + 1$ is the index of the list of likely values, and $\delta_C = \frac{1}{2\Delta+1}$.

(1.2) The switch counters $N^S_{j,i_C} = \frac{1}{2}$, where $j \in \{0, 1\}$ is the switch index (or the switch value) in each context $i_C = 1, 2, \ldots, 5$.

(1.2) The default context counters $N^d_j = \frac{1}{2^B}$, for $j = 0, 1, \ldots, 2^B - 1$.

(2) Order the labels of regions in Ω in the order in which the regions are found in the image row-wise scanning. *(Do the same ordering for $\mathcal{D}$.)*

(3) For each Ω_ℓ, encode $d_\ell, \ell = 1 : n_\Omega$ *(iterate until the last element in $\mathcal{D}$)*, as follows:

(3.1) Find the list of neighboring regions $\mathcal{N}_\ell = \{\Omega_{\ell,1}, \Omega_{\ell,2}, \ldots, \Omega_{\ell,K_\ell}\}$, and their list of depth values $\mathcal{D}_\ell = \{d_{\ell,1}, d_{\ell,2}, \ldots, d_{\ell,K_\ell}\}$.

(3.2) Create the vector D_ℓ of unique know values from $\mathcal{D}_\ell$.

(3.3) Cluster the values in D_ℓ, and construct the list of likely values, L_ℓ.

(3.4) If d_ℓ is found in the vector L_ℓ, then set the switch $S_\ell = 1$, otherwise $S_\ell = 0$. Encode S_ℓ using $-\log_2 \frac{N^S_{S_\ell,i_C}}{\sum_j N^S_{j,i_C}}$ bits. Update the count $N^S_{S_\ell,i_C} \leftarrow N^S_{S_\ell,i_C} + 1$.

(3.5) If $S_\ell = 1$, encode the index position j^* of d_ℓ in the list L_ℓ, (for which $L_\ell(j^*) = d_\ell$), using $-\log_2 \frac{N^I_{j^*,i_C}}{\sum_j N^I_{j,i_C}}$ bits. Update the count $N^I_{j^*,i_C} \leftarrow N^I_{j^*,i_C} + 1$.

(3.5) If $S_\ell = 0$, encode the value d_ℓ (which is known to belong to the set $\mathcal{W}_\ell = \mathcal{A} \setminus L_\ell \setminus D_\ell$) using $-\log_2 \frac{N^d_{d_\ell}}{\sum_{j \in \mathcal{W}_\ell} N^d_j}$ bits. Update the count $N^d_{d_\ell} \leftarrow N^d_{d_\ell} + 1$.

If the number of clusters is $n_Q > 2$, then the algorithm selects the first two most populated clusters, that have the centers Q_1 and Q_2. If the distance between the two centers, Q_1 and Q_2, is smaller than Δ (i.e. $|Q_1 - Q_2| < \Delta$), then a unique cluster is created: n_Q is set to 1, and the new center is computed as the rounded mean of the values in the two clusters. If the number of resulted clusters is larger than two, then we set $n_Q = 2$ and only the two most populated clusters are kept to be used by the algorithm. Let us consider that the two clusters are having the centers denoted as Q_1 and Q_2. Finally, for the region Ω_ℓ, the list of likely values $\mathcal{L}_\ell$ is set, depending on the number of clusters n_Q found for D_ℓ, as:

- if $n_Q = 1$, then $\mathcal{L}_\ell = \{Q_1,\ Q_1 + 1,\ Q_1 - 1,\ Q_1 + 2,\ Q_1 - 2,\ \ldots\}$;
- if $n_Q = 2$, then $\mathcal{L}_\ell = \{Q_1,\ Q_2,\ Q_1 + 1,\ Q_1 - 1,\ Q_2 + 1,\ Q_2 - 1,\ Q_1 + 2,\ \ldots\}$.

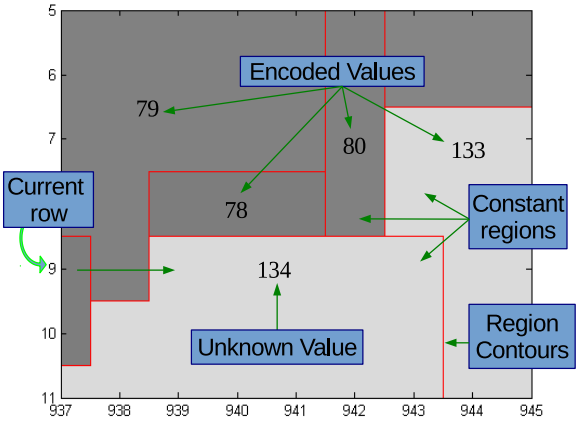


Figure 3.9: Example of a case where the unknown value $d_\ell = 134$ is encoded using the value of the known neighboring regions $D_\ell = [79\ 78\ 80\ 133]$.

List $\mathcal{L}_\ell$ is constructed in a specific order and d_ℓ is expected to be often located in the top ranks of $\mathcal{L}_\ell$. The depth values of the known neighbor regions, D_ℓ, are then excluded from $\mathcal{L}_\ell$, since the neighboring regions have different values. The first $2\Delta+1$ elements in the list are then saved in a vector L_ℓ, in the order they are found in $\mathcal{L}_\ell$. The algorithm distinguishes five contexts when encoding the value d_ℓ, that are having index i_C, created depending on the way the list $\mathcal{L}_\ell$ is constructed:

$$i_C = \begin{cases} 1, & \text{if } k_\ell = 1 \text{ (implicitly } n_Q = 1); \\ 2, & \text{if } k_\ell = 2 \text{ and } n_Q = 1; \\ 3, & \text{if } k_\ell = 2 \text{ and } n_Q = 2; \\ 4, & \text{if } k_\ell > 2 \text{ and } n_Q = 1; \\ 5, & \text{if } k_\ell > 2 \text{ and } n_Q = 2. \end{cases} \tag{3.4}$$

In (3.4), context $i_C = 1$ is used when one single unique neighbor value is known. If two unique neighbor values are known, then the distance between the values is checked: if the values are at a distance smaller than Δ, then the context $i_C = 2$ is used, else the values are at a large distance and the context $i_C = 3$ is used. If more than two unique neighbor values are known, then the clustering algorithm is used, and if one cluster is generated, then we use context $i_C = 4$, else $i_C = 5$. Finally the depth d_ℓ is encoded using the position j^* for which $L_\ell(j^*) = d_\ell$. Note that j^* is encoded using the statistics in the found context i_C.

In the cases when d_ℓ is not found in L_ℓ, the context $i_C = 6$ is used and a binary switch, S_ℓ, is encoded to signal the decoder if the current context is $i_C = 6$. In this context, d_ℓ is encoded using the list of default values, $\mathcal{W}_\ell$, obtained by excluding from the default alphabet the values that were checked and found not equal to d_ℓ, i.e. $\mathcal{W}_\ell$ is the set difference between $\mathcal{A}$ and the values in L_ℓ and D_ℓ, where $\mathcal{A} = \{0, 1, \ldots, 2^B - 1\}$ is the alphabet of the input values stored on B bits in the initial depth-map image Z (i.e. $\mathcal{W}_\ell = \mathcal{A} \setminus L_\ell \setminus D_\ell$).

In Algorithm 3.3 we present a summary of the region reconstruction stage from CERV [P3], which was denoted Algorithm Y (Alg. Y). Let us now consider the example from Figure 3.9, which is a zoom in inside Figure 2.3.(a). In the current row (i.e. row number 9), the next value to be encoded is $d_\ell = 134$ and its corresponding region Ω_ℓ has $k_\ell = 4$ neighbor regions with known values. The vector of unique known values is $D_\ell = [79 \quad 78 \quad 80 \quad 133]$. Next we apply the clustering algorithm and create the list of likely values L_ℓ as follows:

- Select the center of the first cluster as the first value found in D_ℓ, i.e. $q_1 = 79$. Pass throw D_ℓ and mark all the elements that are within the distance $\Delta = 5$ from q_1, i.e. inside the interval $[q_1 - \Delta, q_1 + \Delta] = [74, 84]$. The set of elements marked for the first cluster is $S_{q_1} = \{79,\ 78,\ 80\}$, and the center of the cluster is recomputed using S_{q_1} as $Q_1 = \frac{79+78+80}{3} = 79$.
- The remaining unmarked element in D_ℓ is selected as the center of the second cluster, i.e. $q_2 = 133$. This cluster has $S_{q_2} = \{133\}$, and its recomputed center is $Q_2 = 133$.
- Since $|Q_1 - Q_2| > \Delta$ (i.e $54 > 5$), then $n_Q = 2$ and $\mathcal{L}_\ell = \{Q_1,\ Q_2,\ Q_1 + 1,\ Q_1 - 1,\ Q_2 + 1,\ Q_2 - 1,\ Q_1 + 2,\ \ldots\}$, i.e. $\mathcal{L}_\ell = \{79,\ 133,\ 80,\ 78,\ 134,\ 132,\ 81,\ 77,\ 135,\ 131,\ 76,\ 82,\ \ldots\}$.
- Using $n_Q = 2$ and $k_\ell = 4$ in (3.4) the context index $i_C = 5$ is selected.
- The known values from D_ℓ are excluded from $\mathcal{L}_\ell$ and the first $2\Delta + 1 = 11$ elements are saved in $L_\ell = [134 \ \ 132 \ \ 81 \ \ 77 \ \ 135 \ \ 131 \ \ 82 \ \ 76 \ \ 136 \ \ 130 \ \ 83]$.
- Search for the position j^* of the unknown value $d_\ell = 134$ inside the vector L_ℓ for which $L_\ell(j^*) = d_\ell$. The value d_ℓ is found inside L_ℓ at $j^* = 1$, therefore the region's corresponding switch is set as $S_\ell = 1$.
- $S_\ell = 1$ and $j^* = 1$ are encoded as described at steps (3.4) and (3.5) in Alg. Y.

3.4 Algorithms summary

In this chapter, we described the algorithms introduced by three publications [P1, P3, P5]. Each publication is presenting a lossless compression algorithm with two stages, where the following information is encoded: in the first stage the image contour, and in the second stage the information needed for region reconstruction.

The algorithm from [P1] is compressing the contour by transmitting to decoder the vertex positions, and is using different statistical models to encode its sequence of symbols (see Section 3.2.1). The regions are reconstructed using a predictive coding technique (see Section 3.3.1). The algorithm is denoted New Complex Version (NCV), because more complex procedures are improving the results from [82]. NCV is designing also a segmentation in the first stage, in contrast to [P3] and [P5] which are using an over-segmentation. The segmentation algorithm is further developed in [P2], see Section 4.2. Its main idea is to create large regions by collecting pixels into maximum connected sets, where a current pixel position (x, y) is added to the set only if the absolute difference, between its depth value $Z(x, y)$ and the depth value of its neighboring pixels that are already in the set, is not exceeding a variability threshold denoted λ. The variability threshold is set in

turn as $\lambda = 1, 2, 3$, and the connected sets of pixels, having the cardinality larger than N_λ pixels, are labeled as a region. The algorithm is simple and depends on empirically set values N_λ and on the selected order in which the pixels are added to the connected sets.

CERV algorithm, from [P3], is transmitting to the decoder the contour using the contour edges encoded by Alg. C (see Section 3.2.2), and is reconstructing the regions using Alg. Y (see Section 3.3.2). Two versions were developed for the algorithm: CERV Fast version (CERV-Fast), that uses in Alg. C one pass through the image; and CERV High Complexity version (CERV-HiCo), having all steps from Alg. C. APC algorithm, from [P5], is transmitting to the decoder the image contour using a vertex representation (see Section 3.2.1), while Alg. Y from [P3] is used to encode the depth value of each constant region. CERV and APC are our best lossless compression coders, and there is no dramatic difference between their results, as we show in Chapter 5. However, if the image contains a large number of regions, i.e. the contour map has a high contour edge density, CERV performs better, while if the image is very simple and the constant regions are large, APC performs better.

Chapter 4

Lossy Compression of Depth-Map Images

"If you can't fly then run, if you can't run then walk, if you can't walk then crawl, but whatever you do you have to keep moving forward."

— Martin Luther King, Jr.

"If you are going through hell, keep going."

— Winston Churchill

In this chapter, the state of the art coders for lossy compression of depth-map images are briefly presented in the first section, while our four algorithms from [P2, P4, P6, P7] are described in the following sections. The second and third sections present algorithms based on image segmentation, in the fourth section we tackle the problem of progressive coding, while in the fifth section we study the parameterization of planar models, based on the selection of three pixel locations in a region, in order to ensure a better rate-distortion performance.

4.1 State of the art coders

The field of lossy compression of depth-map was intensively studied, and during the recent years a lot of algorithms were developed, based on different approaches like: quad-tree decomposition [62], platelets [96], wedgelets [23], block partitioning [106], down and up sampling of the image [64] or image decompositions [40]. In video compression, H.264/AVC was expanded, with a lot of algorithms, which modified parts of the code to improve the compression. However, the H.264/AVC standard was recently complemented by the new High Efficiency Video Coding (HEVC) standard. Other algorithms were designed especially for compressing the depth-map images obtained by the Kinect [105] sensor.

In [62], Morvan *et al.* developed an algorithm that encodes the quad-tree decomposition of the depth-map image and uses a region reconstruction algorithm based on Wedgelets [23] and Platelets [61, 96]. The depth-map image contains

smooth regions delineated by sharp edges. The smooth regions were modeled by piecewise-linear functions and the sharp edges by straight lines. Four different piecewise-linear functions were used in modeling: for approximating smooth regions they use a constant and a linear function, while for compressing the depth discontinuities (the sharp edges) they use a wedgelet and a platelet function. Their method was able to outperform the JPEG-2000 [92] encoder with $1-3$ dB. The Region Of Interest (ROI) feature of the JPEG-2000 encoder was used by Krishnamurty *et al.* in [43]. The sensitivity of the rendering error depends on image content, but a bigger impact has the quality of the depth-map, that is why they consider the ROI coding, where they identify those regions of the image where the depth-map must have a good quality. Their method offers an improvement of 1.1 dB over the JPEG-2000 encoder. This group of methods used elaborated mathematical tools to preserve the edges in the depth-map image, but they were later outperformed by simpler algorithms using other approaches.

Another approach is based on the idea of creating a different image representation using a decomposition of the depth-map image. A hierarchical decomposition was presented in [40], where the image is decomposed into three disjoint images plus a description image, as follows: the regular mesh image contains the depth-map levels; the edge-region image contains the edges of the depth-map areas; the no-edge-region image contains the non-edge depth-map areas; and the number-of-layer image contains information about the class each depth map value belongs to. All images are then entropy coded by H.264/AVC. A binary tree triangular decomposition is presented in [18], where values inside each triangle are reconstructed by a triangle based planar approximation. The algorithm has two stages: **(i)** the depth analysis stage, where the tree level decomposition is obtained by imposing MSE and Percentage of ERRored pixels (PERR) thresholds; **(ii)** the coding stage, where the tree is encoded by the 7-Zip implementation of Lempel-Ziv Markov chain algorithm, and the sorted depth values are entropy coded by the difference between two consecutive values. Another example of a lossy compression algorithm based on depth-map image decomposition is proposed by Milani and Cavagno in [57], where the image is first over-segmented into a large set of small regions using the Graph-Based Segmentation algorithm [24], and then the regions are merged according to their average depth value and the number of objects set to be identified in the depth-map image by the coding routine. Each object is coded by a binary mask and has an average depth value associated. A prediction for the current depth image is first built and then refined by computing the residual signal and iterating the coding process. The H.264/AVC standard is used to encode the residual signal. This group of methods builds upon the idea of utilizing an entropy coder to encode the entities that represent the depth-map image. They were later outperform due to the development of better entropy coders and because of the high number of entities needed to be encoded.

Another family of approaches is based on the conversion of the depth-map into a 3D mesh. The 3D surface models are represented using regular (or semi regular) meshes, where in a regular mesh, each vertex is connected to the same number of edges (six in case of triangular meshes), and the edge lengths are close to an average

value. All the depth-maps are brought to a common 3D space and are resulting a large 3D point cloud, which represent the starting point of the algorithm. In [67], multiple depth-map geometric proxies are compressed and used to create the 3D mesh. The geometry proxy is an approximate description of the scene, and it can be a parametric object such as a bounding box, a best-fit ellipsoid or an approximate polygon-based model of the scene. The methods were developed for a different type of application, but they proposed interesting concepts for the compression of different patches of a depth-map image.

In another approach, Oh *et al.* [64] presented a method that subsamples the depth information, and then it reconstructs the image using an adaptive interpolating filter. The method is based on the idea of efficiently compressing object boundaries, which are more sensitive to compression artifacts. The reconstruction filter, introduced by the method, consists of a frequent-low-high filter and a bilateral filter. The method offered improvements over the H.264 standard and a reduced distortion around object boundaries. The precision of the boundary information was improved even further in [38], where the method proposed efficient intra prediction modes around object boundaries. Due to a smooth depth-level distribution within an object and a sharp depth-level variation near object boundaries the method was able to achieve a gain of $2.5 - 4$ dB over H.264. This group of methods is focusing on the idea of preserving the edges in the depth-map image using different approaches. They all show better results comparing to the H.264 standard.

The observation that "pixels of similar depth have similar motion" was exploited in several articles. In [19], the authors used this observation to eliminate a priori the unlikely coding modes, and they achieved a 40% faster H.264 encoding at the expense of 0.7% increase in bitrate and 0.07 dB decrease in PSNR. In [21], the authors focused on the motion prediction of arbitrarily shaped sub-blocks and they used predictive coding to predict the next chain-code symbol of an F4 representation using a linear prediction model. Their method obtained an average overall bitrate reduction of up to 30% over H.264. These methods are emphasizing the role of predictive coding and the successful use of chain-code representations in depth-map image compression.

In another approach, the authors proposed methods based on the idea of segmentation or block partitioning. In [106], a segmentation is obtained using the depth and color information and the regions of the image are classified as "inner motion" regions, "background subtracted with edges" regions and "edge" regions. The H.264/AVC standard is encoding the image's macro-blocks according to the features of the regions they belong to. A geometry-adaptive block partitioning algorithm was introduced in [20], and modified in [38], so that no side information is required for block partitioning, because the partitioning information is estimated from the previous reconstructed neighboring blocks. In [36], a segmentation of the depth-map image is obtained by selecting the most important contours using piecewise linear approximations; the contour lines are then encoded by JBIG2 [32]. As a result, the method managed to preserve the edge information and to achieve less geometric distortions, having a gain of 9 dB in PSNR compared to

the JPEG-2000 encoder. In [22], the authors are using an adaptive lifting scheme to preserve edges. Although a small gain of around 0.5 dB is obtained in PSNR, a quality improvement is achieved in the synthesized view due to the preservation of the edges. The block partitioning is a concept widely used in compression, e.g. in the H.264/AVC and HEVC standards. This group of methods is successfully exploiting different variations of the concept.

An algorithm with progressive coding functionality was first proposed in [17], where "reversible cellular automata" is used to achieve spatial scalability. In [53, 54], geometry information is conveyed by prioritized breakpoints found by a breakpoint-adaptive transform. In [104], the authors present an approach where the image is segmented into a desired number of regions, where each region is represented using region shape and the averaged value inside the region. In [73], the authors use an approach based on lifting operations and the piecewise planar model for the images. Similarly, in [41] the graph-based transform is used. All these methods propose inspiring ideas that were used in our algorithms.

In November 2010, Microsoft launched the *XBOX 360* video game console, which incorporates a motion sensor, that is placed in an input device called *Kinect*. The Kinect [105] sensor is able to estimate a depth-map image, and then use it to interpret some specific gestures, which made XBOX an electronic device with a completely hands-free control. In [55], the researchers from Microsoft developed an algorithm for compressing the 16-bit depth-maps generated by Kinect, using a simple low complexity algorithm. They started from the observation that the accuracy of the depth estimate actually decreases with the inverse of the depth. When reading a depth value which is twice as far as another one, the error in the estimated depth is two to four times larger. The algorithm has three steps: inverse coding of the image, a simple prediction after a raster scan, and adaptive Run-Length/Golomb-Rice (RLGR) coding [50].

In another approach, the planar model is used in the depth-map image compression to reconstruct different regions from the image. In [58], the color data segmentation is used to predict the depth-map image, while each region is reconstructed using a planar model. The plane parameters are encoded only for the region where the approximation of the planar model is sufficiently accurate for the target bitrate, while the others are encoded using H.264/AVC. A similar algorithm was presented in [72], where the compression scheme is using a joint depth and texture coding scheme, by extending the Locally Adaptive Resolution (LAR) codec [66]. In [65], the planar model for depth-map image compression is studied intensively. The method is placing the segmentation process in the virtual 3D point cloud, rather than simply in a left or right view. However, the planar models are relatively few in number, and they are obtained by running a graph-cut co-segmentation algorithm. This group of articles studied the use of the planar model and they proposed different solutions for the problem of parameterization and selection of the planar model. In our latest work we manage to exploit better the planar models and to increase the number of regions where it is used.

Other approaches were proposed for depth-map video coding, and are focused on solving problems like pixel-based motion estimation [48], disparity estimation

[16], and many others. The most commonly used approach is the modification of the H.264 standard [49, 100, 102, 103], where patches of code are inserted in the standard and the methods manage to achieve a better result than H.264.

In the following sections four lossy compression algorithms are proposed, where different concepts of the state of the art coders are inserted in our approach to help us in our quest to outperform the state of the art coders.

4.2 Segmentation into constrained regions

Publication [P2] describes the first lossy compression algorithm we developed. The algorithm generates different segmentations by constraining the variability and the number of distinct depth levels inside each region. Each segmentation creates a lossy image that is compressed using prediction and quantization techniques.

The main idea in [P2] is to use prediction inside each region to encode residual errors with a distribution close to the exponential distribution, and to obtain different segmentations suitable for compression. Each segmentation is having a corresponding lossy version of Z and is represented by a point in the Rate-Distortion (RD) curve.

To ensure low prediction errors, we constrain the regions to contain connected pixels in 4-connectivity having only small differences between their depth values. Let us consider a region in the image, $\Omega_\ell = \{(x_i, y_i)\}_{i=1,2,\ldots,n}$, formed of n pixels, containing pixels with the depth values denoted $z_i = Z(x_i, y_i),\ i = 1, 2, \ldots, n$. For a pixel position (x, y), we define its neighborhood in the 4-connectivity as $\mathcal{N}_4(x, y) = \{(x-1, y), (x+1, y), (x, y-1), (x, y+1)\}$. For the pixel position (x_t, y_t) and the region Ω_ℓ, we define the variability of the pixel as:

$$V(x_t, y_t) = \min_{(x_i, y_i) \in \mathcal{N}_4(x_t, y_t) \cap \Omega_\ell} |Z(x_t, y_t) - Z(x_i, y_i)|. \tag{4.1}$$

If in Ω_ℓ we impose a variability threshold $\lambda = 0$ (i.e. $\forall (x_i, y_i) \in \Omega_\ell$, $V(x_i, y_i) \leq \lambda$), then Ω_ℓ is a constant region. When the variability threshold is increased, the segmentation is able to find in the image regions containing planar objects.

The contour compression has the highest percentage in the final codelength and that is why, in the first step, the algorithm is smoothing the contours as follows: if a pixel is surrounded by a majority of neighbors having very similar depth value, then it is better to smooth the contour by setting the pixel's value with the value of its neighbors. If the pixel is the single position in a region with cardinality one, then four contour edges are eliminated from the contour, which are encoded using an anchor point (the most expensive information that is sent to the decoder). Note that the distortion introduced by this change is very small. The same procedure is applied when three out of the four pixel positions from $\mathcal{N}_4(x, y)$ have the same depth value. The change of the pixel's value is straitening the contour by replacing three contour edges with one contour edge, and increases the likelihood that introduced contour edge is codified by a 3OT symbol '0' or '1' instead of the three initial symbols encoding the three contour edges, where one of them is '2'.

Algorithm 4.1: **Lossy Constrained Region Segmentation (L-CRS)**

The algorithm encodes the image Z by generating a segmentation, and by encoding the contour and applying prediction and quantization techniques for region reconstruction.

(0) Smooth the contour by changing the contour edges separating pixels with similar values as the majority of their neighbors: if at least 3 neighbors in $\mathcal{N}_4(x, y)$ have the same depth value, v, and if $|Z(x, y) - v| \leq 2$, then set $Z(x, y) = v$.

(1) Find the connected sets of pixels having variability $\lambda = 0$ (i.e. constant regions).

(2) Label as regions the sets of connected pixels containing more than N_1 pixels. Out of the remaining pixels, find the connected sets of pixels having variability at most $\lambda = 1$ and having the maximum number of distinct depth values $\gamma = 2$.

(3) Label as regions the sets of connected pixels with cardinality at least N_2. Out of the remaining pixels, find connected sets of pixels having variability at most $\lambda = 2$.

(4) Label as regions the sets of connected pixels containing more than 100 pixels. Out of remaining pixels, find connected sets of pixels having variability at most $\lambda = 3$.

(5) Declare the remaining connected sets of pixels as regions.

(6) Remove the regions having less than five pixels by merging them with the biggest neighbor region, and set the region pixels with the value of the selected region.

(7) Encode the region contours by encoding vertices using AMM and Laplace estimator.

(8) Quantize and encode the regions with almost constant depth, using $T_3 = 50$.

(9) For the remaining regions use the near-lossless predictive compression by predicting using the Martucci predictor and quantizing the residual values using (4.2).

The algorithm has the possibility of distinguishing the case when an object, or the image's background (a vertical plane), is contained in a large constant region. That is why, after obtaining the set of connected pixels having $\lambda = 0$, the large sets, that contain at least N_1 pixels, are labeled as regions, while the remaining pixels are used to re-determine new sets of connected pixels constrained with an increased variability. In the next step, we use $\lambda = 1$ and also constrain the regions to contain a small number of distinct depth values, say $\gamma = 2$, so we can quantize these regions better, as explained below. In the next two steps we use $\lambda = 2$ and $\lambda = 3$, and label as regions the largest sets of pixels that contain at least N_2 and $N_3 = 100$ pixels. Note that these regions contain mostly objects with planar models. The region size is constrained because we want to decrease the length of the residual error vector, but also to be able to have a small prediction error over a large area of the image. The remaining sets of pixels are declared regions. The smallest regions, denoted here pixel regions, that contain five or less pixels, are merged with the biggest neighboring regions, and hence less contour edges are encoded in the contour compression stage. This ends the segmentation generation stage, where N_3 was fixed, and N_1 and N_2 are set with values between 50 and 1000 pixels, depending on the image size and characteristics.

In the next stage, quantization techniques are used as follows. If a region

contains only a small number of distinct levels, denoted γ, then is better to quantize and encode the region in a much simpler procedure, denoted *Quantization and coding of regions with almost constant depth*:

- If the region contains $\gamma = 2$ distinct depth levels, i.e. g and $g + 1$, then the quantized depth value is set as g^*, the value which occur most often in the region, which ensures that the lowest distortion is introduced in the region.
- If the region contains $\gamma = 3$ distinct depth levels, then the optimal value g^* is set as the truncated average over the region. The distortion introduced by g^* is then computed and compared with a threshold T_3, and if it is above the threshold, then we use the similar procedure as previously and encode the optimal quantization level for the region reconstruction stage. However, if the distortion is too high and the threshold is exceeded, then a different procedure based on prediction and quantization is used instead.

The remaining regions, that failed the distortion constraint, are first predicted using the optimal predictor selected from the mixture of predictors, as described in Section 3.3.1. However, the results showed that the Marttuci predictor is selected most of the time. For the region Ω_ℓ, containing pixels with the depth values $z_i = Z(x_i, y_i),\ i = 1, 2, \ldots, n$, the predictor computes the set of predictions, $\{\hat{z}_i\}_{i=1,2,\ldots,n}$, for each corresponding pixel. Prediction errors ϵ_i are then computed as $\epsilon_i = z_i - \hat{z}_i,\ i = 1, 2, \ldots, n$. Quantization of the values is done in the same way as in LOCO-I [93], i.e. the prediction errors are uniformly quantized by:

$$Q(\epsilon_i) = sign(\epsilon_i) \left\lfloor \frac{|\epsilon_i| + \eta}{2\eta + 1} \right\rfloor, \tag{4.2}$$

where the *sign* (*signum*) function returns '1' for a positive argument, '-1' for a negative argument, and '0' for a '0' argument. The decoder is reconstructing z_i as

$$\tilde{z}_i = \hat{z}_i + Q'(\epsilon_i) \cdot (2\eta + 1), \tag{4.3}$$

where the reconstruction level is biased from the midpoint of the quantization interval towards zero, to account for the typical monotonic decreasing Probability Density Function (PDF) of the absolute value of the residual:

$$Q'(\epsilon_i) = Q(\epsilon_i) - \frac{sign(Q(\epsilon_i)) \cdot \mu(\eta)}{2\eta + 1}. \tag{4.4}$$

The tests showed that the best results are obtained for the bias term corresponding to $\mu(\eta) = \eta$. The coding of the quantized prediction errors is done by applying AMM with Laplace estimator. The vector of errors is centered first, and an escape mode is introduced. The image contour is encoded in a similar way as in [P1].

When the targeted bitrate reaches the low range (say below 0.2 *bpp*), then we set the last significant bit, in the binary representation of the depth values, as the dominant value between the bits gathered from the image. The algorithm is denoted Lossy Constrained Region Segmentation (L-CRS), and its summary is presented in Algorithm 4.1. To improve the results at high bitrates, the step (6) from the L-CRS algorithm is eliminated and the Near-Lossless Constrained Region Segmentation (NL-CRS) algorithm is obtained.

4.3 Greedy Slope Optimization

Publication [P4] presents an image segmentation algorithm, having the possibility to generate segmentations for any given distortion. For each segmentation a corresponding lossy version of Z is obtained. The algorithm consists of two phases. In the first phase, the regions in the current lossy image are merged as follows: at each iterative step we search, and then merge, the optimal pair of two regions that introduces the smallest distortion and the highest bitrate reduction. In the second phase, the regions in a template image are split using straight lines. In the RD plot, each point of the curve is corresponding to a lossy image created by the segmentation of the initial image.

Let us consider now that the initial image is denoted Z_0, and is initially divided into the set of n_Ω constant regions $\Omega = \{\Omega_1, \Omega_2, \ldots, \Omega_{n_\Omega}\}$, where each region Ω_ℓ contains pixels having same depth value d_ℓ, i.e. the set Ω has a corresponding set $\mathcal{D} = \{d_1, d_2, \ldots, d_{n_\Omega}\}$. The algorithm from [P4] was denoted Greedy rate-distortion Slope Optimization (GSO). The merging phase is called GSO with region *merging* (GSOm), and generates a sequence of images $\mathcal{Z}_n$. The splitting phase is called GSO with region *splitting* (GSOs), and generates a sequence of images $\mathcal{Y}_n$.

4.3.1 GSO with region merging

GSOm is using the initial image Z_0 to generate a sequence of n lossy images $\mathcal{Z}_n = \{Z_1, Z_2, \ldots, Z_n\}$, where each image in the sequence has a given distortion. From the current image, say Z_τ, GSOm is searching and merging the necessary number of regions so that it can generate the next image in the sequence, $Z_{\tau+1}$.

Let us consider, at iteration τ, the current image Z_τ is represented using: the set of $n_\Omega^{(\tau)}$ constant regions $\Omega(Z_\tau) = \{\Omega_1^{(\tau)}, \Omega_2^{(\tau)}, \ldots, \Omega_{n_\Omega^{(\tau)}}^{(\tau)}\}$, the corresponding set of $n_\Omega^{(\tau)}$ depth values $\mathcal{D}(Z_\tau) = \{d_1^{(\tau)}, d_2^{(\tau)}, \ldots, d_{n_\Omega^{(\tau)}}^{(\tau)}\}$, and the contour $\Gamma(Z_\tau)$ that separates the regions, which is the reunion of all contour segments $\Gamma_{i,j}^{(\tau)}$, where $\Gamma_{i,j}^{(\tau)}$ separates the pair of regions $\left(\Omega_i^{(\tau)}, \Omega_j^{(\tau)}\right)$. GSOm is evaluating each possible merge and is selecting the optimal pair based on a criteria. Next we study the merge, obtain an evaluation of the merge, and formulate the criteria.

Let us consider a pair of regions $\left(\Omega_i^{(\tau)}, \Omega_j^{(\tau)}\right)$, denoted simply $p = (i^{(\tau)}, j^{(\tau)})$, which is separated by their contour segment $\Gamma_p = \Gamma_{i,j}^{(\tau)}$. The next image in the sequence, $Z_{\tau+1}$, is generated from the current image Z_τ by merging the optimal pair of regions. If we choose to merge the pair of region $p = (i^{(\tau)}, j^{(\tau)})$, then a new region $\Omega_\ell^{(\tau+1)} = \Omega_i^{(\tau)} \cup \Omega_j^{(\tau)}$ is obtained, and the depth of all the region pixels is set to the truncated average of all initial depth values of the pixels, i.e. $d_\ell^{(\tau+1)} = \left\lfloor \frac{\sum_{(x,y)\in\Omega_\ell^{(\tau+1)}} Z_0(x,y)}{m_\ell^{(\tau+1)}} \right\rfloor$, where $m_\ell^{(\tau+1)}$ is the number of pixels in $\Omega_\ell^{(\tau+1)}$. The image $Z_{\tau+1}$ contains the new region $\Omega_\ell^{(\tau+1)}$ and the remaining regions from Z_τ, and can be represented by relabeling the elements used to describe Z_τ, i.e. using:

the set $\Omega(Z_{\tau+1}) = \{\Omega_1^{(\tau+1)}, \Omega_2^{(\tau+1)}, \ldots, \Omega_{n_\Omega^{(\tau+1)}}^{(\tau+1)}\}$ of $n_\Omega^{(\tau+1)} = n_\Omega^{(\tau)} - 1$ constant regions, the corresponding set $\mathcal{D}(Z_{\tau+1}) = \{d_1^{(\tau+1)}, d_2^{(\tau+1)}, \ldots, d_{n_\Omega^{(\tau+1)}}^{(\tau+1)}\}$ of depth values, and the contour $\Gamma(Z_{\tau+1}) = \Gamma(Z_\tau) \backslash \Gamma_{i,j}^{(\tau)}$. The merging has two consequences. First, by merging the pair of region $p = (i^{(\tau)}, j^{(\tau)})$ into $\Omega_\ell^{(\tau+1)}$, the values of all the region pixels are modified, and an increase in distortion, ΔD_p, is introduced. Secondly, the contour segment $\Gamma_{i,j}^{(\tau)}$, that separated the two regions, is eliminated and the set $\mathcal{D}(Z_{\tau+1})$ contains one element less, which is producing a decrease in bitrate, ΔR_p.

We evaluate ΔR_p by adding the estimated codelength of the eliminated contour and the codelength of encoding one depth value. The contour codelength is evaluated using the model: $C(\Gamma_p) = C_1 \cdot L(\Gamma_p)$, where C_1 is a constant representing the cost of encoding a contour edge, and $L(\Gamma_p)$ is the number of contour edges in Γ_p. In [P4] we used $C_1 = 1.5$ bits. In the reconstruction stage, the set $\mathcal{D}(Z_{\tau+1})$ has $n_\Omega^{(\tau+1)} = n_\Omega^{(\tau)} - 1$ elements, i.e. one depth value is not encoded and an estimated $C_2 = 8$ bits are saved. Now we can estimate the decrease in bitrate as:

$$\Delta R_p = C_1 \cdot L(\Gamma_p) + C_2. \tag{4.5}$$

Note that the cost for encoding one 3OT symbol is $log_2(3) \approx 1.585$. When a sequence of 3OT symbols is encoded using a statistical model like AMM or CTM, the cost is decreased, but other auxiliary information is required to be encoded, which depends on many variables: image contour shape, image size, number of contour segments, or contour edge density of image areas. Later tests have shown that the cost of encoding one 3OT symbol may vary between 1.1 and 1.9 bits.

We evaluate the change in distortion as $\Delta D_p = MSE(Z_\tau) - MSE(Z_{\tau+1})$, which depends only on the regions involved in merging: $\Omega_i^{(\tau)} \cup \Omega_j^{(\tau)} \rightarrow \Omega_\ell^{(\tau+1)}$, due to the way the sequence of images, $\mathcal{Z}_n$, is generated. Let us denote for any $\Omega_k^{(\tau)} \in \Omega(Z_\tau)$, $k = 1, 2, \ldots, n_\Omega^{(\tau)}$, having $m_k^{(\tau)}$ pixels and depth value $d_k^{(\tau)}$, the following:

(i) the sum of the original depth values as $\phi_k^{(\tau)} = \sum_{(x,y)\in\Omega_k^{(\tau)}} Z_0(x,y)$;

(ii) the sum of the squared original depth values as $\varphi_k^{(\tau)} = \sum_{(x,y)\in\Omega_k^{(\tau)}} Z_0(x,y)^2$;

(iii) the sum of the squared reconstruction errors as
$\rho\left(\Omega_k^{(\tau)}\right) = \sum_{(x,y)\in\Omega_k^{(\tau)}} \left(Z_0(x,y) - d_k^{(\tau)}\right)^2 = \varphi_k^{(\tau)} - 2\phi_k^{(\tau)} d_k^{(\tau)} + m_k^{(\tau)} \left(d_k^{(\tau)}\right)^2$.

We can compute MSE over the region $\Omega_k^{(\tau)}$ as $MSE_k^{(\tau)} = \frac{1}{n_r n_c}\rho(\Omega_k^{(\tau)})$, and MSE over the image Z_τ as $MSE(Z_\tau) = \sum_{k=1}^{n_\Omega^{(\tau)}} MSE_k^{(\tau)}$. When the pair of regions $p = (i^{(\tau)}, j^{(\tau)})$ is merged, the new region $\Omega_\ell^{(\tau+1)}$ contains $m_\ell^{(\tau+1)} = m_i^{(\tau)} + m_j^{(\tau)}$ pixels, has the sum of the pixels initial depth values $\phi_\ell^{(\tau+1)} = \phi_i^{(\tau)} + \phi_j^{(\tau)}$ and the sum of the squared depth values $\varphi_\ell^{(\tau+1)} = \varphi_i^{(\tau)} + \varphi_j^{(\tau)}$, and it is having the depth

Algorithm 4.2: **Greedy Slope Optimization with region merging (GSOm)**

The algorithm generates the sequence $\mathcal{Z}_n = \{Z_1, Z_2, \ldots, Z_n\}$ *using the initial image* Z_0.

(0) Smooth the contour by changing the contour edges separating pixels with similar values as the majority of their neighbors: if at least 3 neighbors in $\mathcal{N}_4(x_i, y_i)$ have the same depth value, v, and if $|Z_0(x_i, y_i) - v| \leq 2$, then set $Z_0(x, y) = v$.

(1) Set $\tau = 0$, and find in Z_τ constant regions and represent the image using: set of $n_\Omega^{(\tau)}$ regions $\Omega(Z_\tau)$, corresponding set of depth values $\mathcal{D}(Z_\tau)$, and contour $\Gamma(Z_\tau)$.

(2) For every region $\Omega_k^{(\tau)} \in \Omega(Z_\tau)$, $k = 1, 2, \ldots, n_\Omega^{(\tau)}$ in Z_τ, having $m_k^{(\tau)}$ pixels and depth value $d_k^{(\tau)}$, compute $\phi_k^{(\tau)}, \varphi_k^{(\tau)}, \rho(\Omega_k^{(\tau)})$, and $MSE_k^{(\tau)}$. Use $MSE_k^{(\tau)}$ to compute $MSE(Z_\tau)$ and $PSNR(\tau)$.

(3) For every pair or regions $p = (i^{(\tau)}, j^{(\tau)})$ in Z_τ, separated by $\Gamma_p = \Gamma_{i^{(\tau)}, j^{(\tau)}}$:

  (3.1) Estimate the rate variation, ΔR_p, using (4.5).

  (3.2) Compute the change in distortion, ΔD_p, using (4.6).

  (3.3) Compute the slope, λ_p, using (4.7).

(4) Sort increasingly first n_M slope values in vector $\boldsymbol{\lambda} = [\lambda_{p_1}, \lambda_{p_2}, \ldots, \lambda_{p_{n_M}}]$, with corresponding the list of pairs $P_M = [(i_1, j_1), (i_2, j_2), \ldots, (i_{n_M}, j_{n_M})]$.

(5) While $P_M \neq \varnothing$, merge the first pair, $p_1 = (i_1, j_1)$, and update variables as follows:

  (5.1) Merge the pair of region $p_1 = (i_1, j_1)$ from Z_τ, and compute for the new region $\Omega_i^{(\tau)} \cup \Omega_j^{(\tau)} \rightarrow \Omega_\ell^{(\tau+1)}$ the corresponding values of $m_\ell^{(\tau+1)}$, $d_\ell^{(\tau+1)}$, $\phi_\ell^{(\tau+1)}$, $\varphi_\ell^{(\tau+1)}$, $\rho(\Omega_\ell^{(\tau+1)})$, and $MSE_\ell^{(\tau+1)}$.

  (5.2) Obtain $MSE(Z_{\tau+1})$ by updating $MSE(Z_\tau)$.

  (5.3) Exclude from P_M the pair $p_1 = (i_1, j_1)$ and any other pairs containing region indexes i_1 or j_1 (only if Ω_{i_1} or Ω_{j_1} has more than three pixels).

(6) Compute $\Delta PSNR(\tau + 1)$ using $MSE(Z_{\tau+1})$, and check if $\Delta PSNR(\tau + 1) > \Delta PSNR = 0.5\ dB$. If the condition is true, then save $Z_{\tau+1}$ and increase τ as $\tau \leftarrow \tau + 1$, else go to step (3).

value $d_\ell^{(\tau+1)} = \left\lfloor \frac{\phi_\ell^{(\tau+1)}}{m_\ell^{(\tau+1)}} \right\rfloor$. Hence, the change in distortion introduced in the image $Z_{\tau+1}$ is computed as $\Delta D_p = \frac{\rho\left(\Omega_\ell^{(\tau+1)}\right)}{n_r n_c} - \frac{\rho\left(\Omega_i^{(\tau)}\right) + \rho\left(\Omega_j^{(\tau+1)}\right)}{n_r n_c}$, which can be rewritten as:

$$\Delta D_p = \frac{1}{n_r n_c} \left(m_\ell^{(\tau+1)} \left(d_\ell^{(\tau+1)}\right)^2 - m_i^{(\tau)} \left(d_i^{(\tau)}\right)^2 - m_j^{(\tau)} \left(d_j^{(\tau)}\right)^2 \right) + \frac{2}{n_r n_c} \left(d_i^{(\tau)} \phi_i^{(\tau)} + d_j^{(\tau)} \phi_j^{(\tau)} - d_\ell^{(\tau+1)} \phi_\ell^{(\tau+1)} \right). \tag{4.6}$$

Finally, we can evaluate in the RD plot the pair $p = (i^{(\tau)}, j^{(\tau)})$ by computing the slope, λ_p, of the line that connects the points corresponding to the images Z_τ and $Z_{\tau+1}$. We define the slope λ_p using the corresponding increase in distortion,

ΔD_p, and the estimated decrease in bitrate, ΔR_p, as:

$$\lambda_p = \tan \alpha_p = \frac{\Delta D_p}{\Delta R_p}. \tag{4.7}$$

The pair of regions to merge, $(\Omega_{i^*}^{(\tau)}, \Omega_{j^*}^{(\tau)})$, is chosen from the set of all possible pairs of neighboring regions, by greedily minimizing, in RD, the slope between the corresponding points of Z_τ and $Z_{\tau+1}$, i.e. by choosing the pair $p^* = (i^{*,(\tau)}, j^{*,(\tau)})$ for which the minimum slope λ^* is obtained. However, image $Z_{\tau+1}$ is generated after a number of pairs are merged and the variation of distortion computed in Peak Signal-to-Noise Ratio (PSNR), denoted $\Delta PSNR(\tau+1) = PSNR(Z_{\tau+1}) - PSNR(Z_\tau)$, reaches a threshold set by the user or a specific threshold, i.e. $\Delta PSNR(\tau+1) > \Delta PSNR$. In [P4] we used $\Delta PSNR = 0.5\ dB$.

The concept of contour smoothing from [P2], described in the previous section, is used in GSO to generate images with very low distortion. This time the MSE value is used while taking the decision to generate the next image. In [P4] we used a vector of empirically MSE selected values, like [1, 5, 30, 60, 90, 500, 1000, 2000, ...].

From the implementation point of view, the evaluation of all the possible pairs to merge has a high computational cost. One way of reducing it is to do the evaluation at the beginning of the process, and then update the pairs that contain one of the regions selected to be merged. However, in [P4] we choose to avoid making updates after each merging step. The λ_p values, for all possible pairs region, are sorted increasingly in the vector $\boldsymbol{\lambda} = [\lambda_{p_1}, \lambda_{p_2}, \ldots]$, which is then truncated to the first n_M values $\boldsymbol{\lambda} = [\lambda_{p_1}, \lambda_{p_2}, \ldots, \lambda_{n_M}]$, having the corresponding list of pairs, $P_M = [(i_1, j_1), (i_2, j_2), \ldots, (i_{n_M}, j_{n_M})]$. In the list P_M, we start with the first pair $p_1 = (i_1, j_1)$ and do the following: **(i)** merge the corresponding regions in Z_τ, **(ii)** exclude $p_1 = (i_1, j_1)$ from P_M, **(iii)** exclude from P_M any other pair of regions that contains in the pair the region indexes i_1 or j_1. The process stops when the list P_M is empty. Additionally, we 'bend the rule' at step (iii) and do not apply it for the smaller regions, having three or less pixels. The value n_M is set as follows: if Z_τ contains more than $n_\Omega^{(\tau)} = 1000$ regions, then $n_M = 100$; if Z_τ contains $n_\Omega^{(\tau)} \in [10,\ 1000]$, then $n_M = \lfloor n_\Omega/10 \rfloor$; if Z_τ contains less than 10 regions, then $n_M = 1$. A summary of the GSOm algorithm is presented in Algorithm 4.2.

4.3.2 GSO with region splitting

GSOs is designed to generate images at low bitrate. Its input image is a template image, $Y_0 = Z_{templ}$, which contains the most important contours in Z_0. GSOs generates the sequence of images $\mathcal{Y}_n = \{Y_1, Y_2, \ldots, Y_n\}$, where the regions, from the current image Y_τ, are split using horizontal and vertical straight lines to obtain the regions of the next image $Y_{\tau+1}$. At the decoder, the contour is drawn using the contour of the template image Y_0 and the additional information encoded for each region. The information contains: the decisions to split or not a current region and, for each split, the orientation (vertical or horizontal) and the location of the line segment (relative to the region's bounding box).

GSOs associates a target slope λ_τ to each image Y_τ in the sequence $\mathcal{Y}_n$. The target slope $\lambda_{\tau+1}$ is used to obtain the image $Y_{\tau+1}$ by managing the split process inside each region from the image Y_τ. In the current image Y_τ, the regions are further split only if the target slope $\lambda_{\tau+1}$ allows it, i.e. a region is split if the slope that evaluates the split is larger than $\lambda_{\tau+1}$. The decisions to split a region or not forms a ternary tree, which is traversed in a Depth-First (DF) order. GSOs is searching to split a region $\Omega_\ell^{(\tau)}$ into two regions either with a vertical line segment before column J or a horizontal line segment before line I. The region's bounding box is defining the range of each position, e.g. for $\Omega_\ell^{(\tau)} = \{(x_i, y_i)\}_{i=1,2,\ldots,n}$ we have $x_m \leq I < x_M$, $y_m \leq J < y_M$, where $x_m = \min_i x_i$, $x_M = \max_i x_i$, $y_m = \min_i y_i$, $y_M = \max_i y_i$, $i = 1, 2, \ldots, n$. The horizontal line I is splitting $\Omega_\ell^{(\tau)}$ into $\Omega_{\ell_1}^{I,(\tau)}$ and $\Omega_{\ell_2}^{I,(\tau)}$, and the vertical line J is splitting $\Omega_\ell^{(\tau)}$ into $\Omega_{\ell_1}^{J,(\tau)}$ and $\Omega_{\ell_2}^{J,(\tau)}$.

Similarly as in GSOm, each possible split of the region $\Omega_\ell^{(\tau)}$ is evaluated in turn. The splitting has also two consequences. Firstly, it is introducing for the pixels in the region $\Omega_\ell^{(\tau)}$ a change (decrease) in distortion, which is denoted: **(i)** $\Delta D_{\ell,h}(I)$ for the horizontal split and is computed as $\Delta D_{\ell,h}^{(\tau)}(I) = \rho(\Omega_\ell^{(\tau)}) - \rho(\Omega_{\ell_1}^{I,(\tau)}) - \rho(\Omega_{\ell_2}^{I,(\tau)})$; **(ii)** $\Delta D_{\ell,v}^{(\tau)}(J)$ for the vertical split and is computed as $\Delta D_{\ell,v}^{(\tau)}(J) = \rho(\Omega_\ell^{(\tau)}) - \rho(\Omega_{\ell_1}^{J,(\tau)}) - \rho(\Omega_{\ell_2}^{J,(\tau)})$. Secondly, the split is introducing a change (increase) in bitrate, which is denoted: **(i)** $\Delta R_{\ell,h}^{I,(\tau)}$ for the horizontal split and is estimated as $\Delta R_{\ell,h}^{I,(\tau)} = C_3 + \log_2(x_M - x_m)$; **(ii)** $\Delta R_{\ell,v}^{J,(\tau)}$ for the vertical split and is estimated as $\Delta R_{\ell,v}^{J,(\tau)} = C_3 + \log_2(y_M - y_m)$, where $C_3 = 8$ *bits* includes the cost of encoding the additional depth value after the split and the codelength of the decision.

The split is evaluated using the slope computed as $\lambda_{\ell,h}^{(\tau)}(I) = \frac{\Delta D_{\ell,h}^{(\tau)}(I)}{\Delta R_{\ell,h}^{I,(\tau)}}$ or $\lambda_{\ell,v}^{(\tau)}(J) = \frac{\Delta D_{\ell,v}^{(\tau)}(I)}{\Delta R_{\ell,v}^{J,(\tau)}}$. GSOs selects for each $\Omega_\ell^{(\tau)}$ the optimal split by searching the maximum slope from the set of slopes of all possible splits:

$$\Psi_\ell^{(\tau)} = \left\{ \frac{\Delta D_{\ell,h}^{(\tau)}(I)}{\Delta R_{\ell,h}^{I,(\tau)}} \right\}_{x_m \leq I < x_M} \bigcup \left\{ \frac{\Delta D_{\ell,v}^{(\tau)}(J)}{\Delta R_{\ell,v}^{J,(\tau)}} \right\}_{y_m \leq J < y_M}. \tag{4.8}$$

Let $\lambda_\ell^{*,(\tau)}$ be the optimum slope, then the split decision is taken by checking

$$\lambda_\ell^{*,(\tau)} \geq \lambda_{\tau+1}. \tag{4.9}$$

If the truth value of (4.9) is true, then the region $\Omega_\ell^{(\tau)}$ is *not split*, and the value $\xi_\ell^{(\tau)} = 1$ is encoded. If the truth value of (4.9) is false, then the region $\Omega_\ell^{(\tau)}$ is split into two regions; if $\Omega_\ell^{(\tau)}$ is split vertically, then the value $\xi_\ell^{(\tau)} = 2$ is encoded; if $\Omega_\ell^{(\tau)}$ is split horizontally, then the value $\xi_\ell^{(\tau)} = 3$ is encoded. The split decisions, $\xi_\ell^{(\tau)}$, are collected in a vector $\boldsymbol{\xi}^{(\tau)}$ for all the regions in Z_τ and all the sub-regions obtained by the decision to split. The vector $\boldsymbol{\xi}^{(\tau)}$ is encoded using an order two AMM with Laplace estimator. Besides the split decision, GSOs encodes additional information about the introduced straight lines. If $\xi_\ell^{(\tau)} = 2$, then the row index

Algorithm 4.3: **Greedy Slope Optimization with region splitting (GSOs) - Split a region into two regions (GSOs-SplitInto2)**

A recursive algorithm that splits the region $\Omega_\ell^{(\tau)}$ into two regions using the target slope $\lambda_{\tau+1}$, and encodes auxiliary contour information.

(0) Determine the bounding box of $\Omega_{\ell_1}^{(\tau)}$ by finding x_m, x_M, y_m and y_M.

(1) Estimate the additional rate introduced by an horizontal split, $\Delta R_{\ell,h}^{I,(\tau)} = C_3 + \log_2(x_M - x_m)$, and by a vertical split, $\Delta R_{\ell,v}^{J,(\tau)} = C_3 + \log_2(y_M - y_m)$.

(2) For each line index $I = x_m, x_m + 1, \ldots, x_M - 1$:

(2.1) Split $\Omega_\ell^{(\tau)}$ into $\Omega_{\ell_1}^{I,(\tau)}$ and $\Omega_{\ell_2}^{I,(\tau)}$, using a straight line between rows $I, I+1$.

(2.2) Compute the change in distortion $\Delta D_{\ell,h}^{(\tau)}(I) = \rho(\Omega_\ell^{(\tau)}) - \rho(\Omega_{\ell_1}^{I,(\tau)}) - \rho(\Omega_{\ell_2}^{I,(\tau)})$.

(2.3) Compute the slope $\lambda_{\ell,h}^{(\tau)}(I) = \frac{\Delta D_{\ell,h}^{(\tau)}(I)}{\Delta R_{\ell,h}^{I,(\tau)}}$.

(3) For each column index $J = y_m, y_m + 1, \ldots, y_M - 1$:

(3.1) Split $\Omega_\ell^{(\tau)}$ into $\Omega_{\ell_1}^{J,(\tau)}$ and $\Omega_{\ell_2}^{J,(\tau)}$, using a straight line between columns $J, J+1$.

(3.2) Compute the change in distortion: $\Delta D_{\ell,v}^{(\tau)}(J) = \rho(\Omega_\ell^{(\tau)}) - \rho(\Omega_{\ell_1}^{J,(\tau)}) - \rho(\Omega_{\ell_2}^{J,(\tau)})$.

(3.3) Compute the slope: $\lambda_{\ell,v}^{(\tau)}(J) = \frac{\Delta D_{\ell,v}^{(\tau)}(J)}{\Delta R_{\ell,v}^{J,(\tau)}}$.

(4) Find $\lambda_\ell^{*,(\tau)}$ as the maximum slope in the set $\Psi_\ell^{(\tau)}$ obtained by (4.9).

(5) Set the split decision as follows:

(5.1) If $\lambda_\ell^{*,(\tau)} \leq \lambda_{k+1}$, then set $\xi_\ell^{(\tau)} = 1$;

(5.2) If $\lambda_\ell^{*,(\tau)} > \lambda_{k+1}$, and $\lambda_\ell^{*,(\tau)} \in \left\{\lambda_{\ell,h}^{(\tau)}(J)\right\}_{x_m \leq I < x_M}$, then

(5.2.1) Set $\xi_\ell^{(\tau)} = 2$ and encode I^* using $\log_2(x_{max} - x_{min})$ bits.

(5.2.2) Apply the GSOs-SplitInto2 algorithm for the region $\Omega_{\ell_1}^{I,(\tau)}$.

(5.2.3) Apply the GSOs-SplitInto2 algorithm for the region $\Omega_{\ell_2}^{I,(\tau)}$.

(5.3) If $\lambda_\ell^{*,(\tau)} > \lambda_{k+1}$, and $\lambda_\ell^{*,(\tau)} \in \left\{\lambda_{\ell,v}^{(\tau)}(J)\right\}_{y_m \leq J < y_M}$, then

(5.3.1) Set $\xi_\ell^{(\tau)} = 3$ and compress J^* using $\log_2(y_M - y_m)$ bits.

(5.3.2) Apply the GSOs-SplitInto2 algorithm for the region $\Omega_{\ell_1}^{J,(\tau)}$.

(5.3.3) Apply the GSOs-SplitInto2 algorithm for the region $\Omega_{\ell_2}^{J,(\tau)}$.

I^* corresponding to the optimal split slope $\lambda_\ell^{*,(\tau)}$ is encoded using $\log_2(x_M - x_m)$ bits, and $\Omega_\ell^{(\tau)}$ is split into two regions having the same column indices and the following row indices: $[x_m, I^*]$ for $\Omega_{\ell_1}^{I,(\tau)}$, and $[I^*+1, x_M]$ for $\Omega_{\ell_2}^{I,(\tau)}$. If $\xi_\ell^{(\tau)} = 3$, the column index J^* corresponding to the optimal split slope $\lambda_\ell^{*,(\tau)}$ is encoded using $\log_2(y_M - y_m)$ bits, and $\Omega_\ell^{(\tau)}$ is split into two regions having the same row indices, and the following column indices $[y_m, J^*]$ for the region $\Omega_{\ell_1}^{J,(\tau)}$, and $[J^* + 1, y_M]$ for the region $\Omega_{\ell_2}^{J,(\tau)}$.

Algorithm 4.4: **Greedy Slope Optimization with region splitting (GSOs)**

The algorithm generates the sequence $\mathcal{Y}_n = \{Y_1, Y_2, \ldots, Y_n\}$ *using the template image* Y_0 *and a corresponding set of increasing target slopes* $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$.

(1) For $\tau = 0, 1, \ldots, n-1$, use the current image Y_τ and the input target slope $\lambda_{\tau+1}$ to generate the next image in the sequence, $Y_{\tau+1}$, as follows:

(1.1) Smooth contour by changing the contour edges separating pixels with similar values as the majority of their neighbors: if at least 3 neighbors in $\mathcal{N}_4(x, y)$ have the same depth value, v, and if $|Y(x, y) - v| < \gamma$, then set $Y(x, y) = v$.

(1.2) Find the constant regions in the image Y_τ, and represent Y_τ using the set of $n_\Omega^{(\tau)}$ constant regions $\Omega(Y_\tau)$, the corresponding set of depth values $\mathcal{D}(Y_\tau)$, and the contour $\Gamma(Y_\tau)$.

(1.3) For each region $\Omega_\ell^{(\tau)}$ from Y_τ, $\ell = 1, 2, \ldots n_\Omega^{(\tau)}$

(1.3.1) Apply the GSOs-SplitInto2 recursive algorithm (see Algorithm 4.3) for the region $\Omega_\ell^{(\tau)}$ and the target slope $\lambda_{\tau+1}$.

As small implementation aspect we can mention that, to decrease the distortion, GSOs sets the depth value $d_\ell^{(\tau)} - 1$ to the pixels of each first line and first column of $\Omega_\ell^{(\tau)}$, if the neighbor region $\Omega_i^{(\tau)}$ has $d_i^{(\tau)} < d_\ell^{(\tau)}$, and similarly the depth value $d_\ell^{(\tau)} + 1$ is set to the pixels of each first line and first column of $\Omega_\ell^{(\tau)}$, if $d_i^{(\tau)} > d_\ell^{(\tau)}$. Also, the contour in Y_0 is smoothed as in GSOm, this time by constraining the absolute difference to be smaller than a threshold γ, where γ is increased until it reaches the value 100.

GSOs is summarized in Algorithm 4.4, where Algorithm 4.3 presents the recursive algorithm, denoted GSOs - Split a region Into two regions (GSOs-SplitInto2).

4.3.3 Entropy coding the GSO sequences

GSO was designed to generate sequences of lossy images from an initial image Z_0. The coding of any image in the two sequences, $\mathcal{Z}_n$ and $\mathcal{Y}_n$, can be done using CERV, APC or any other encoder.

In [P4], two algorithms are introduced by combining the contour compression algorithm from [P2] and the region reconstruction algorithm from [P3] (Alg. Y). The algorithm that encodes the sequence $\mathcal{Z}_n$ is denoted Chain-Code–Value (CCV), and the algorithm that encodes the sequence $\mathcal{Y}_n$ is denoted Chain-Code-Line–Value (CCLV).

The main concept in CCLV is that the value $d_\ell^{(\tau)}$, of the current region $\Omega_\ell^{(\tau)}$, is encoded using a list of neighbors containing also the up and left neighboring regions, because the continuous splitting with horizontal and vertical lines does not guarantee that two neighboring regions have different depth values (i.e. $d_\ell^{(\tau)} \notin D_\ell$). Hence, in Alg. Y (see Algorithm 3.3 in Section 3.3.2), the depth value $d_\ell^{(\tau)}$ is now encoded using the list $\mathcal{L}_\ell$, instead of the list L_ℓ.

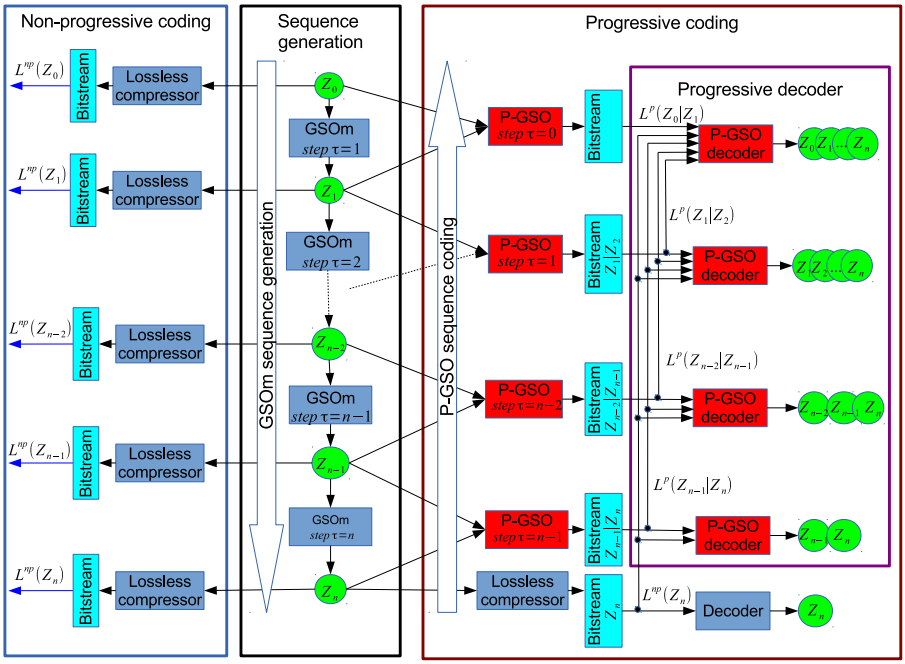


Figure 4.1: Non-progressive and progressive coding and decoding of the sequence $\mathcal{Z}_n$ generated from Z_0. *(Middle panel)*: Generation of the sequence $\mathcal{Z}_n$ using GSOm; at each step $\tau = 1, 2, \ldots, n$ a lossy image Z_τ is generated. *(Left panel)*: Non-progressive coding of $\mathcal{Z}^p$ using a lossless compressor. *(Right panel)*: Lossy-to-lossless progressive coding of $\mathcal{Z}_p$; at each step $\tau = n-1, n-2, \ldots, 0$ a lossy image Z_τ is encoded conditionally to the previous encoded image $Z_{\tau+1}$. The red rectangles are representing P-GSO and the blue rectangles the other methods. Green circles are representing an uncompressed image. Cyan rectangles are representing bitstreams, out of which the lossy images can be decoded.

4.4 Progressive coding of GSO sequences

Publication [P6] introduces an algorithm that encodes progressively the entire sequence of images generated by GSOm, $\mathcal{Z}_n = \{Z_1, Z_2, \ldots, Z_n\}$, that is called Progressive coding of GSO sequences (P-GSO).

In the previous section we presented the way GSOm is generating $Z_{\tau+1}$ from Z_τ by merging several pairs of regions from Z_τ, where the next pair to be merged is selected by finding the minimum slope from (4.7). Each image Z_τ from the sequence $\mathcal{Z}_n$ was encoded using a lossless compression algorithm in $\mathcal{L}^{np}(Z_\tau)$ bits.

In [P6], P-GSO is creating a bitstream from which the sequence $\mathcal{Z}_n$, or any subsequence of it, can be generated in the reversed order, $\mathcal{Z}^p = \{Z_n, Z_{n-1}, \ldots, Z_0\}$. If the bitstream is truncated, P-GSO is still able to generate the first part of the

sequence until the information is lost. A non-progressive coding method is encoding independently each image Z_τ, while the progressive coding of Z_τ is created from the prefixes of the bitstream, corresponding to the sequence of images from Z_n until $Z_{\tau+1}$ (see Figure 4.1). Therefore, the progressive encoding of Z_τ is less efficient than the non-progressive coding, and the progressive codelength $\mathcal{L}^p(Z_\tau)$ is larger than the non-progressive codelength $\mathcal{L}^{np}(Z_\tau)$. P-GSO is designed to encode each image $Z_{\tau+1}$, relative to the previous image Z_τ, in such a way that the 'price' paid for the progressive functionality, $\mathcal{L}^p(Z_\tau) - \mathcal{L}^{np}(Z_\tau)$, is as small as possible.

On one hand, at each step τ from 1 to n, GSOm has removed from Z_τ the set of $n_{\Delta\Gamma}^{(\tau)}$ contour segments $\Delta\Gamma^{(\tau)} = \{\Gamma_k\}_{k=1,2,\ldots,n_{\Delta\Gamma}^{(\tau)}}$ to obtain $Z_{\tau+1}$, which now is having a lower number of regions than Z_τ. From Section 4.3.1 we summarize here that after we choose to merge the pair of regions $p = (i^{(\tau)}, j^{(\tau)})$, a new region $\Omega_\ell^{(\tau+1)} = \Omega_i^{(\tau)} \cup \Omega_j^{(\tau)}$ is obtained, having the associated depth value $d_\ell^{(\tau+1)} = \left\lfloor \frac{\sum_{(x,y)\in\Omega_\ell^{(\tau+1)}} Z_0(x,y)}{m_\ell^{(\tau+1)}} \right\rfloor$, where $m_\ell^{(\tau+1)}$ is the number of pixels in the region $\Omega_\ell^{(\tau+1)}$.

On the other hand, at each step τ from $n-1$ to 0, P-GSO is generating each image Z_τ using the known image $Z_{\tau+1}$ and auxiliary information (encoded with the lowest possible codelength) to obtain Z_τ. Hence, P-GSO is encoding $n_{\Delta\Gamma}^{(\tau)}$ contour segments, $\Delta\Gamma^{(\tau)}$, that are added to the contours of $Z_{\tau+1}$, in order to obtain the contours of Z_τ. In the reconstruction stage, the depth values $d_i^{(\tau)}$ and $d_j^{(\tau)}$ are reconstructed using $d_\ell^{(\tau+1)}$. Although at the beginning of the process only a few regions (in the order of tens) are split in Z_n, at the end of the process P-GSO typically gets to split several thousand regions.

4.4.1 Progressive coding of contours

At each iteration τ, the contour segments from $\Delta\Gamma^{(\tau)}$ are added to the already encoded contours $\Gamma^{(\tau+1)}$, of the last known image $Z_{\tau+1}$ (see Figure 4.2), to obtain the contours of the next image in the sequence, Z_τ, i.e. $\Gamma^{(\tau)} = \Delta\Gamma^{(\tau)} \cup \Gamma^{(\tau+1)}$. The contour segments $\Delta\Gamma^{(\tau)}$ are encoded using a vertex representation, i.e. Γ_k is represented by an anchor point, P_1, a direction point, P_2, and a chain-code vector S_k (see Section 3.2.1). The same anchor point classification as in [P5] is used also in [P6]: *double*, *relative*, and *edge*. However, P-GSO has a progressive functionality, and, at step τ, the contour $\Gamma^{(\tau+1)}$ is used as a priori information by anchor points search procedures, which are different than the ones in [P5].

As we described in Section 3.2.1, the first step, in the coding of the contour using the vertex representation, is to find the anchor points. The *relative* anchor points are found by traversing the contour segments $\Gamma^{(\tau+1)}$, and by marking, in a binary vector $\Psi^{(\tau)}$, the vertices that have more adjacent vertices in $\Gamma^{(\tau)}$ than in $\Gamma^{(\tau+1)}$. The found vertices may have a minimum of two adjacent vertices part of the contour $\Gamma^{(\tau+1)}$, and a maximum of two adjacent vertices part of the contour $\Gamma^{(\tau)}$, which are possible direction points. Next we analyse the possible cases that appear for a found vertex, say P_k. **(i)** If four adjacent vertices are part of $\Gamma^{(\tau+1)}$, then P_k can not be an anchor point, since there is no other possibility to start a

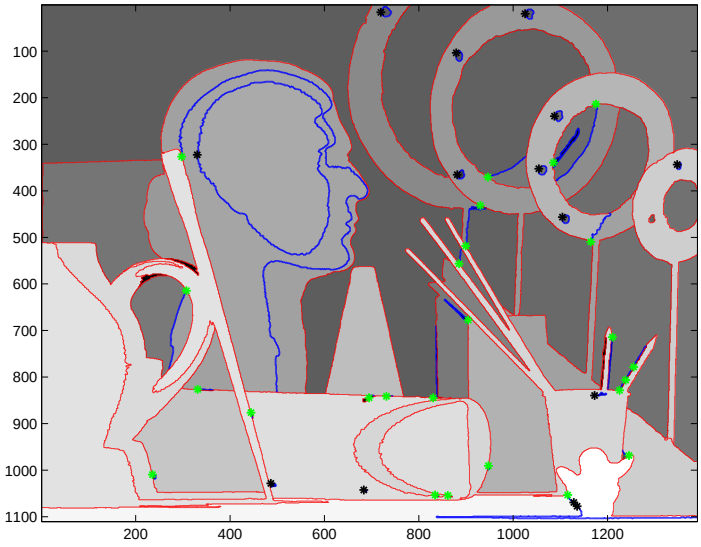


Figure 4.2: Contour coding of Z_τ at step τ. Contour map of last known image, $Z_{\tau+1}$, is represented by contour edges marked with red. Contour segments $\Delta\Gamma^{(\tau)}$, encoded at step τ, are represented by contour edges marked with blue. Double anchor points are marked with black; relative anchor points are marked with green.

contour segment in $\Gamma^{(\tau)}$. **(ii)** If three adjacent vertices are part of $\Gamma^{(\tau+1)}$, then it is possible that P_k is an anchor point and the remaining adjacent vertex is the direction point. However, this case is very rare and may appear when the image has a high contour density. **(iii)** If two adjacent vertices are part of $\Gamma^{(\tau+1)}$ and P_k is set as anchor point, then there are two options for the direction point. Therefore, a binary switch, ξ_k, is used to select the correct position. The binary switches are collected in a vector $\Xi^{(\tau)}$. Let us denote $n_\xi^{(\tau)}$ the number of relative anchor points, and $n_\xi^{(\tau)}$ the number of direction points found for $\Delta\Gamma^{(\tau)}$.

The column-wise search is finding the *edge* and *double* anchor points, which are then marked in a vector $\Upsilon^{(\tau)}$ by checking column-wise the contour graph of Z_τ. There is no need to encode a direction point for these two types of anchor points as we have already shown in Section 3.2.1.

The contour segments from $\Delta\Gamma^{(\tau)}$ are encoded using two functions. One is called Integer Value Encoder (IVE), which was developed to encode efficiently one integer value that belongs to a predictable range. The other function is called Golomb-Rice Encoding of Vectors (GR-EV) and was developed to encode a vector of integer values using a modified version of the GR algorithm. The first stage of the P-GSO algorithm is the progressive coding of the image contours, which is summarized here in Algorithm 4.5. See Figure 4.3 for examples of progressive reconstructions for the depth-map image *Art*. Note that on step (6) of the algorithm, before encoding $S^{(\tau)}$, a deterministic change is done (see Appendix A); also the IVE function was involved in the coding of the optimal context tree.

Algorithm 4.5: **Progressive coding of GSO sequences (P-GSO) - one loop step in contour compression stage**

Algorithm for progressive coding of the contour $\Gamma^{(\tau)}$ *of the image* Z_τ *using the contour* $\Gamma^{(\tau+1)}$ *of the image* $Z_{\tau+1}$.

(1) Search for the relative anchor points by traversing contour $\Gamma^{(\tau)}$, and then mark in the list $\Psi^{(\tau)}$ the anchor point vertices found by analyzing each contour vertex. Find for each Γ_k its direction point, ξ_k, and vector of 3OT symbols, S_k^T.

(2) Search for the edge and double anchor points using a column-wise search, and mark their positions in $\Upsilon^{(\tau)}$. Find for each Γ_k its vector of 3OT symbols, S_k^T.

(3) Encode the number of relative anchor points $v_1 = n_\xi^{(\tau)}$, and the number of edge and double anchor points $v_2 = n_{\Delta\Gamma}^{(\tau)} - n_\xi^{(\tau)}$, by applying the *IVE* function as $IVE(v_i, 4, 2),\ i = 1, 2$.

(4) Encode the anchor points by first obtaining a vector Φ, which saves the number of consecutive values 0, and then by applying the GR-EV function to Φ. Apply this procedure first to $\Psi^{(\tau)}$, and then to the vectorized matrix $\Upsilon^{(\tau)}$.

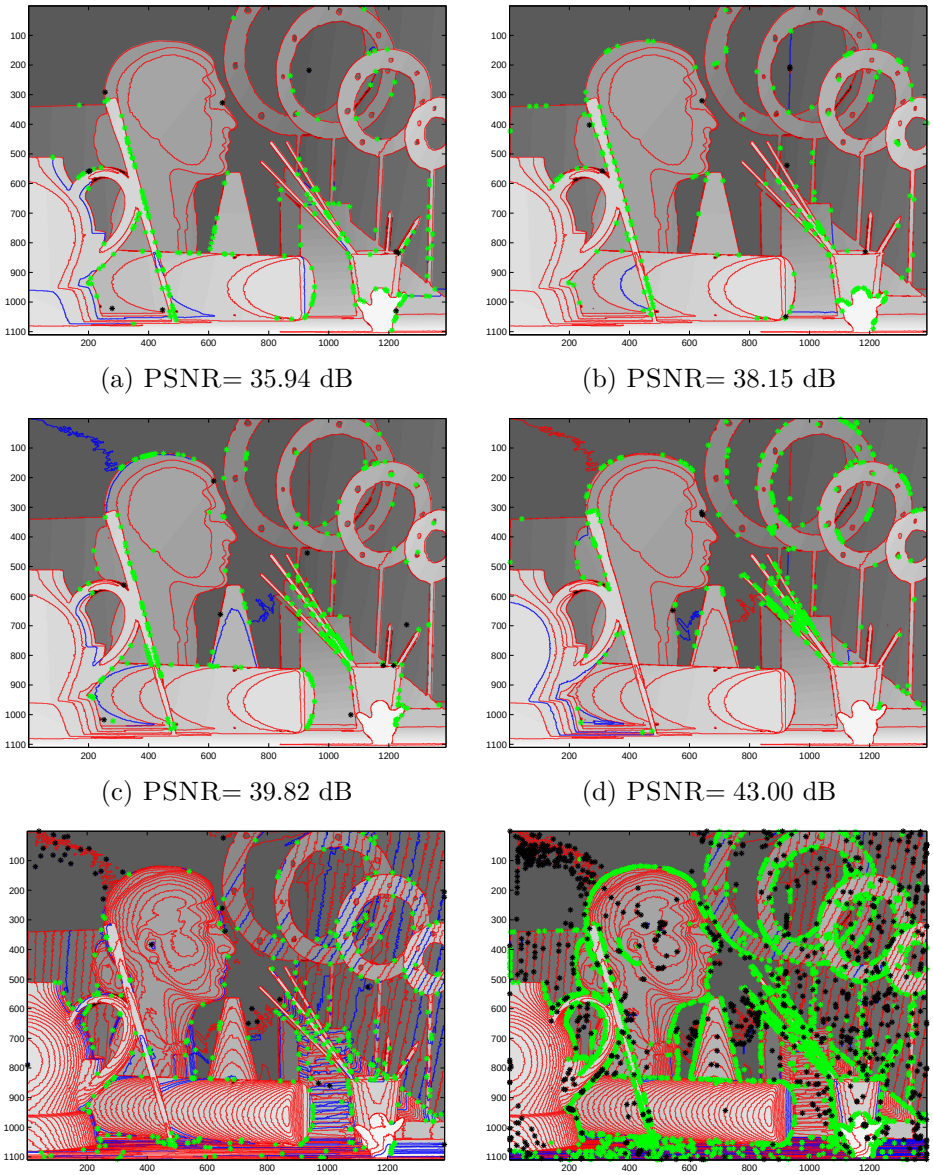
(5) Write directly in the output file the vector of direction points, $\Xi^{(\tau)}$, on $n_\xi^{(\tau)}$ bits.

(6) Encode the vector of concatenated 3OT vectors, $S^{(\tau)} = \left[S_1^T\ S_2^T\ \cdots\ S_{n_{\Delta\Gamma}^{(\tau)}}^T\right]^T$, using CTM with Laplace estimator, and with a maximum tree depth of $d_{\mathcal{T}} = 17$.

Integer Value Encoder, IVE(v, nb_δ, nb_r)

The IVE function was developed to encode an integer $v \geq 0$ by first finding a smaller interval in which it belongs, and then by encoding integer v more efficient, using the found interval limits. The process of designing the set of intervals is the most important part of the algorithm.

Using the parameters nb_δ and nb_r, the IVE function is generating four intervals, in which v is searched, as follows. The first interval is set as $[0,\ 2^{nb_\delta}]$ and, if it is selected, integer v is encoded using $\log_2 2^{nb_\delta} = nb_\delta$ bits. The following intervals are obtained by shifting incrementally, with nb_r, the number of bits needed to represent the upper bound of the previous interval. The second interval is $[2^{nb_\delta},\ 2^{nb_\delta}(1 + 2^{nb_r})]$ and $\log_2 2^{nb_\delta+nb_r} = nb_\delta + nb_r$ bits are needed to encode now $v' = v - 2^{nb_\delta}$ (we subtract from v the lower limit of the interval to scale the integer to the found interval). The third interval is $[2^{nb_\delta}(1 + 2^{nb_r}),\ 2^{nb_\delta}(1 + 2^{nb_r}(1 + 2^{nb_r}))]$ and $\log_2 2^{nb_\delta+2nb_r} = nb_\delta + 2nb_r$ bits are needed to encode $v' = v - 2^{nb_\delta}(1 + 2^{nb_r}(1 + 2^{nb_r}))$. The last interval is reserved for encoding high values, i.e. for $v \geq 2^{nb_\delta}(1+2^{nb_r}(1+2^{nb_r}))$, where we encode the value $v' = v-2^{nb_\delta}(1+2^{nb_r}(1+2^{nb_r}))$ on $\log_2 2^{nb_\delta+3nb_r+Q} = nb_\delta + 3nb_r + Q$ bits, where Q is a positive integer constant. Since v is an integer number, IVE does not cover all possible cases. However, for the values v encoded we can easily predict their upper bound and usually this last interval is not selected.



(a) PSNR= 35.94 dB (b) PSNR= 38.15 dB

(c) PSNR= 39.82 dB (d) PSNR= 43.00 dB

(e) PSNR= 73.02 dB (f) PSNR= 90.47 dB

Figure 4.3: Examples of progressive reconstructions of depth-map images having different distortion. The contour map of the last known image is represented by contour edges marked with red. The contour segments encoded at the current step are represented by contour edges marked with blue. Double anchor points are marked with black; relative anchor points are marked with green.

Golomb-Rice Encoding of Vectors, GR-EV(Φ)

The GR-EV function distinguishes two cases when coding the vector Φ, depending on its number of elements, n_Φ. A switch is used to select one of the two cases. In the first case, the GR algorithm is applied to Φ with the parameter $M = 2^{k^*}$, by first finding the best parameter k^* and then encoding the vector Φ using M.

In the second case, the input vector is divided into two sub-vectors, denoted Φ_L and Φ_H, then for each element from Φ we need to encode a decision specifying if the current value is added to Φ_L or to Φ_H. The sub-vector Φ_L collects the elements from Φ smaller than M, while the sub-vector Φ_H collects the remaining elements. The two sub-vectors are each encoded by the GR algorithm applied to Φ_L and Φ_H with the parameters $M_L = 2^{k_L^*}$ and $M_H = 2^{k_H^*}$, respectively. The variables k^*, $k^* - k_L^*$ and $k_H^* - k^*$ are each encoded by AMM with Laplace estimator.

4.4.2 Progressive region reconstruction

The second stage of P-GSO is the progressive coding of the parameters used for regions reconstruction, where each region of Z_τ is reconstructed at the decoder using the previously known image in $\mathcal{Z}^p$, $Z_{\tau+1}$, and using the information regarding the contour of Z_τ encoded in the first stage (so the regions of the image Z_τ are currently available). Below we analyze the algorithm, which is summarized in the Algorithm 4.6.

Let us first analyze how a region, $\Omega_\ell^{(\tau+1)}$, from $Z_{\tau+1}$ is split in Z_τ. GSOm merged not only two regions from Z_τ to form $Z_{\tau+1}$, but enough regions until the decision to save $Z_{\tau+1}$ is taken. The region $\Omega_\ell^{(\tau+1)}$ is usually split, in Z_τ, not only into two regions, but in (let us say) $n_\ell^{(\tau)}$ regions denoted $\{\Omega_\ell^{j,(\tau)}\}_{j=1,2,\ldots,n_\ell^{(\tau)}}$. Since GSOm is generating both $Z_{\tau+1}$ and Z_τ, then:

(a) Each region $\Omega_\ell^{j,(\tau)}$ is formed of $m_\ell^{j,(\tau)}$ pixels, $\Omega_\ell^{j,(\tau)} = \{(x_i, y_i)\}_{i=1,2,\ldots,m_\ell^{j,(\tau)}}$, having the depth values $z_i = Z_0(x_i, y_i)$ in the original depth-map image Z_0.

(b) Let us denote the *real value* of the *average* inside a region $\Omega_\ell^{j,(\tau)}$ as $\breve{d}_\ell^{j,(\tau)} = \frac{1}{m_\ell^{j,(\tau)}} \sum_{i=1}^{m_\ell^{j,(\tau)}} z_i = \frac{\phi_\ell^{j,(\tau)}}{m_\ell^{j,(\tau)}}$, and its *truncated integer average* value as $d_\ell^{j,(\tau)} = \left\lfloor \breve{d}_\ell^{j,(\tau)} \right\rceil = \breve{d}_\ell^{j,(\tau)} - \Delta_\ell^{j,(\tau)}$, where $\Delta_\ell^{j,(\tau)} \in [-0.5, 0.5]$ is the rounding error.

Similarly, the region $\Omega_\ell^{(\tau+1)}$ is defined as having: $m_\ell^{(\tau+1)}$ pixels, the real-value average $\breve{d}_\ell^{(\tau+1)} = \frac{1}{m_\ell^{(\tau+1)}} \sum_{i=1}^{m_\ell^{(\tau+1)}} z_i$, and the truncated average $d_\ell^{(\tau+1)} = \left\lfloor \breve{d}_\ell^{(\tau+1)} \right\rceil = \breve{d}_\ell^{(\tau+1)} - \Delta_\ell^{(\tau+1)}$, where $\Delta_\ell^{(\tau+1)} \in [-0.5,\ 0.5]$ is the rounding error. From the split of the region $\Omega_\ell^{(\tau+1)}$ into the set of regions $\{\Omega_\ell^{j,(\tau)}\}_{j=1,2,\ldots,n_\ell^{(\tau)}}$, it results the following equation between the number of pixels of the regions:

$$m_\ell^{(\tau+1)} = \sum_{j=1}^{n_\ell^{(\tau)}} m_\ell^{j,(\tau)}. \tag{4.10}$$

Since the contour $\Gamma^{(\tau)}$ is already known at the decoder, only the set of integer depth values $\{d_\ell^{j,(\tau)}\}_{j=1,2,\ldots,n_\ell^{(\tau)}}$ for each $\Omega_\ell^{(\tau+1)}$ are needed in the reconstruction

Algorithm 4.6: **Progressive coding of GSO sequences (P-GSO) - one loop step in region reconstruction stage**

The algorithm is coding progressively the parameters required to reconstruct Z_τ, using the contour $\Gamma^{(\tau)}$ and a priori image $Z_{\tau+1}$.

(0) Set as empty current vectors $A^{(\tau)}$ and $B^{(\tau)}$.
(1) Find the set of region splits using $\Gamma^{(\tau)}$ and $\Gamma^{(\tau+1)}$, i.e. find the correspondences between each $\Omega_\ell^{(\tau+1)}$ from $Z_{\tau+1}$ and a set of regions, $\{\Omega_\ell^{j,(\tau)}\}_{j=1,2,\ldots,n_\ell^{(\tau)}}$, from Z_τ.
(2) For every region index $\ell = 1, 2, \ldots, n_\Omega^{(\tau+1)}$ do:
  (2.1) Find the index j for which (4.14) is true and relabel the region as $n_\ell^{(\tau)}$.
  (2.2) Compute the differences $b_\ell^{j,(\tau)}$ using (4.11), and append them at the end of the vector $B^{(\tau)}$.
  (2.3) Compute the difference $a_\ell^{(\tau)}$ using (4.13) and (4.15), and append it to $A^{(\tau)}$.
(3) Encode $A^{(\tau)}$ and $B^{(\tau)}$ as described at ***(Label 1)*** and ***(Label 2)*** labels.

stage of the image Z_τ. The algorithm encodes each set of integer depth values $\{d_\ell^{j,(\tau)}\}_{j=1,2,\ldots,n_\ell^{(\tau)}}$ as follows: the first $n_\ell^{(\tau)} - 1$ truncated averages of the set, i.e. $\{d_\ell^{j,(\tau)}\}_{j=1,2,\ldots,n_\ell^{(\tau)}-1}$, are encoded relative to the truncated average depth $d_\ell^{(\tau)}$; and the $n_\ell^{(\tau)}$-th value is predicted from the encoded averages $\{d_\ell^{j,(\tau)}\}_{j=1,2,\ldots,n_\ell^{(\tau)}-1}$ and the available truncated average $d_\ell^{(\tau+1)}$.

The algorithm assumes that the values d_ℓ^j are scattered around the truncated average d_ℓ, since GSOm selected their regions to be merged together into Ω_ℓ. Hence, the first $n_\ell^{(\tau)} - 1$ truncated averages, $\{d_\ell^{j,(\tau)}\}_{j=1,2,\ldots,n_\ell^{(\tau)}-1}$, are encoded by the differences

$$b_\ell^{j,(\tau)} = d_\ell^{(\tau)} - d_\ell^{j,(\tau)},\ j = 1, 2, \ldots, n_\ell^{(\tau)} - 1,\ \ell = 1, 2, \ldots, n_\Omega^{(\tau+1)}. \tag{4.11}$$

(Label 1) Let us collect all $b_\ell^{j,(\tau)}$ differences, obtained at the current step τ, in the vector $B^{(\tau)} = \left[b_1\ b_2\ \cdots\ b_{n_B^{(\tau)}}\right]^T$, where $n_B^{(\tau)}$ is the difference between the number of regions in Z_τ and in $Z_{\tau+1}$. Each vector $B^{(\tau)}$ is encoded as follows: for each element $b_j,\ j = 1, 2, \ldots, n_B^{(\tau)}$, we encode its sign, using one bit, and its absolute value, $|b_j|$, using the AMM with Laplace estimator applied for ns_B symbols. The value ns_B is found as follows: **(i)** find the maximum absolute difference, $b_M = \max_{j=1,2,\ldots,n_B^{(\tau)}} |b_j|$; **(ii)** compute the number of bits needed to represent it, $n_M = \lceil log_2(b_M) \rceil$; **(iii)** set $ns_B = 2^{n_M}$. Variable ns_B can be transmitted to the decoder by encoding n_M using $\log_2(7)$ bits. Sometimes is better to estimate the codelength of $B^{(\tau)}$ using $ns_B = b_M$, and check if by encoding an additional $n_M - 1$ bits the result is improved (see next section).

P-GSO encodes the truncated average $d_\ell^{n_\ell^{(\tau)},(\tau)}$ by predicting it using the already encoded truncated averages $\{d_\ell^{j,(\tau)}\}_{j=1,2,\ldots,n_\ell^{(\tau)}-1}$. Since the split of $\Omega_\ell^{(\tau+1)}$ does

not change the initial values of the pixels in Z_0, using real value averages we obtain:

$$\underbrace{\left(\sum_{j=1}^{n_\ell^{(\tau)}} m_\ell^{j,(\tau)}\right)}_{m_\ell^{(\tau+1)}} \breve{d}_\ell^{j,(\tau+1)} = \sum_{j=1}^{n_\ell^{(\tau)}} m_\ell^{j,(\tau)} \breve{d}_\ell^{j,(\tau)}. \tag{4.12}$$

On the other hand, using the truncated averages we obtain

$$\left(\sum_{j=1}^{n_\ell^{(\tau)}} m_\ell^{j,(\tau)}\right) d_\ell^{(\tau+1)} + \left(\sum_{j=1}^{n_\ell^{(\tau)}} m_\ell^{j,(\tau)}\right) \Delta_\ell^{(\tau+1)} = \sum_{j=1}^{n_\ell^{(\tau)}} m_\ell^{j,(\tau)} d_\ell^{j,(\tau)} + \sum_{j=1}^{n_\ell^{(\tau)}} m_\ell^{j,(\tau)} \Delta_\ell^{j,(\tau)},$$

and if we separate in the left term the unknown truncated average $d_\ell^{n_\ell^{(\tau)},(\tau)}$, then:

$$d_\ell^{n_\ell^{(\tau)},(\tau)} = \underbrace{d_\ell^{(\tau+1)} + \sum_{j=1}^{n_\ell^{(\tau)}-1} \frac{m_\ell^{j,(\tau)}}{m_\ell^{n_\ell^{(\tau)},(\tau)}} \left(d_\ell^{(\tau+1)} - d_\ell^{j,(\tau)}\right)}_{\hat{d}_\ell^{n_\ell^{(\tau)},(\tau)}} + \underbrace{\sum_{j=1}^{n_\ell^{(\tau)}} \frac{m_\ell^{j,(\tau)}}{m_\ell^{n_\ell^{(\tau)},(\tau)}} \left(\Delta_\ell^{(\tau)} - \Delta_\ell^{j,(\tau)}\right)}_{\varepsilon_\ell^{n_\ell^{(\tau)},(\tau)}}, \tag{4.13}$$

where we denoted $\hat{d}_\ell^{n_\ell^{(\tau)},(\tau)}$, the estimation of truncated average $d_\ell^{n_\ell^{(\tau)},(\tau)}$ computed by the decoder; and $\varepsilon_\ell^{n_\ell^{(\tau)},(\tau)}$, the rounding residual which depends on the coefficients $\left\{\frac{m_\ell^{j,(\tau)}}{m_\ell^{n_\ell^{(\tau)},(\tau)}}\right\}_{j=1,2,\ldots,n_\ell^{(\tau)}-1}$. We obtain the minimum for $\left\{\frac{m_\ell^{j,(\tau)}}{m_\ell^{n_\ell^{(\tau)},(\tau)}}\right\}_{j=1,2,\ldots,n_\ell^{(\tau)}-1}$ by selecting $\Omega_\ell^{n_\ell^{(\tau)},(\tau)}$ as the region with the highest number of pixels among all the regions in $\left\{\Omega_\ell^{j,(\tau)}\right\}_{j=1,2,\ldots,n_\ell^{(\tau)}}$, and hence we manage to minimize $\varepsilon_\ell^{n_\ell^{(\tau)},(\tau)}$ so that

$$\frac{m_\ell^{j,(\tau)}}{m_\ell^{n_\ell^{(\tau)},(\tau)}} < 1, \ \forall j = 1, 2, \ldots, n_\ell^{(\tau)} - 1. \tag{4.14}$$

Since Z_τ stores integer values, the prediction error must be also rounded as $\hat{d}_\ell^{n_\ell^{(\tau)},(\tau)} = \left\lfloor \hat{d}_\ell^{n_\ell^{(\tau)},(\tau)} \right\rceil - \hat{\Delta}_\ell^{n_\ell^{(\tau)},(\tau)}$, where $\hat{\Delta}_\ell^{n_\ell^{(\tau)},(\tau)}$ is the rounding error.

Finally, the depth value $d_\ell^{n_\ell^{(\tau)},(\tau)}$ is encoded by the difference:

$$a_\ell^{(\tau)} = d_\ell^{n_\ell^{(\tau)},(\tau)} - \left\lfloor \hat{d}_\ell^{n_\ell^{(\tau)},(\tau)} \right\rceil = \varepsilon_\ell^{n_\ell^{(\tau)},(\tau)} + \hat{\Delta}_\ell^{n_\ell^{(\tau)},(\tau)}. \tag{4.15}$$

Similarly, we collect all values $a_\ell^{(\tau)}, \ell = 1, 2, \ldots, n_\Omega^{(\tau+1)}$, in a vector $A^{(\tau)}$. Hence, at iteration τ, we encode $A^{(\tau)} = \left[a_1 \ a_2 \ \cdots \ a_{n_A^{(\tau)}}\right]^T$, where $n_A^{(\tau)}$ is the number of regions that were split in $Z_{\tau+1}$ to obtain Z_τ.

(Label 2) The alphabet for the elements of the vectors $A^{(\tau)}$ was found to be $\mathcal{A}_A = \{-2, -1, 0, 1, 2\}$. The P-GSO algorithm introduced a vector of possible symbols,

$$\Lambda = [-1 \ 0 \ 1 \ -2 \ 2]^T = [\lambda_1 \ \lambda_2 \ \lambda_3 \ \lambda_4 \ \lambda_5]^T, \tag{4.16}$$

and finds, for each element a_ℓ of $A^{(\tau)}$, the index k_ℓ for which $\lambda_{k_\ell} = a_\ell$. Hence, the procedure first finds the maximum index $k_M = \max_{\ell=1,2,\ldots,n_A^{(\tau)}} k_\ell$, encodes it using a codelength of $\log_2(5)$ bits, and then encodes each index k_ℓ (which corresponds to the element a_ℓ from $A^{(\tau)}$) using the AMM with Laplace estimator with an alphabet of k_M symbols.

4.5 Parameterizations of planar models

In all the developed algorithms described until now we used for the region reconstruction stage one depth value to parameterize the reconstruction model. All the pixels in the region are set with the truncated average over the region. This model is known as the piecewise constant model or, simply, *constant model*. However, also other models can be fitted over the region, and if the segmentation divides the image into regions containing objects, a more complex model is more efficient than the constant model. Nevertheless, the gain obtained by the decrease in distortion is 'payed' with the drawback of coding (for the complex model) more real-value parameters compared to one integer-value parameter in the constant model case. To solve this problem, in [P7], we introduce several parameterizations for the planar model that finds a compromise between the entropy coding of the plane parameters and the introduced distortion.

Section 2.4.4 shown that for a region $\Omega_\ell = \{(x_i, y_i)\}_{i=1,2,\ldots,n}$, formed of n pixels and having the depth values in the initial image Z_0 denoted $z_i = Z_0(x_i, y_i),\ i = 1, 2, \ldots, n$, the planar model parameters can be estimated by the LS algorithm, which computes the best fitting plane with the plane parameters $\underline{\theta}^* = [a^*\ b^*\ c^*]^T$. The distortion introduced by the plane estimated by $\underline{\theta}^*$ is given by (2.13). In [P7], we choose to parameterize the planar model using the heights of three selected pixels in the $\underline{\theta}^*$ plane (after truncating them to integers). In this way the entropy coding algorithm receives as input tree integer values instead of three floating point values, and because integer values are encoded with a lower codelength than real values the bitrate is reduced. The drawback of this parameterization is that the distortion introduced in the region, computed by (2.13), is higher than the value $MSE_p^{(LS)}$, since the new plane will only *approximate* the optimal plane $\underline{\theta}^*$. The introduced parameterization is discussed in the next subsection. In [P7], seven different methods were developed and tested to find the positions of the three heights, which are described in Section 4.5.2. The entropy coding of the three parameters used to estimate the planar model is described in Section 4.5.3, while in Section 4.5.4 we extend GSOm by introducing a competition between the constant model and the planar model.

4.5.1 Planar model parameterization using three heights

Let us choose three pixel positions, denoted (using their locations in the region) as $A = (x_\alpha, y_\alpha)$, $B = (x_\beta, y_\beta)$ and $C = (x_\gamma, y_\gamma)$, having the values z_α, z_β and z_γ in the initial depth-map image Z_0. The selection of the three points A, B and C,

for any given region Ω_ℓ, is done by a method that knows only the segmentation, i.e. the positions of the pixels and the shape of every region Ω_ℓ (generated by the segmentation), so that it can be applied at both encoder and decoder, after the contour compression stage.

The heights of these three points A, B, C, in optimal plane $\underline{\theta}^*$, are the real numbers z^*_α, z^*_β, z^*_γ, which are given by (2.10), for $i = \alpha, \beta, \gamma$. Since we choose to encode integer valued parameters, the three heights are rounded to $\hat{z}_\alpha$, $\hat{z}_\beta$, $\hat{z}_\gamma$, which are the three integer parameters encoded and used for region reconstruction. This is the reason why the method is called the Three Heights parameterization (3H). Let us now stack the corresponding equations from (2.11), written for $i = \alpha, \beta, \gamma$, in a matrix form as follows:

$$\begin{bmatrix} \hat{z}_\alpha \\ \hat{z}_\beta \\ \hat{z}_\gamma \end{bmatrix} = \begin{bmatrix} z^*_\alpha \\ z^*_\beta \\ z^*_\gamma \end{bmatrix} - \begin{bmatrix} \Delta^*_\alpha \\ \Delta^*_\beta \\ \Delta^*_\gamma \end{bmatrix} = \begin{bmatrix} z^*_\alpha - \Delta^*_\alpha \\ z^*_\beta - \Delta^*_\beta \\ z^*_\gamma - \Delta^*_\gamma \end{bmatrix} = \underbrace{\begin{bmatrix} x_\alpha & y_\alpha & 1 \\ x_\beta & y_\beta & 1 \\ x_\gamma & y_\gamma & 1 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} a^* \\ b^* \\ c^* \end{bmatrix}}_{\underline{\theta}^*} - \underbrace{\begin{bmatrix} \Delta^*_\alpha \\ \Delta^*_\beta \\ \Delta^*_\gamma \end{bmatrix}}_{\underline{\Delta}}, \quad (4.17)$$

where the matrix Q stacks the coordinates of the selected locations A, B, C, which are defining a triangle $\triangle ABC$, and the vector $\underline{\Delta}$ stacks the rounding errors in these three locations.

The reconstruction plane $\tilde{\underline{\theta}} = \begin{bmatrix} \tilde{a} & \tilde{b} & \tilde{c} \end{bmatrix}^T$, that estimates plane $\underline{\theta}^*$ using 3H, is computed using the three parameters $\hat{z}_\alpha$, $\hat{z}_\beta$, $\hat{z}_\gamma$, and their pixel location given by the selected method. Decoder computes the reconstruction plane, $\tilde{\underline{\theta}}$, by solving:

$$\underbrace{\begin{bmatrix} x_\alpha & y_\alpha & 1 \\ x_\beta & y_\beta & 1 \\ x_\gamma & y_\gamma & 1 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} \tilde{a} \\ \tilde{b} \\ \tilde{c} \end{bmatrix}}_{\tilde{\underline{\theta}}} = Q\tilde{\underline{\theta}} = \begin{bmatrix} \hat{z}_\alpha \\ \hat{z}_\beta \\ \hat{z}_\gamma \end{bmatrix}. \quad (4.18)$$

In general, $\underline{\theta}^*$ is different from $\tilde{\underline{\theta}}$, but the experiments show that, using the selection of the three pixel positions A, B, C, our methods found them reasonably close.

In the region reconstruction stage at the decoder, the reconstructed value of the pixel location (x_i, y_i) is denoted $\breve{z}_i$, and is computed by rounding $\tilde{z}_i$, the corresponding height of the pixel in the reconstruction plane $\tilde{\underline{\theta}}$, where

$$\breve{z}_i = \lfloor \tilde{z}_i \rceil = \tilde{z}_i - \tilde{\Delta}_i = [x_i \ y_i \ 1] \ \tilde{\underline{\theta}} - \tilde{\Delta}_i, \ i = 1, 2, \ldots, n, \quad (4.19)$$

where $\tilde{\Delta}_i \in [-0.5, \ 0.5]$ are the rounding errors.

In conclusion, the 3H method uses three heights in the LS plane to compute the reconstruction plane $\tilde{\underline{\theta}}$, which is an estimation of the optimal plane $\underline{\theta}^*$. The distortion introduced by the 3H method for planar model in the region Ω_ℓ is

$$MSE_p^{(3H)} = \frac{1}{n} \sum_{i=1}^{n} (z_i - \breve{z}_i)^2. \quad (4.20)$$

4.5.2 Selecting A, B, C for MSE optimization

In 3H, the selection of the three pixel positions A, B, C has an important impact for both distortion and bitrate. The distortion computed in MSE for the 3H method, $MSE_p^{(3H)}$, is closer to the distortion for the LS method, $MSE_p^{(LS)}$, if $\underline{\tilde{\theta}}$ approximates much better $\underline{\theta}^*$.

Heuristically, it is easy to imagine that, when the A, B, C points are as far apart as possible one from another, the rounding error in A, B, C will have a smaller influence in (4.17) and in the reconstruction plane $\underline{\tilde{\theta}}$, taking it much closer to the ideal plane $\underline{\theta}^*$. The disadvantage of this selection is that the codelength associated to the plane parameters is increasing a lot, since the values z_α^*, z_β^*, z_γ^* are increasing in magnitude when the distances between A, B and C increase. The seven methods, presented below, are either selecting the three pixel positions A, B, C inside the region Ω_ℓ or outside the region Ω_ℓ (but close to the regions contour), and in such a way that the triangle formed out of the three points, $\triangle ABC$, includes Ω_ℓ.

Methods for selecting A, B, C inside Ω_ℓ

Let us denote the matrix $Q_{ABC} = \begin{bmatrix} x_\alpha & y_\alpha & 1 \\ x_\beta & y_\beta & 1 \\ x_\gamma & y_\gamma & 1 \end{bmatrix}$, using three indexes $i = \alpha, \beta, \gamma$, which correspond to the position of three pixel positions $A = (x_\alpha, y_\alpha)$, $B = (x_\beta, y_\beta)$ and $C = (x_\gamma, y_\gamma)$, which are defining the constrained triangle $\triangle ABC$.

METHOD 1 (M_1)

M_1 is minimizing the expected MSE. From the MSE excess introduced by 3H, $MSE_p^{(3H)} - MSE_p^{(LS)}$, we isolate one component (MSE excess) and denote it MSE_e. The criterion of interest for the method is defined as the expected value of MSE_e, i.e. $\mathrm{E}[MSE_e]$.

Let us now rewrite (4.20) by highlighting this time Q, the matrix that we have to search for. First, we evaluate the difference between the two sets of parameters, $\underline{\theta}^*$ and $\underline{\tilde{\theta}}$, using (4.17) and (4.18) as follows:

$$Q\underline{\theta}^* = Q\underline{\tilde{\theta}} + \underline{\Delta} \iff \underline{\theta}^* - \underline{\tilde{\theta}} = Q^{-1}\underline{\Delta}. \tag{4.21}$$

Next we rewrite the difference $z_i - \breve{z}_i$ using (2.10), (2.11), and (4.19) as:

$$\begin{aligned} z_i - \breve{z}_i &= (z_i - \hat{z}_i) + (\tilde{\Delta}_i - \Delta_i^*) + [x_i \ y_i \ 1](\underline{\theta}^* - \underline{\tilde{\theta}}) \\ &\overset{(4.21)}{=} (z_i - \hat{z}_i) + (\tilde{\Delta}_i - \Delta_i^*) + [x_i \ y_i \ 1]Q^{-1}\underline{\Delta}. \end{aligned}$$

Finally, we rewrite (4.20) using the square of $z_i - \breve{z}_i$, and by neglecting the cross-terms, since we assume that the cross-correlations between the modeling errors and the rounding errors are negligible. Hence, equation (4.20) is rewritten as:

$$MSE_p^{(3H)} \approx MSE_p^{(LS)} + \frac{1}{n}\sum_{i=1}^{n}(\tilde{\Delta}_i - \Delta_i^*)^2 + \underbrace{\underline{\Delta}^T Q^{-T} R Q^{-1}\underline{\Delta}}_{MSE_e},$$

where $R = \frac{1}{n}\sum_{i=1}^{n}[x_i\ y_i\ 1]^T[x_i\ y_i\ 1]$.

If we assume that the rounding errors collected by $\underline{\Delta}$ are behaving like independent variables, having the distribution $\mathcal{U}(-0.5, 0.5)$ when the LS plane is taking all possible positions, then we obtain the expression of the criterion of interest by taking the mean of MSE_e, denoted $\mathrm{E}[MSE_e]$, which is obtained as:

$$\mathrm{E}[MSE_e] = \mathrm{E}\left[\underline{\Delta}^T Q^{-T} R Q^{-1} \underline{\Delta}\right] = \frac{1}{12}\mathrm{tr}\, Q^{-T} R Q^{-1}. \tag{4.22}$$

Finally, we can set the three pixel locations A, B and C as the solution of the following optimization problem:

$$\min_{A,B,C\in\Omega_\ell} \mathrm{tr}\ Q_{ABC}^{-T} R Q_{ABC}^{-1}. \tag{4.23}$$

METHOD 2 (M_2)

The second method is based on the heuristic idea that, because the three pixel positions A, B and C are used to compute the reconstruction plane, the triangle $\triangle ABC$ must cover as much area as possible from the optimal LS plane. However, we add also the constraint that the three points are inside Ω_ℓ, so that the algorithm is forced to encode heights which are closer to the depth values found in Ω_ℓ. Therefore, the pixel positions A, B, C, located as far as possible one from another inside Ω_ℓ, should form the triangle $\triangle ABC$ with the maximum area. The value of the triangle area of $\triangle ABC$ is given by $|\det Q_{ABC}|$. Hence, M_2 is setting the three pixel locations A, B and C as the solution of the following optimization problem:

$$\max_{A,B,C\in\Omega_\ell} Area(\triangle ABC) = \max_{A,B,C\in\Omega_\ell} |\det Q_{ABC}|. \tag{4.24}$$

In both methods, M_1 and M_2, the matrix Q_{ABC} is found by going through all triplets of points in the region Ω_ℓ. To reduce the complexity, we set first the pixel locations A and B based on the idea that the two points must be placed as far apart as possible. We first set $A = (x_\alpha, y_\alpha)$ as the first pixel position in Ω_ℓ, found in the first column of the first row in region mask, i.e. $x_\alpha = \min_{i=1,2,\ldots,n}\{x_i\}$, $y_\alpha = \min_{i=1,2,\ldots,n}\{y_i | x_i = x_\alpha\}$. The second pixel location $B = (x_\beta, y_\beta) = (x_{k_2}, y_{k_2})$ is selected as the pixel position found at the maximum distance from the already selected position (x_α, y_α). The index position k_2 is found as $k_2 = \arg\max_{i=1,2,\ldots,n}\{(x_\alpha - x_i)^2 + (y_\alpha - y_i)^2\}$. Hence, the search in both methods, M_1 and M_2, is done only for the remaining unknown positions $C \in \Omega_\ell$.

Methods setting A, B, C using bounding box of Ω_ℓ

This type of methods is based on the idea that triangle $\triangle ABC$ should select an area of the bounding box for the region Ω_ℓ. Let us first denote the following coordinates: $x_m = \min_i x_i$, $x_M = \max_i x_i$, $y_m = \min_i y_i$, $y_M = \max_i y_i$, $i = 1, 2, \ldots, n$. These four coordinates define four points which are forming a rectangle that includes Ω_ℓ, which is called the *bounding box* of Ω_ℓ. Let us denote also $\delta_x = x_M - x_m$, $\delta_y = y_M - y_m$, $\bar{x} = x_m + \frac{\delta_x}{2}$, $\bar{y} = y_m + \frac{\delta_y}{2}$.

METHOD 3 (M_3)

This method sets $\triangle ABC$ as a triangle that has the maximum area inside the bounding box of Ω_ℓ. The triangle can be formed by selecting any side of the bounding box as the triangle's base and any point on the opposite side. However, we decided to search for the triangle by first finding the larger side of the bounding box, i.e. by checking $\delta_x > \delta_y$, and then selecting two points as the ends of one of the larger sides and the third point at the middle of the other larger side of the bounding box. Hence, we select $A = (x_M, y_m)$, and if $\delta_x > \delta_y$, then $B = (x_m, y_m)$ and $C = (\bar{x}, y_M)$, else $B = (x_M, y_M)$ and $C = (x_m, \bar{y})$.

METHOD 4 (M_4)

This method sets the triangle $\triangle ABC$ as the smallest triangle that includes the bounding box of Ω_ℓ. Hence, the three pixel positions are selected as follows: $A = (x_m - \delta_x, y_m)$, $B = (x_M, y_m)$, and $C = (x_M, y_M + \delta_y)$.

Methods with a different parameterization

This type of methods are all selecting the same three pixel locations $A = (x_M, y_m)$, $B = (x_m, y_M)$, and $C = (x_m, y_m)$, but are using different entropy coding methods for the parameters. In 3H, three rounded heights $\hat{z}_\alpha$, $\hat{z}_\beta$, and $\hat{z}_\gamma$ were selected as parameters. For this type of methods we developed a modified version of 3H, where we select as parameters the following values: one rounded height, $\hat{z}_\gamma$, and two rounded height differences, $\hat{z}_\alpha - \hat{z}_\gamma$ and $\hat{z}_\beta - \hat{z}_\gamma$. This new method of parameterization is denoted here One Height and two height Differences parameterization (1H2D). For the selected pixel locations, the differences can be written using (4.17), at the encoder, and using (4.18), at the decoder, to obtain:

$$\hat{z}_\alpha - \hat{z}_\gamma = \lfloor z^*_\alpha \rceil - \lfloor z^*_\gamma \rceil = \delta_x a^* - (\Delta^*_\alpha - \Delta^*_\gamma) \quad = \quad \delta_x \tilde{a} \tag{4.25}$$

$$\hat{z}_\beta - \hat{z}_\gamma = \underbrace{\lfloor z^*_\beta \rceil - \lfloor z^*_\gamma \rceil = \delta_y b^* - (\Delta^*_\beta - \Delta^*_\gamma)}_{\text{encoder}} \quad = \quad \underbrace{\delta_y \tilde{b}.}_{\text{decoder}} \tag{4.26}$$

The second idea used for these methods is that the parameters are encoded with a higher precision because the new selected parameters are more sensitive to the rounding errors Δ^*_α, Δ^*_β and Δ^*_γ. Hence, the parameters are rounded at the N^{th} bit of the binary representation of their fractional value. The parameter $\hat{z}_\gamma$ is transmitted, in the both methods described below, using a 1-bit extra-precision after the decimal point, by encoding the value $2\hat{z}_\gamma = \lfloor 2z^*_\gamma \rceil$. The sign of each of the parameters $\hat{z}_\alpha - \hat{z}_\gamma$ and $\hat{z}_\beta - \hat{z}_\gamma$ is written in the output file on one bit.

METHOD 5 (M_5)

The two height differences are encoded using their sign and their absolute values $|\hat{z}_\alpha - \hat{z}_\gamma|$ and $|\hat{z}_\beta - \hat{z}_\gamma|$. The absolute values are transmitted to the decoder, using a 1-bit extra-precision after the decimal point, by encoding using GR the values $|2(\hat{z}_\alpha - \hat{z}_\gamma)| = ||\lfloor 2z^*_\alpha \rceil - \lfloor 2z^*_\gamma \rceil|$ and $|2(\hat{z}_\beta - \hat{z}_\gamma)| = ||\lfloor 2z^*_\beta \rceil - \lfloor 2z^*_\gamma \rceil|$.

METHOD 6 (M_6)

M_6 is encoding the values $|\hat{z}_\alpha - \hat{z}_\gamma|$ and $|\hat{z}_\beta - \hat{z}_\gamma|$ using the first $N_6 = 10$ bits of the following sub-unitary values: $||\lfloor 2^{-9} z^*_\alpha \rceil - \lfloor 2^{-9} z^*_\gamma \rceil||$ and $||\lfloor 2^{-9} z^*_\beta \rceil - \lfloor 2^{-9} z^*_\gamma \rceil||$. The first k^*_{LP} bits are used to obtain a decimal value, which is encoded using the AMM with Laplace estimator, while the remaining $N_6 - k^*_{LP}$ bits are written directly in the output file. Note that the optimal values k^*_{LP}, for a^* and b^*, are first searched, encoded (each) on $\log_2(10)$ bits, and then used in the coding procedure.

Baseline method for encoding $\underline{\theta}^*$

We compare all the methods presented above with a baseline method which encodes the LS parameters $\underline{\theta}^*$ with an improved precision after the decimal point.

METHOD 7 (M_7)

First two parameters a^* and b^* are transmitted using $N_7 = 8$ bits extra-precision after the decimal point, by encoding using GR, this time, the values $||\lfloor 2^{N_7} a^* \rceil||$ and $||\lfloor 2^{N_7} b^* \rceil||$. The parameter signs are written on one bit in the output file. The parameter c^* is transmitted using 1-bit extra-precision after the decimal point by encoding $\lfloor 2c^* \rceil$.

4.5.3 Entropy coding the parameters

We assume that the planar model parameters in 3H, i.e. the three heights, are scattered around the truncated average d_ℓ, the constant model parameter, the reason why each height is encoded relative to d_ℓ. Hence, 3H can encode its parameters in a separate bitstream, and the method can be considered as an improvement over the results obtained by a coder based on the constant model. The parameters are encoded by Algorithm D (Alg. D) using the differences between each height and d_ℓ. Hence, for the methods M_1:M_4 we encode the differences $d_\ell - \hat{z}_\alpha$, $d_\ell - \hat{z}_\beta$ and $d_\ell - \hat{z}_\gamma$, by collecting them in a vector V. Let us label each element in V as v_j, $j = 1, 2, \ldots, n_V$, where n_V is the number of elements in V. For methods M_5 and M_6, the vector V contains the differences for the parameter $\hat{z}_\gamma$ computed as $v_j = 2\lfloor d_\ell \rceil - \lfloor 2z^*_\gamma \rceil$. For the method M_7, the parameter c^* is encoded using the difference $v_j = 2\lfloor d_\ell \rceil - \lfloor 2c^* \rceil$.

Algorithm D

The vector of differences, V, is encoded by an efficient algorithm denoted Alg. D. Below we discuss the algorithm, while its summary can be found in Algorithm 4.7. Its main idea is to encode each element v_j, $j = 1, 2, \ldots, n_V$, by its sign, $sgn(v_j)$, written in the output file on one bit, and its absolute value, $|v_i|$, using AMM with Laplace estimator with ns_V symbols.

Alg. D is a complex algorithm designed to encode the differences v_j as best as possible by finding the best value for the parameter ns_V. The first step of the algorithm is to find the maximum absolute difference $v_M = \max_{j=1,2,\ldots,n_V} |v_j|$ in

Algorithm 4.7: **Entropy coding of differences - Algorithm D (Alg. D)**

Algorithm for encoding an input vector V of differences.

(0) Do one pass through V, find the elements $|v_j| \geq 2^7$, modify their values, and set the corresponding escape mode.

(1) Find $v_M = \max_{j=1,2,\ldots,n_V} |v_j|$ and $n_M = \lceil \log_2(v_M) \rceil$. Compute $\bar{v}_M$ using (4.28). Estimate $\mathcal{L}(\bar{v}_M)$.

(2) Encode n_M using $\log_2(7)$. Check inequality (4.27) and set ns_V.

(3) For each element $v_j,\ j = 1, 2, \ldots, n_V$, do:

(3.1) Write $sgn(v_j)$ on one bit directly in the output file, and encode $|v_j|$ using AMM with Laplace estimator with ns_V symbols.

(3.2) If $v_j = v_M$ and it is the first time we find this case, encode the escape mode maximum index C_{em} on $\log_2(3)$ bits and the current escape mode index on $log_2(C_{em})$. Other times when $v_j = v_M$ (and if $C_{em} \neq 0$, i.e. at least one escape mode is used), encode the escape mode index using $log_2(C_{em})$ bits. (If $C_{em} = 2$, then encode also the initial parameter with AMM with Laplace estimator for 2^8 symbols.)

vector V. Next we compute the number of bits needed to represent v_M as $n_M = \lceil \log_2(v_M) \rceil$, and then encode n_M on $\log_2(7)$ bits using the *Arithmetic Coder* for 7 symbols. The first option for setting parameter ns_V is $ns_V = 2^{n_M}$. This means that, using only $\log_2(7)$ bits, we inform the decoder to use the biggest number represented on n_M bits as the length of the alphabet in the Laplace estimator.

However, sometimes the value 2^{n_M} is too large, and the algorithm becomes inefficient. That is why we introduced a second option for setting ns_V, which informs the decoder to use $ns_V = v_M$, by encoding the last $n_M - 1$ bits in the representation of v_M. Now, the only question that remains is when should we use each option? The decision to select an option is taken by checking if we can obtain a decrease in codelength by encoding these $n_M - 1$ more bits. We estimate the decrease in bitrate using the estimated codelength difference between encoding using the parameter $ns_V = 2^{n_M}$ and encoding using the parameter $ns_V = \bar{v}_M$. Hence, the decision is taken using the inequality:

$$n_M - 1 < n_V(\mathcal{L}(2^{n_M}) - \mathcal{L}(\bar{v}_M)) - g, \tag{4.27}$$

where: $g = 3$ bits is the smallest gain accepted, $\mathcal{L}(2^{n_M}) = log_2(2^{n_M}) = n_M\ [bits]$ is the codelength for encoding a symbol using AMM with Laplace estimator for $ns_V = 2^{n_M}$ symbols, $\mathcal{L}(\bar{v}_M) = log_2(\bar{v}_M)$ is the estimated codelength of a symbol encoded using the AMM with Laplace estimator, with $ns_V = \bar{v}_M$, where

$$\bar{v}_M = \frac{(2^{n_M} - 1) + (2^{n_M - 1})}{2} = 1.5 \cdot 2^{n_M - 1} - 0.5 \tag{4.28}$$

is the average of all possible values that the maximum absolute value v_M can take, i.e. $v_M \in [2^{n_M - 1}, 2^{n_M} - 1]$, since n_M bits are needed to represent v_M. Inequality

(4.27) can be rewritten as:

$$n_M < n_V(n_M - \log_2(3 \cdot 2^{n_M-1} - 1) + 1) - 2. \tag{4.29}$$

Alg. D uses also an escape mode procedure. For the value $|v_j| \geq 2^7$ an escape mode is introduced by setting $v_j = ns_V$ and encoding a switch using $\log_2(3)$ bits to signal it. The following cases are used: **(0)** no escape case is used, **(1)** the selected parameter (height) has the values zero, **(2)** the selected parameter is encoded with AMM with Laplace estimator for 2^8 symbols. After the first value $v_i = ns_V$ is encoded, we signal the case by encoding the escape case index 0, 1 or 2 using $\log_2(C_{em})$ bits, where C_{em} is the maximum index of the found escape mode.

4.5.4 Extension of GSOm with plane fitting

In [P7] an algorithm denoted simply Plane Fitting (PF) was developed for the planar model, where M_2 is set as the default method. The PF algorithm was used to extend the GSOm algorithm by selecting not only which region pair to be merged, but also which method of reconstruction to use for that pair. The new algorithm was denoted GSOmPF, and its strategy is to choose, among all possible pairs and among the two models, the pair and model which leads to the best slope, in RD plot, after merging.

Next we discuss the conditions to be checked before choosing the planar model instead of the constant model. For region Ω_ℓ, we first check if

$$MSE_p^{(3H)}(\Omega_\ell) < MSE_c(\Omega_\ell), \tag{4.30}$$

where MSE_c is computed by (2.14) and $MSE_p^{(3H)}$ by (4.22). In the ideal case, when the exact (real) parameters are encoded, the planar model has a better distortion, however, in our case the quantization of the estimated parameters may change the inequality. As a second condition, we estimate the two slopes in the RD plot and then select the smallest one. Let us first consider that currently we have a lossy version of the initial image, that has the distortion MSE_x, and for which we estimated, using the constant model, a bitrate decrease of ΔR with (4.5). The bitrate decrease obtained, using now the planar model, will be $\Delta R - \Delta C$, where ΔC is the difference between the two estimated codelength of the models. Note that the constant model was estimated to be encoded using 8 bits and the planar model using 12 bits, i.e. $\Delta C = 4$ bits. The slope in RD plot for the constant model is $\lambda_c = \frac{MSE_c - MSE_x}{\Delta R}$, while for the planar model is $\lambda_p = \frac{MSE_p^{(3H)} - MSE_x}{\Delta R - \Delta C}$. The second condition to be checked, before selecting the planar model, is $\lambda_p < \lambda_c$ or

$$\frac{MSE_p^{(3H)} - MSE_x}{\Delta R - \Delta C} < \frac{MSE_c - MSE_x}{\Delta R}. \tag{4.31}$$

Chapter 5

Results for Datasets of Depth-Map Images

"Tempus edax rerum." (Time is the devourer of all things.)
— Publius Ovidius Naso *Metamorphoses*

In this chapter, we present the results for the proposed algorithms and compare them with the results of the state of the art algorithms for the available datasets of depth-map images. The first section presents a summary of all the algorithms presented in this dissertation. The second section presents results for the lossless compression of depth-map images, while the third section presents results for the lossy compression.

5.1 Summary of the developed algorithms

All implemented algorithms show good results compared with the state of the art coders, for both lossless and lossy compression of depth-map images. Before describing the results, we would like to present a summary of all the algorithms collected in this dissertation, see Table 5.1 for the three *lossless* compression algorithms, and Table 5.2 for the four *lossy* compression algorithms.

The following datasets were used to test the algorithms in each of the seven publications included in this dissertation:

- *Middlebury* dataset [31, 75], available online [76]. It contains 162 disparity images estimated using various techniques by the Middlebury Stereo Vision group from Middlebury College, Vermont, USA. The images have three different sizes: *(i)* 54 images at full resolution; *(ii)* 54 images at half resolution; *(iii)* 54 images at third resolution ($\frac{1}{3}$ of full resolution).

- *Ballet* and *Breakdancers* sequences [107], available online [56]. They are sequences of depth-map images from Microsoft Corporation, having available

100 images for each of the eight views. In our tests we used only the first view for the comparison with the state of the art articles because the compression results do not change significantly from one view to another.

- *Beer-garden* sequence [35]. A multiview sequence of high resolution depth-map images from Philips. The dataset was used only for lossless compression.
- *Kendo* and *Balloons* sequences [44]. Moving camera test sequences from the Tanimoto Laboratory, Nagoya University, Japan. This datasets were used only for lossy compression.

Table 5.1: Summary of the algorithms developed for lossless compression

Algorithm	**Description**
NCV	*New Complex Version (*NCV*)* algorithm described in [P1]. NCV is our first algorithm developed for lossless compression. It generates a specific segmentation based on the introduced concept of variability in a region, and offers a mixture of predictors for region reconstruction. The segmentation's contour is encoded using a basic algorithm, which represents the contour using a sequence of adjacent vertices, with vertex positions codified by the 3OT representation.
CERV CERV-ALG.C CERV-ALG.Y	*Crack-Edge–Region–Value (*CERV*)* algorithm from [P3]. CERV is a two stage algorithm that compresses the images using an over-segmentation. In the first stage, ALG. C is used to compress the image contours. The contour is represented using its elements called contour edges (or crack-edges), classified as horizontal and vertical. In the second stage, ALG. Y is used to compress the depth value of each region, using the list of known neighboring regions.
APC	*Anchor Points Coding (*APC*)* algorithm from [P5]. APC presents an efficient solution for compressing the contour of an over-segmentation using the concept of contour segments generation, and by encoding vertex positions codified by the 3OT representation. Contour segments generation, vertex analysis, and classification and coding of anchor points are the main contributions of this algorithm. APC is using CERV-ALG.Y for compressing the depth value in each region, to achieve lossless compression.

Table 5.2: Summary of the algorithms developed for lossy compression

Algorithm	Description
L-CRS NL-CRS	*Lossy Constrained Region Segmentation (*L-CRS*)* and its near-lossless version (NL-CRS) are algorithms from [P2]. L-CRS is our first algorithm developed for lossy image segmentation, which deals with the generation of segmentations for depth-map images, using the concept of variability inside a region, in such a way that the region reconstruction is done using the concepts of quantization and coding of almost constant regions, and the uniform quantization of the residuals obtained by the *Martucci* predictor.
GSO GSOm GSOs	*Greedy Slope Optimization (*GSO*)* algorithm from [P4]. GSO is generating a sequence of image segmentations for lossy compression by creating different lossy images of the initial image with a given distortion. The sequence of segmentations is obtained either by merging regions, or by splitting template regions. In the region merging version, GSOm, greedy decisions to merge a pair of neighboring regions are taken at each step, and the segmentations are saved when the distortion reaches a specific level. The splitting regions version, GSOs, is designed to split the regions of a template image, using vertical and horizontal straight lines, until a target slope is reached.
P-GSO	*Progressive coding of* GSOm *sequences (*P-GSO*)* algorithm from [P6]. P-GSO is compressing progressively a sequence (or subsequence) of GSOm lossy images. The contour compression and the region reconstruction algorithm are encoding efficiently the current image using a priori information from the previously reconstructed image. The performance of the P-GSO algorithm is close to that of the non-progressive algorithms.
PF Alg.D GSOmPF	*Plane Fitting (PF)* algorithm from [P7]. PF deals with the parameterization of the planar model using the introduced Three Heights method (3H). Seven different methods are developed and tested in [P7] to find, for any region shape, the optimal positions of the three heights. The algorithm is encoding the three parameters, relative to the constant model parameter, using Algorithm D (Alg. D). An extension of the GSOm algorithm, called GSOmPF, was created using PF to improve the quality of the segmentations.

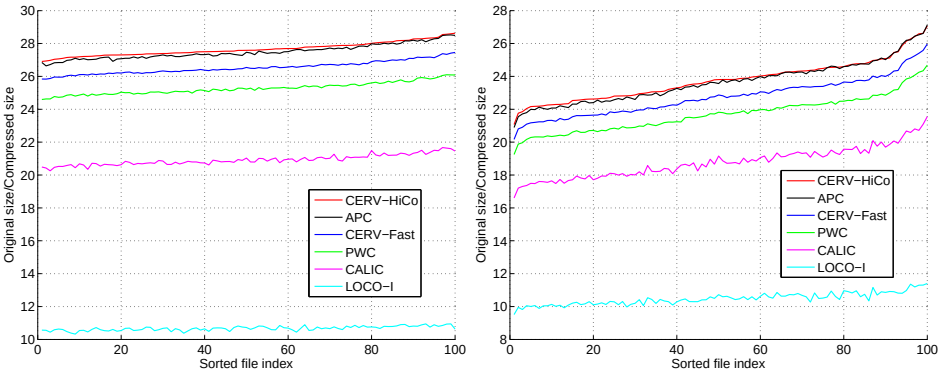


Figure 5.1: Lossless compression results for **(a)** the *Ballet* sequence; **(b)** the *Breakdancers* sequence. A comparison between our algorithms: CERV-HiCo, CERV-Fast and APC, and state of the art lossless coders: PWC, CALIC and LOCO-I.

5.2 Results for lossless compression

The NCV algorithm, published in 2012, offered good results compared to the state of the art coders LOCO-I and Portable Network Graphics (PNG). However, the later developed algorithms, CERV and APC, offered much better results and that is why we chose to only show their results here.

Figures 5.1 and 5.2 show comparative results between the CERV algorithm (with its two versions CERV-HiCo and CERV-Fast), the APC algorithm, and the state of the art coders: CALIC, LOCO-I, and PWC (algorithm specialized in the compression of palette images). Compression results are presented using the *compression ratio* between the file size of the original image over the compressed image, where the best compression is offered by the compressor with the highest compression ratio value. Here we present the results for three datasets, however, more results can be found on CERV's webpage [89] and APC's webpage [79]. First two datasets are *Ballet* and *Breakdancers*, each formed of 100 depth-map images, acquired from a sensor in the first view (camera labeled '0') of a multi-view arrangement. The *Middlebury* dataset is formed of 162 estimated disparity images, divided into three resolutions: full, half, and third. The *Middlebury* disparity images are estimated from a stereo pair and structural light additional images.

In Figure 5.1, one can see that all our algorithms have better results than the state of the art coders. Although the CERV-Fast algorithm does not use the optimal context tree in the contour coding stage, its results are close to CERV-HiCo. Comparing our two algorithms CERV-HiCo and APC, one can notice that there is not much difference in their results, and only that CERV-HiCo has a small advantage for these two datasets.

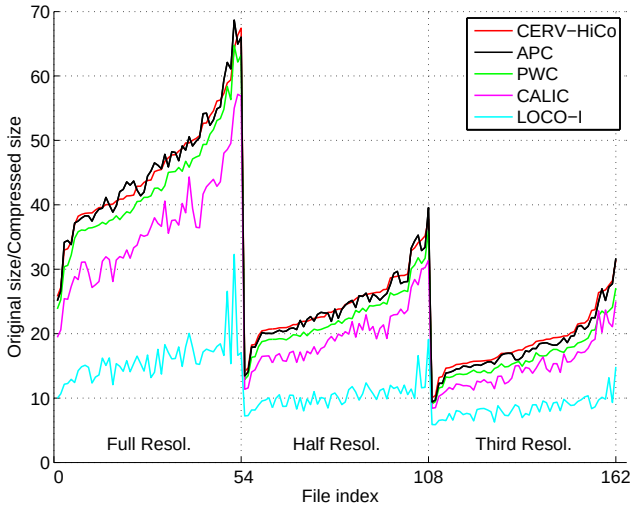


Figure 5.2: Lossless compression results for the *Middlebury* dataset. A comparison between our algorithms: CERV-HiCo, CERV-Fast, and APC, and state of the art lossless coders: PWC, CALIC and LOCO-I.

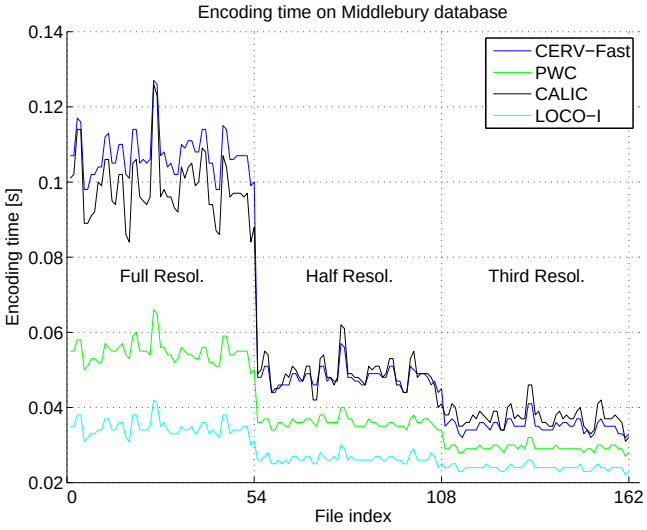
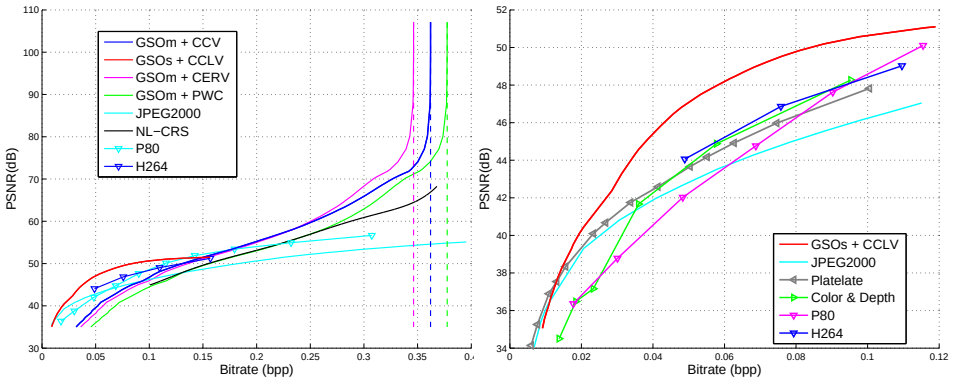


Figure 5.3: Encoding time of four coders over the *Middlebury* dataset.

In Figure 5.2, we tested the algorithms for *Middlebury* disparity images. This time one can notice that the APC algorithm has sometimes a small advantage for the full resolution, while for third resolution the advantage goes again to the CERV-HiCo algorithm. In comparison with other three state of the art coders, our algorithms are in the same positions as previously, by having better results.



(a) (b)

Figure 5.4: Lossy compression results for the frame 0, first view (cam0), from the *Breakdancers* sequence: **(a)** from low bitrate to high bitrate; the lossless compression results for three entropy coders (CCV, CERV and PWC) are marked by dotted vertical lines, having the same color label as for the lossy compression results; **(b)** a zoom in at low bitrate.

We conclude that both CERV and APC algorithms are good solutions for compressing the depth-map images. If the image has a high density of contour edges in the contour graph, then we recommend to use the CERV algorithm, while if the depth-map is much simpler, with a low density of contour edges in the contour graph, then we recommend to use the APC algorithm.

In Figure 5.3, we show the encoding time for CERV-Fast, CALIC, PWC, and LOCO-I for algorithm complexity comparison. The figure shows that the complexity of the CERV-Fast algorithm is close to the complexity of the CALIC coder. Note that only CERV-Fast was chosen for comparison because is the only implemented algorithm for which we have done at least some type of optimization. The `C` implementation of CERV-HiCo and APC were not optimized, and their runtime is around 60% longer than the CERV-Fast runtime.

5.3 Results for lossy compression

The results for our lossy compression algorithms are obtained by first generating a sequence of lossy images having given distortions, using the GSOm or GSOs algorithm, and then applying for each image one of the lossless compression algorithms: CERV, APC, CCV, CCLV or PWC, for compression. In the following subsections we selected a few images for which to show comparative results, but more results can be found on the webpages developed for each of our algorithms: for the GSO algorithm see [80], for the P-GSO algorithm see [78], for the PF algorithm see [81].

For each image, the compression results are presented in RD plots, where the distortion of the depth-map image is measured using PSNR, which is computed

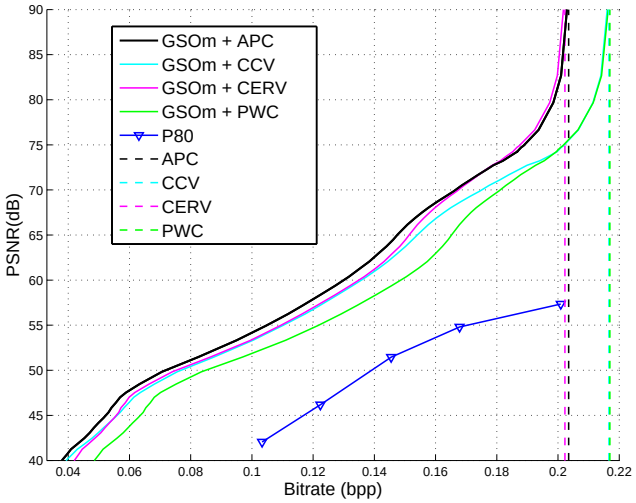


Figure 5.5: Lossy compression results for image *Aloe*, full-size resolutions, left view, from *Middlebury* dataset.

using the MSE values as

$$PSNR = 10 \cdot \log_2 \frac{255^2}{MSE}, \tag{5.1}$$

and the bitrate is computed as

$$bpp = 8 \cdot \frac{compressed\_file\_size}{image\_size}, \tag{5.2}$$

where the bitrate value, *bpp* (*bits per pixel*), shows the number of bits needed to encode a pixel in the image, and $compressed\_file\_size$ is the size of the file in bytes.

5.3.1 Region reconstruction using constant model

Figure 5.4 shows comparative results for one frame from the *Breakdancers* sequence. The GSOm algorithm generates a sequence of lossy images, which are then encoded using an entropy coding algorithm such as CCV, CERV or PWC. The GSOs algorithm generated a sequence of lossy images for the CCLV algorithm, while NL-CRS is also shown in the figure. From the state of the art algorithms we selected the JPEG2000 algorithm, the H.264 standard, and the algorithm from [53], called here P80. In Figure 5.5, APC is also used for comparison this time for the image *Aloe* from *Middlebury* dataset.

One can see that the GSO algorithm generates segmentations for a wide range of distortions, and the combinations of the GSO algorithm with the lossless compression algorithms is a good solution for compression, since the obtained results show

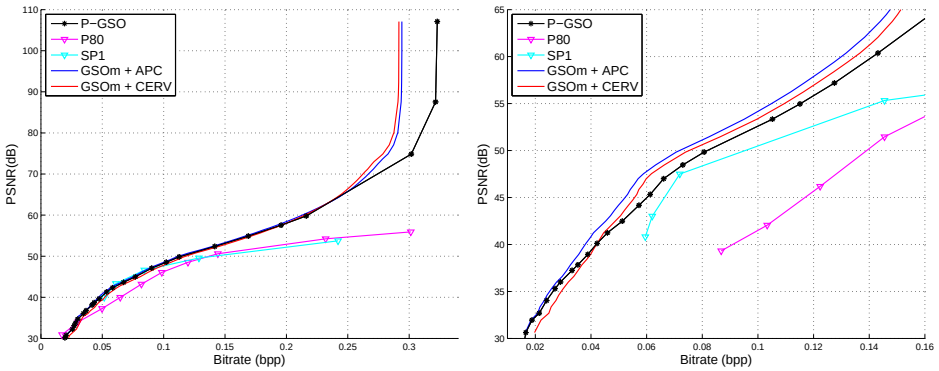


Figure 5.6: Progressive coding results for **(a)** frame 96, first view (cam0), from *Ballet* sequence, **(b)** *Aloe* image, *Middlebury* dataset, full-size resolution, left view, zoomed in the PSNR range [30, 65] dB.

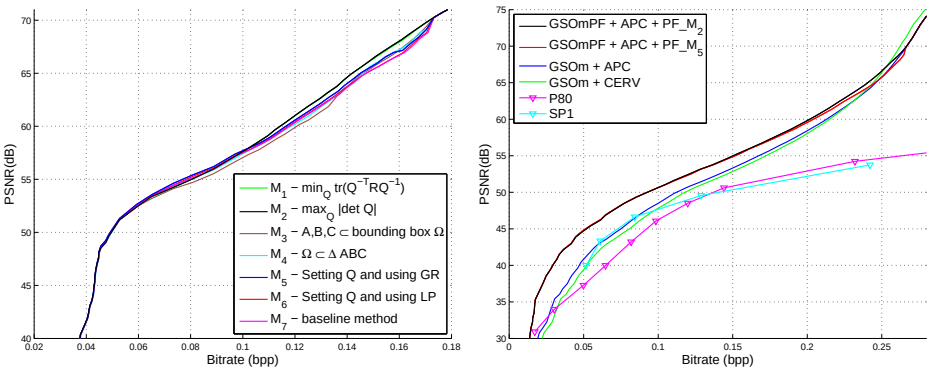
significant improvement over all other lossy compression algorithms. The Figure 5.5 shows also that, even if at lossless compression the CERV-HiCo algorithms shows better results than the APC algorithm, for lossy compression the APC algorithm has an advantage over the CERV algorithm, since the density of the contour edges is decreased by the region merging process.

5.3.2 Progressive coding

For progressive coding we compare our P-GSO algorithm with the state of the art progressive method P80, and with other non-progressive methods: SP1 from [53, 104], and our combination of GSOm algorithm and the APC or CERV algorithms. Figure 5.6 shows progressive coding results, where one can see that P-GSO obtains better results than the other progressive coding methods and almost similar results to the non-progressive methods for a large interval of bitrate. Also, one can notice that the P-GSO algorithm can provide lossy-to-lossless compression, an achievement that the other progressive methods can not reach.

5.3.3 Region reconstruction using the planar model

In Section 4.5 we describe the algorithm from [P7], where we introduced seven methods for finding the three positions A, B, C, whose heights in LS plane represents the parameters of the 3H method. Figure 5.7.(a) shows comparative results of the seven methods, for an image from the *Middlebury* dataset. One can see that the M_2 method obtains the best results for almost half of the [40, 70] dB range, and very similar results for the rest of the range, while the M_5 method has the best results for the other half of the range. The decision, to select these two methods, was taken based on the experiments done on a set of test images. The comparative results for this set can be found on the PF webpage [81].



(a) (b)

Figure 5.7: **(a)** Comparison between the seven methods M_1:M_7 tested for plane fitting, for the *Art* image, *Middlebury* dataset, full-size resolution, left view. **(b)** Comparison results between the state of the art methods and our algorithms used with and without the PF method.

In Figure 5.7.(b), we compare the two state of the art methods, P80 and SP1, with the previously developed algorithms: GSOm is used together with CERV or APC, at which we added the PF method for extending GSOm and for improving the region reconstruction model. Here we selected the APC lossless compressor for the sequence of images, but the CERV algorithm can be also used, or any other lossless compressor. In 5.7.(b), one can see that the PF method is producing a significant improvement of the results, with a maximum of around 5 *dB* at the 0.05 *bpp* bitrate. The improvement is even larger for other images, for more results see the PF webpage [81].

Chapter 6

Original Contributions and Conclusions

"Nothing is impossible, the word itself says 'I'm possible'!"

— Audrey Hepburn

This final chapter presents in the first section the original contributions of the compilation of articles included in this dissertation in the domains of Signal Processing (SP) and Image Processing (IP). The second section presents the author's contributions to the seven publications [P1]-[P7], while in the third section we draw the final conclusions for the work presented in this dissertation.

6.1 Original contributions

The goal of this work was to propose algorithms for depth-map image compression. To achieve this goal, the research was divided into two parts. The first part was focused on developing lossless compression algorithms for depth-map images, which was the subject of Chapter 3. The approach used for compressing a depth-map image was to encode a segmentation and to reconstruct each region of the segmentation. Therefore, we developed algorithms for compressing the contours of an image segmentation: CERV-Alg. C [P3] and APC [P5]; and algorithms for region reconstruction: NCV [P1] and CERV-Alg. Y [P3]. In the second part of this research we focused on developing algorithms for lossy compression of depth-map images, which was the subject of Chapter 4. The lossy compression was achieved by the algorithms by tackling different problems: image segmentation (L-CRS [P2] and GSO [P4]), progressive coding (P-GSO [P6]), and parameterization of planar models (PF [P7]).

In our research, the designed coders are mostly *asymmetric*, because the encoder performs more time consuming tasks compared to the decoder, e.g. context

tree pruning, anchor points search, contour segments generation, image segmentation, plane estimation, etc.; while the decoder is less complex and is much faster than the encoder, since all received information is just 'patched' together to create the reconstructed image.

The main contributions of the dissertation can be summarized in the following list:

(1) *Predictive coding [P1].* The predictive coding techniques are part of an important approach in image compression, which was successfully used in the state of the art coders like CALIC and LOCO-I. Our research presented a mixture of 15 predictors, for both column-wise search and row-wise search, which obtained good results. It was also the first subject of our research.

(2) *Image Segmentation [P2, P4].* An important side outcome, obtained in our research, is the development of an efficient image segmentation algorithm. The GSO algorithm offered to the user the possibility to generate a segmentation of an image by creating a lossy image of the initial image, having a given distortion measured in PSNR. Image segmentation is an important research topic, where many of our results are relevant.

(3) *Contour compression [P3, P5].* One of the main problems that we had to solve, for obtaining an efficient lossless coder, was to develop an efficient contour compression algorithm. The main reason, for which we researched the problem, is that the codelength for encoding the contour represents a large percentage (between 70% and 90%) of the final codelength. Two algorithms were developed: one is APC, which is 'drawing' very easy and fast the contour using vertex positions, has a complex contour segment generation, and is more suitable for less complex segmentations; the other one is CERV-Alg. C, which is more suitable for complex segmentations, since it is able to find deterministic cases and to classify much better the contour information into horizontal contour edges and vertical contour edges.

(4) *Region reconstruction using constant model [P3].* One of the most important algorithms, developed in our research, is the CERV-Alg. Y algorithm. Because of the efficiency of Alg. Y in encoding constant model parameters, the results were improved with around 15%, and the percentage of codelength allocated for the region's reconstruction stage was decreased to less than 15% of the final codelength, reason why Alg. Y was integrated with most of our coders.

(5) *Progressive coding [P6].* Progressive coding is an important functionality for an algorithm, and hence we developed an efficient algorithm to compress a sequence of GSOm images. The algorithm takes advantage of the way the images are generated, by searching for anchor points on the contours of the previously decoded image, and by encoding the constant model parameters using a priori information.

(6) *Parameterization of planar model [P7].* Our last published algorithm has shown that a good parameterization of the planar model is producing signif-

icant improvements in lossy compression. The Three Heights (3H) parameterization is taking advantage of Alg. Y, and introduces an efficient algorithm for encoding depth differences, called Alg. D. The contribution in the parameterization was to develop methods that select the positions of three heights, in such a way that the obtained distortion is close to the minimum distortion, and the plane parameters are encoded efficiently by Alg. D.

6.2 Author's contribution

This research in the field of depth-map image compression was performed while the author was a researcher at the Department of Signal Processing, Tampere University of Technology, Tampere, Finland. The research was supervised by *Professor Ioan Tabus* from the Department of Signal Processing of Tampere University of Technology. All publications are the result of a close collaboration with my supervisor. The author of the thesis is the first author and the main contributor to publications [P1, P2, P4, P5, P6, P7], and the second author of publication [P3].

The strategy for encoding the depth-map images was suggested by Professor Ioan Tabus, and it was first used for the development of the algorithm from [82], which presents the research done in my *Master of Science* Thesis [77]. Brief descriptions of the author's contributions to each publication, included in this compilation of articles, are presented in the following:

[P1] The author proposed the main idea of the algorithm. He is also responsible for the implementation of the algorithm, experimental results, simulations, and for writing the manuscript. The collaboration with the second author, Professor Ioan Tabus, consisted in several constructive discussions about the algorithm. He also helped to improve the quality of the final manuscript.

[P2] The author proposed the algorithm's scheme and is responsible for the design and implementation of the algorithm, for running the experiments and obtaining the results, and for writing the manuscript. The collaboration with Professor Ioan Tabus consisted of several constructive discussions about the algorithm. He was also involved in the effort to improve the final manuscript of the publication.

[P3] First author proposed the main idea of the CERV algorithm and an implementation of the algorithm in the MATrix LABoratory (MATLAB) programming language. He was also responsible for writing the manuscript. During the several constructive discussions, the author of the thesis presented modifications of the initial algorithm which improved the results. The author of the thesis (second author of the publication) implemented the CERV algorithm in the `C` programing language and added important characteristics to the algorithm: due to the `C` programing language the author was able to develop an implementation where the maximum context tree depth was increased, which allowed the algorithm to create more contexts; he improved the pruning process of the context tree with a faster and more precise estimation of

the codelengths computed for each context tree branch; he decreased the codelength needed to transmit the optimal context tree to the decoder; he proposed implementation aspects that lowered the complexity of the algorithm and decreased the runtime of the algorithm from a few seconds to a few tens of milliseconds. He is responsible for the implementation of all four versions of the algorithm, including CERV-HiCo and CERV-Fast, the design and implementation of the website, and the running of the experiments for all the reported results. The author is responsible for adding to the CERV-Fast implementation different types of optimizations, the most important being the patches of code hand-written in assembly which replaced the most performance-sensitive parts of an algorithm. The author also helped to improve the quality of the final manuscript.

[P4] The author is responsible for the main ideas in the GSO algorithm. The author is responsible for the designed and implemented the website and the two algorithms, GSOm and GSOs, for the running all the experiments and simulations, and for writing the manuscript. The collaboration with Professor Ioan Tabus consisted in constructive discussions about the algorithm. His experience played an important role in writing the final manuscript of the publication, where the problem regarding the length of the publication, imposed by the IEEE Signal Processing Letters, was overcome after long discussions.

[P5] The author is responsible for the design and implementation of the APC algorithm and its website, for running the experiments and simulation, and for writing the manuscript. The collaboration with the second author consisted in discussions about the design of the algorithm and about the final manuscript of the publication.

[P6] The author is responsible for the design and implementation of the P-GSO algorithm and its website, for running the experiments and simulation, and for writing the manuscript. The collaboration with the second author consisted in discussions about the algorithm. He also helped to improve the final manuscript of the publication.

[P7] The author is responsible for the `C` implementation of the seven methods introduced by the article, for the extension of the GSOm algorithm to the GSOmPF algorithm, for running the experiments, for designing and implementing the algorithm's website, and for the writing of the manuscript. The second author had the main idea of the publication: the selection of the three positions A, B, C as the points that form the triangle with a maximum area in the region, i.e. M_2 method. He also wrote the first MATLAB script that tested the methods M_1 and M_2. Several constructive discussions helped the author to design the final version of some methods. The second author was also involved in the effort to improve the final manuscript of the publication.

6.3 Conclusions

The dissertation is a compilation of seven articles, where several algorithms are proposed for depth-map image compression: three algorithms for lossless compression [P1, P3, P5], and four algorithms for lossy compression [P2, P4, P6, P7]. In our research, we choose to encode the depth-map images using an image segmentation, transmitted to the decoder using a contour compression algorithm, and then filling the obtained regions, using a region reconstruction algorithm. Hence, our developed lossless compression coders provide an algorithm for each of these tasks. Two strategies were studied and two algorithms were developed to encode the contour using vertex positions or contour edges. Also, two algorithms were developed for region reconstruction: CERV and APC coders are the best lossless compression algorithms that we developed. They outperform the state of the art coders all the time.

In the proposed lossy compression algorithms, we used the strategy of generating sequences of images for each specific distortion, which are then compressed by a lossless compressor. We further proposed the development of a progressive coding algorithm, P-GSO, and a study of the parameterization of planar models, PF.

At the beginning of our research two algorithms were developed, NCV and L-CRS, that generated segmentations based on the concept of variability inside a region. Later we developed algorithms with better results, nevertheless they represent the starting point of our research in the two areas.

Appendix A

Implementation Aspects

"Simplicity is prerequisite for reliability."
— Edsger W. Dijkstra

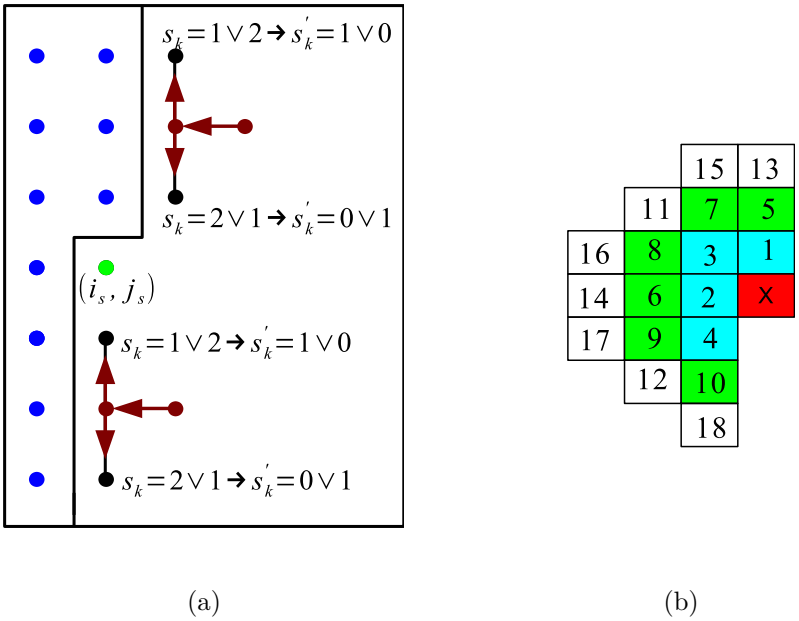
In this chapter, we present auxiliary information regarding the implementation of the algorithms presented in Chapter 3 and Chapter 4. All developed algorithms were first implemented and tested in MATLAB [59, 60], the programming language invented by *Cleve Barry Moler* in 1980. However, the MATLAB implementations of the algorithms were found to be too slow, and after some preliminary MATLAB tests almost every algorithm was reimplemented in the `C` programming language, where Microsoft Visual Studio [8] was the Integrated Development Environment (IDE) used to develop and compile each algorithm's `C` project. For the NCV [P1] and L-CRS [P2] algorithms we used a combination of MATLAB and `C` code, while the other five algorithms CERV [P3], GSO [P4], APC [P5], P-GSO [P6] and PF [P7] are implemented in the `C` programming language.

The last stage in each compressor is represented by the entropy coding stage, which has as the basic principle the *Arithmetic Coding* algorithm [46], and hence in each `C` project we used the implementation of the *Arithmetic Coder* designed by Ian H. Witten in [97].

A.1 APC algorithm

In APC, before coding the vector S of 3OT symbols, some deterministic changes are introduced at the encoder. The role of these changes is to modify the distribution of the 3OT symbols in such a way that the final codelength is reduced. Note that the changes are deterministic and the decoder can always detect and reverse them.

The first change is to reduce the number of 3OT symbols 2 in the vector S, and is done by analyzing the special cases when the description of a contour segment Γ_k reaches a vertex, placed on the border of an image area, where the column-wise search guarantees that there is no adjacent vertex to codify a 3OT symbol 2. For example, when we find an anchor point $P_1 = (i_s, j_s)$ using the column-wise search,



(a) (b)

Figure A.1: **(a)** Reducing the number of 3OT symbols $s_k = 2$, when symbol $s_k = 0$ is impossible. Blue dots are marking the vertices checked for anchor points, the arrows indicate the next vertex to be visited, red (black) dots are marking the visited (unvisited) vertices, and the green dot is marking the found anchor point. **(b)** Template context used for coding Υ, where 'x' is the label of current position.

then this guarantees that, in the contour graph, there are no unvisited vertex positions in the previous $j_s - 1$ columns and in the first $i_s - 1$ positions (lines) of the j_s column. Hence, any vertex $P_k = (i, j)$ found at the boundary of this area, for which $i < i_s$ and $j = j_s + 1$ (or with $i \geq i_s$ and $j = j_s$), has a maximum of three unvisited adjacent vertices. When a vertex $P_k = (i, j)$ is reached and the previous vertex in the contour segment was $P_{k-1} = (i, j + 1)$, then there are only two more possibilities to further describe the contour segment, by selecting the adjacent vertex $P_{k+1} = (i - 1, j)$ or $P_{k+1} = (i + 1, j)$ that is unvisited. In this case symbol $s_k = 0$ is an impossible 3OT symbols, and that is why we remap the remaining possible 3OT symbols, 1 and 2, with the symbol 0 and respectively symbol 1, as it is shown in Figure A.1.(a).

The second change done is to modify the optimal context tree obtained for the vector S of 3OT symbols. Note that only for the 3OT representation, the optimal context tree is much more unbalanced than other cases because the representation was designed so that the symbols 2 appears as few times as possible. Hence, in the optimal ternary context tree obtained for vector S, the subtree for the branch labeled '0' (for symbol 0) has many leaf nodes, while the subtree for the branch labeled '2' (for symbol 2) has only a few leaf nodes. By analyzing the 3OT

representation, we notice that any two consecutive vertices, in a contour segment, can not be codified by two consecutive symbols 2, and this case may appear in S only because of vector concatenation. Hence, we studied the context tree of a general 3OT chain-code vector, and noticed that, in the context ($s_{k-2} \neq 2$, $s_{k-1} = 2$), the symbol $s_k = 1$ is more frequent than the symbol $s_k = 0$, i.e. the probability to encode a symbol 1 is higher than the probability to encode a symbol 0. This is why the symbol $s_k = 1$ is changed into $s'_k = 0$, and $s_k = 0$ into $s'_k = 1$. The change introduces also some other small 'branch interchanges' between the subtrees '0' and '1', but overall the change has a positive effect. However, because the branch '2' has only a few leaf nodes with a similar position on other branches, the change does not cause disturbances in the optimal context tree.

Another implementation aspect used in APC is the special coding of matrix Υ, done using the TCTM. Matrix Υ is encoded as the latest information, in column-wise scanning, using the template context of length $d_{\mathcal{T}} = 18$, as it is shown in Figure A.1.(b). In our tests, after pruning, the optimal context tree has about seven leaf nodes. The alphabet used to create the context tree contains three symbols, however the coding distributions for each node are computed using only two symbols, since symbol 2 is ignored because does not codify any useful information and only specifies that the information regarding current position is already available. At the decoder, if $\Upsilon(i, j) = 1$ is decoded, then the contour segment Γ_k having the anchor point $P_1 = (i, j)$ is decoded and Υ is updated with symbols 2 on the vertex positions of Γ_k. Only after drawing Γ_k, the decoding of Υ is resumed.

A.2 Non-stationarity of context distributions

The conditional distribution, collected in each context node of the context tree, is updated on the fly using an estimator, and is a reflection of the data observed in the current context. For depth-map images, the context distributions are non-stationary since the contour may have different shapes in different areas of the image. This is why it is useful to down-weight some counts in each context after a period of time. Hence, a halving process was introduced to halve the counts of a given context, each time when the sum of the number of occurrences, of each symbol in the alphabet, exceeds a certain threshold.

The optimal threshold varies for each image, and it depends on the representation of the symbols for which it is used. Hence, by setting the threshold too low the contexts might not have collected enough information, while by setting the threshold too high the contexts might be updated too slowly and the halving procedure will not influence the results. For example, in the CERV algorithm the threshold was set equal to 250, when coding the contour edges, except for the context 0 which was left unchanged. On the other hand, in the APC algorithm, the threshold was set to 511 when coding of the vector Φ.

Note that for the arithmetic coder with a precision of 16 bits, this threshold was already introduced and set to the value of 16383, i.e. to $2^{14} - 1$.

A.3 Coordinates scaling for a better precision

In (2.9), the coordinates $[x_i\ y_i\ 1],\ i = 1, 2, \ldots n$, obtained for a region Ω_ℓ, must be normalized for improving the precision of the optimal plane parameters, estimated by the LS algorithm. In [27], the authors are showing that the non-isotropic scaling of the coordinates has an important role in improving the precision of the optimal plane parameters $\underline{\theta}^*$. Hence, we used the transformed coordinates $\begin{bmatrix} x_i' & y_i' & 1 \end{bmatrix} = \begin{bmatrix} x_i & y_i & 1 \end{bmatrix} T$ in (2.9), where the matrix T is of form

$$T = \begin{bmatrix} \frac{\sqrt{2}}{\delta_x} & 0 & 0 \\ 0 & \frac{\sqrt{2}}{\delta_y} & 0 \\ -\frac{\sqrt{2}\bar{x}}{\delta_x} & -\frac{\sqrt{2}\bar{y}}{\delta_y} & 1 \end{bmatrix}, \tag{A.1}$$

where we remind that $x_m = \min_i x_i$, $x_M = \max_i x_i$, $y_m = \min_i y_i$, $y_M = \max_i y_i$, $i = 1, 2, \ldots, n$, and $\delta_x = x_M - x_m$, $\delta_y = y_M - y_m$, $\bar{x} = x_m + \frac{\delta_x}{2}$, $\bar{y} = y_m + \frac{\delta_y}{2}$.

The transform was applied for all the methods $M_1 : M_7$ presented in [P7], however, only for the methods $M_5 : M_7$ our tests showed an improvement in the results.

References

[P1] I. Schiopu and I. Tabus. Depth image lossless compression using mixtures of local predictors inside variability constrained regions. In *Proc. International Symposium on Communications, Control, and Signal Precessing (ISCCSP)*, pages 1–4, Rome, Italy, May 2012.

[P2] I. Schiopu and I. Tabus. Lossy and Near-Lossless compression of depth images using segmentation into constrained regions. In *Proc. European Signal Processing Conference (EUSIPCO)*, pages 1099–1103, Bucharest, Romania, August 2012.

[P3] 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 (IEEE TIP)*, 22(11):4195–4210, November 2013.

[P4] I. Schiopu and I. Tabus. Lossy depth image compression using greedy rate-distortion slope optimization. *IEEE Signal Processing Letters (IEEE SPL)*, 20(11):1066–1069, November 2013.

[P5] I. Schiopu and I. Tabus. Anchor points coding for depth map compression. In *Proc. IEEE International Conference on Image Processing (IEEE ICIP)*, pages 5626–5630, Paris, France, October 2014.

[P6] I. Schiopu and I. Tabus. Lossy-to-lossless progressive coding of depth-maps. In *Proc. International Symposium on Signals, Circuits and Systems (ISSCS)*, pages 1–4, Iasi, Romania, July 2015.

[P7] I. Schiopu and I. Tabus. Parametrizations of planar models for region-merging based lossy depth-map compression. In *Proc. 3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*, pages 1–4, Lisbon, Portugal, July 2015.

[8] Microsoft Visual Studio. `www.visualstudio.com/`. Accessed: 2015-04-15.

[9] Advanced video coding for generic audiovisual services. *ITU-T Recommendation H.264 and ISO/IEC 14496-10 (AVC)*, 2009.

[10] A. Akimov, A. Kolesnikov, and P. Franti. Lossless compression of map contours by context tree modeling of chain codes. *Pattern Recognition*, 40(3):944–952, March 2007.

[11] S. Alcaraz-Corona and R. M. Rodriguez-Dagnino. Bi-level image compression estimating the markov order of dependencies. *IEEE Journal of Selected Topics in Signal Processing (IEEE JSTSP)*, 4(3):605–611, June 2010.

[12] P.J. Ausbeck Jr. The piecewise-constant image model. *Proceedings of the IEEE*, 88(11):1779–1789, November 2000.

[13] M. Babel, O. Deforges, and J. Ronsin. Interleaved S+P pyramidal decomposition with refined prediction model. In *Proc. IEEE International Conference on Image Processing (IEEE ICIP)*, volume 2, pages II–750–3, Genova, Italy, September 2005.

[14] R. Begleiter and R. El-Yaniv. Superior guarantees for sequential prediction and lossless compression via alphabet decomposition. *Journal of Machine Learning Research*, 7:379–411, February 2006.

[15] N. Brady and F. Bossenb. Shape compression of moving objects using context-based arithmetic encoding. *Signal Processing: Image Communication*, 15(7-8):601–617, 2000.

[16] M. Cagnazzo and B. Pesquet-Popescu. Depth map coding by dense disparity estimation for MVD compression. In *Proc. International Conference on Digital Signal Processing (ICDSP)*, pages 1–6, July 2011.

[17] L. Cappellari, C. Cruz-Reyes, G. Calvagno, and J. Kari. Lossy to lossless spatially scalable depth map coding with cellular automata. In *Proc. Data Compression Conference (DCC)*, pages 332–341, March 2009.

[18] G. Carmo, M. Naccari, and F. Pereira. Binary tree decomposition depth coding for 3D video applications. In *Proc. IEEE International Conference Multimedia and Expo (IEEE ICME)*, volume 3, pages 1–6, July 2011.

[19] G. Cheung, A. Ortega, and T. Sakamoto. Fast H.264 mode selection using depth information for distributed game viewing. In *Proc. IST/SPIE Visual Communications and Image Processing (VCIP)*, pages 1–12, January 2008.

[20] C. Dai, O. D. Escoda, P. Yin, X. Li, and C. Gomila. Geometry-adaptive block partitioning for intra prediction in image video coding. In *Proc. IEEE International Conference on Image Processing (IEEE ICIP)*, volume 6, pages VI–85 – VI–88, September 2007.

[21] I. Daribo, G. Cheung, and D. Florencio. Arithmetic edge coding for arbitrarily shaped sub-block motion prediction in depth video compression. In *Proc. IEEE International Conference on Image Processing (IEEE ICIP)*, pages 1541–1544, September 2012.

[22] I. Daribo, C. Tillier, and B. Pesquet-Popescu. Adaptive wavelet coding of the depth map for stereoscopic view synthesis. In *Proc. IEEE Workshop on Multimedia Signal Processing (IEEE MMSP)*, pages 413–417, October 2008.

[23] D. Donoho. Wedgelets: nearly minimax estimation of edges. *The Annals of Statistics*, 27(3):859–897, March 1999.

[24] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, September 2004.

[25] S. W. Golomb. Run-length encoding. *IEEE Transactions on Information Theory (IEEE TIT)*, 12(3):399–401, July 1966.

[26] F. Gray. Pulse code communication, March 17 1953. US Patent 2,632,058.

[27] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[28] P. Heinz-Otto and S. Dietmar. *The Science of Fractal Images*. Springer-Verlag New York, Inc., New York, NY, USA, 1988.

[29] J. Heo and Y.-S. Ho. Improved context-based adaptive binary arithmetic coding over H.264/AVC for lossless depth map coding. *IEEE Signal Processing Letters (IEEE SPL)*, 17(10):835–838, 2010.

[30] J. Heo and Y.-S. Ho. Improved CABAC design in H.264/AVC for lossless depth map coding. In *Proc. International Conference on Multimedia and Expo (IEEE ICME)*, pages 1–4, July 2011.

[31] H. Hirschmuller and D. Scharstein. Evaluation of cost functions for stereo matching. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (IEEE CVPR)*, pages 1–8, Minneapolis, MN, USA, June 2007.

[32] P. G. Howard, F. Kossentini, B. Martins, S. Forchhammer, and W. J. Rucklidge. The emerging JBIG2 standard. *IEEE Transactions on Circuits and Systems for Video Technology (IEEE TCSCT)*, 8(7):838–848, November 1998.

[33] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, September 1952.

[34] N. Hur, H. Lee, G. S. Lee, S. J. Lee, A. Gotchev, and S.-I. Park. 3DTV broadcasting and distribution systems. *IEEE Transactions on Broadcasting (IEEE TB)*, 57(2):395–407, June 2011.

[35] Philips (in Coop with 3D4YOU). Response to new call for 3DV test material: Beergarden. ISO/IEC JTC1/SC29/WG11 Doc. M16421, April 2009.

[36] F. Jager. Contour-based segmentation and coding for depth map compression. In *Proc. IEEE Visual Communications and Image Processing (IEEE VCIP)*, November 2011.

[37] F. Jager. Depth-based block partitioning for 3D video coding. In *Proc. Picture Coding Symposium (PCS), 2013*, pages 410–413, December 2013.

[38] M.-K. Kang and Y.-S. Ho. Depth video coding using adaptive geometry based intra prediction for 3-D video systems. *IEEE Transactions on Multimedia (IEEE TM)*, 14(1):121–128, 2012.

[39] K. Y. Kim, G. H. Park, and D. Y. Suh. Bit-plane-based lossless depth-map coding. *SPIE Optical Engineering*, 49(6):067403–067403–10, 2010.

[40] S.-Y. Kim and Y.-S. Ho. Mesh-based depth coding for 3D video using hierarchical decomposition on depth maps. In *Proc. IEEE International Conference on Image Processing (IEEE ICIP)*, pages V–117 – V–120, September 2007.

[41] W.-S. Kim, S. K. Narang, and A. Ortega. Graph based transforms for depth video coding. In *Proc. IEEE International Conference on Asilomar Conference on Acoustics, Speech and Signal Processing (IEEE ICASSP)*, pages 813–816, March 2012.

[42] R. Krichevsky and V. Trofimov. The performance of universal encoding. *IEEE Transactions on Information Theory (IEEE TIT)*, 27(2):199–207, March 1981.

[43] R. Krishnamurthy, B. B. Chai, H. Tao, and S. Sethuraman. Compression and transmission of depth maps for image-based rendering. In *Proc. IEEE International Conference on Image Processing (IEEE ICIP)*, volume 3, pages 828–831, 2001.

[44] Tanimoto Lab. MPEG-FTV. `www.tanimoto.nuee.nagoya-u.ac.jp/`.

[45] HP Labs. LOCO-I/JPEG-LS Download area. `www.hpl.hp.com/research/info_theory/loco/locodown.htm`.

[46] G.G. Langdon Jr. An introduction to arithmetic coding. *IBM Journal of Research and Development*, 28(2):135–149, March 1984.

[47] P. S. Laplace. *Essai philosophique sur les probabilités*. Courcier, 1814.

[48] S. Li, J. Lei, C. Zhu, L. Yu, and C. Hou. Pixel-based inter prediction in coded texture assisted depth coding. *IEEE Signal Processing Letters (IEEE SPL)*, 21(1):74–78, January 2014.

[49] Y. Li and L. Sun. A novel upsampling scheme for depth map compression in 3DTV system. In *Proc. Picture Coding Symposium (PCS)*, pages 186–189, Nagoya, Japan, December 2010.

[50] H. S. Malvar. Adaptive Run-Length / Golomb-Rice encoding of quantized generalized gaussian sources with unknown statistics. In *Proc. Data Compression Conference (DCC)*, pages 23–32, March 2006.

[51] B. Martins and S. Forchhammer. Tree coding of bilevel images. *IEEE Transactions on Image Processing (IEEE TIP)*, 7(4):517–528, April 1998.

[52] S. A. Martucci. Reversible compression of HDTV images using median adaptive prediction and arithmetic coding. In *Proc. IEEE International Symposium on Circuits and Systems (IEEE ISCAS)*, volume 2, pages 1310–1313, May 1990.

[53] R. Mathew, D. Taubman, and P. Zanuttigh. Scalable coding of depth maps with R-D optimized embedding. *IEEE Transaction on Image Processing (IEEE TIP)*, 22(5):1982–1995, May 2013.

[54] R. Mathew, P. Zanuttigh, and D. Taubman. Highly scalable coding of depth maps with arc breakpoints. In *Proc. Data Compression Conference (DCC)*, pages 42–51, April 2012.

[55] S. Mehrotra, Z. Zhang, Q. Cai, C. Zhang, and P. A. Chou. Low-complexity, near-lossless coding of depth maps from Kinect-like depth cameras. In *Proc. IEEE International Workshop on Multimedia Signal Processing (IEEE MMSP)*, pages 1–6, October 2011.

[56] Microsoft Research. MSR 3D Video Dataset. `http://research.microsoft.com/en-us/um/redmond/groups/ivm/vvv/`.

[57] S. Milani and G. Calvagno. A depth image coder based on progressive silhouettes. *IEEE Signal Processing Letters (IEEE SPL)*, 17(8):711–714, August 2010.

[58] S. Milani, P. Zanuttigh, M. Zamarin, and S. Forchhammer. Efficient depth map compression exploiting segmented color data. In *Proc. IEEE International Conference on Multimedia and Expo (IEEE ICME)*, pages 1–6, July 2011.

[59] C. B. Moler. MATLAB — an interactive matrix laboratory. Technical Report 369, University of New Mexico. Dept. of Computer Science, 1980.

[60] C. B. Moler. MATLAB user's guide. Technical Report CS81-1, University of New Mexico. Dept. of Computer Science, November 1980.

[61] Y. Morvan, P. H. N. de With, and D. Farin. Platelet-based coding of depth maps for the transmission of multiview images. In *Proceedings of SPIE, Stereoscopic Displays and Applications*, volume 6055, pages 93–100, San Jose (CA), USA, January 2006.

[62] 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 International Conference on Image Processing (IEEE ICIP)*, volume 5, pages V–105 – V–108, September 2007.

[63] Moving Picture Experts Group (MPEG). The MPEG official webpage. `http://mpeg.chiariglione.org/`.

[64] K.-J. Oh, S. Yea, A. Vetro, and Y-S Ho. Depth reconstruction filter and down/up sampling for depth coding in 3-D video. *IEEE Signal Processing Letters (IEEE SPL)*, 16(9):747–750, September 2009.

[65] B. O. Ozkalayci and A. A. Alatan. 3D planar representation of stereo depth images for 3DTV applications. *IEEE Transactions on Image Processing (IEEE TIP)*, 23(12):5222–5232, December 2014.

[66] F. Pasteau, M. Babel, O. Deforges, C. Strauss, and L. Bedat. Locally Adaptive Resolution (LAR) codec. In Ashraf A. Zaher, editor, *Recent Advances in Signal Processing*, pages 37–48. IN-TECH Education and Publishing, November 2009.

[67] S. K. Penta and P. J. Narayanan. Compression of multiple depth maps for IBR. *The Visual Computer*, 21(8–10):611–618, 2005.

[68] V. Ratnakar. RAPP: lossless image compression with runs of adaptive pixel patterns. In *Proc. Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1251–1255, November 1998.

[69] R. Rice and J. Plaunt. Adaptive variable-length coding for efficient compression of spacecraft television data. *IEEE Transactions on Communication Technology (IEEE TCT)*, 19(6):889–897, December 1971.

[70] J. Rissanen. A universal data compression system. *IEEE Transactions on Information Theory (IEEE TIT)*, 29:656–664, Sepember 1983.

[71] K. Sakuragi and A. Kawanaka. Depth estimation from stereo images using sparsity. In *Proc. IEEE International Conference on Signal Processing (IEE ICSP)*, pages 1161–1164, October 2010.

[72] 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 International Conference on Multimedia and Expo Workshops (IEEE ICMEW)*, pages 1–6, San Jose, CA, USA, 2013.

[73] A. Sanchez, G. Shen, and A. Ortega. Edge-preserving depth-map coding using graph-based wavelets. In *Proc. Asilomar Conference Record on Signals, Systems and Computers*, pages 578–582, November 2009.

[74] H. Sanchez-Cruz and R. M. Rodriguez-Dagnino. Compressing bi-level images by means of a 3-bit chain code. *SPIE Optical Engineering*, 44(9):1–8, 2005.

[75] D. Scharstein and C. Pal. Learning conditional random fields for stereo. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (IEEE CVPR)*, pages 1–7, Minneapolis, MN, USA, June 2007.

[76] D. Scharstein and R. Szeliski. Middlebury Stereo Vision Page. `http://vision.middlebury.edu/stereo/`.

[77] I. Schiopu. Segmentation and compression of depth images. Master's thesis, Tampere University of Technology, Tampere, Finland, May 2011.

[78] I. Schiopu and I. Tabus. Additional results file for the P-GSO Algorithm. `www.cs.tut.fi/~schiopu/PGSO/PGSOresults.pdf`.

[79] I. Schiopu and I. Tabus. The webpage of the APC Algorithm. `www.cs.tut.fi/~schiopu/APC/`.

[80] I. Schiopu and I. Tabus. The webpage of the GSO Algorithm. `www.cs.tut.fi/~schiopu/GSO/`.

[81] I. Schiopu and I. Tabus. The webpage of the PF Algorithm. `www.cs.tut.fi/~schiopu/PF/`.

[82] I. Schiopu and I. Tabus. MDL segmentation and lossless compression of depth images. In *Proc. Workshop on Information Theoretic Methods in Science and Engineering (WITMSE)*, pages 55–58, Helsinki, Finland, August 2011.

[83] I. Schiopu and I. Tabus. Lossless contour compression using chain-code representations and context tree coding. In *Proc. Workshop on Information Theoretic Methods in Science and Engineering (WITMSE)*, pages 6–13, Tokyo, Japan, August 2013.

[84] F. Schumacher and T. Greiner. Matching cost computation algorithm and high speed fpga architecture for high quality real-time semi global matching stereo vision for road scenes. In *Proc. IEEE International Conference on Intelligent Transportation Systems (IEEE ITSC)*, pages 3064–3069, October 2014.

[85] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 623–656, July, October 1948.

[86] I. Tabus, J. Hukkanen, and I. Schiopu. Two-phase compression of histological images with MDL ranking of segmentation images. In *Proc. International Conference on Control Systems and Computer Science (CSCS)*, pages 331–338, Bucharest, Romania, May 2013.

[87] I. Tabus and S. Sarbu. Optimal structure of memory models for lossless compression of binary image contours. In *Proc. IEEE Conference on Acoustics, Speech and Signal Processing (IEEE ICASSP)*, pages 809–812, Prague, Czech Republic, May 2011.

[88] I. Tabus and I. Schiopu. Quaternary crack-edge representations for lossless contour compression. In *Proc. International Conference on Control Systems and Computer Science (CSCS)*, pages 339–344, Bucharest, Romania, May 2013.

[89] I. Tabus, I. Schiopu, and J. Astola. The webpage of the CERV Algorithm. `www.cs.tut.fi/~tabus/CERV/`.

[90] S. R. Tate. Lossless compression of region edge maps. Technical report, Deparment of Computer Science, Duke University, Durham, NC, 1992.

[91] C. Unger. *Contributions to Stereo Vision*. Dissertation, Technische Universitat Munchen, Munich, Germany, 2013.

[92] G. K. Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics (IEEE TCS)*, 38(1):xviii–xxxiv, February 1992.

[93] 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 (IEEE TIP)*, 9:1309–1324, August 2000.

[94] M. J. Weinberger, G. Seroussi, and G. Sapiro. LOCO-I: a low complexity, context-based, lossless image compression algorithm. In *Proc. Data Compression Conference (DCC)*, pages 140–149, March 1996.

[95] M. Weinbergera, J. Rissanen, and R. Arps. Applications of universal context modeling to lossless compression of gray-scale images. *IEEE Transactions on Image Processing (IEEE TIP)*, 4:575–586, April 1996.

[96] R. M. Willett and R. D. Nowak. Platelets: a multiscale approach for recovering edges and surfaces in photon-limited medical imaging. *IEEE Transactions on Medical Imaging (IEEE TMI)*, 22:332–350, March 2003.

[97] I. H. Witten, N. M. Radford, and C. G. John. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.

[98] X. Wu. CALIC executable files. `www.ece.mcmaster.ca/~xwu/calicexe/`.

[99] X. Wu and N. Memon. Context-based, adaptive, lossless image coding. *IEEE Transactions on Communications (IEEE TC)*, 45(4):437–444, April 1997.

[100] X. Xu, L.-M. Po, T. C.-H. Cheung, K.-W. Cheung, L. Feng, C.-W. Ting, and K.-H. Ng. Adaptive depth truncation filter for MVC based compressed depth image. *Signal Processing: Image Communication*, 29(3):316–331, March 2014.

[101] M. Zamarin and S. Forchhammer. Lossless compression of stereo disparity maps for 3D. In *Proc. IEEE International Conference on Multimedia and Expo Workshops (IEEE ICMEW)*, pages 617–622, July 2012.

[102] M. Zamarin and S. Forchhammer. Edge-preserving intra mode for efficient depth map coding based on H.264/AVC. *Signal Processing: Image Communication*, 29(7):711–724, August 2014.

[103] M. Zamarin, M. Salmistraro, S. Forchhammer, and A. Ortega. Edge-preserving intra depth coding based on context-coding and H.264/AVC. In *Proc. IEEE International Conference on Multimedia and Expo (IEEE ICME)*, pages 1–6, July 2013.

[104] P. Zanuttigh and G. M. Cortelazzo. Compression of depth information for 3D rendering. In *Proc. 3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*, pages 1–4, May 2009.

[105] Z. Zhang. Microsoft Kinect sensor and its effect. *IEEE Transactions on Multimedia (IEEE TM)*, 19(2):4–12, April 2012.

[106] B. Zhu, G. Jiang, Y. Zhang, Z. Peng, and M. Yu. View synthesis oriented depth map coding algorithm. In *Proc. Asia-Pacific Conference on Information Processing (APCIP)*, volume 2, pages 104–107, July 2009.

[107] 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*, pages 600–608, Los Angeles, CA, USA, August 2004.

Publications

Publication 1

Copyright ©2012 IEEE. Reprinted, with permission, from

[P1] I. Schiopu and I. Tabus. Depth image lossless compression using mixtures of local predictors inside variability constrained regions. In *Proc. International Symposium on Communications, Control, and Signal Precessing (ISCCSP)*, pages 1–4, Rome, Italy, May 2012.

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Tampere University of Technology's products or services. Internal or personal use of this material is permitted. If interested in reprinting/ republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

DEPTH IMAGE LOSSLESS COMPRESSION USING MIXTURES OF LOCAL PREDICTORS INSIDE VARIABILITY CONSTRAINED REGIONS

Ionuţ Şchiopu and Ioan Tăbuş

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

This paper studies the lossless compression of depth images realized by first transmitting contours of suitably chosen regions and subsequently performing predictive coding inside each region and transmitting the prediction residuals. For the large constant depth regions only the contour needs to be transmitted along with the value of the depth inside each region, while for the rest of the image we find suitable regions where the local variation of the depth level from one pixel to another is constrained above. The nonlinear predictors used for each region combine the results of several linear predictors, each fitting optimally a subset of pixels belonging to the local neighborhood. Overall the obtained results exceed by a wide margin the performance of standard image compression algorithms.

***Index Terms*—** lossless compression, depth image, segmentation, local nonlinear prediction, context tree encoding

1. INTRODUCTION

The interest in depth images has considerably intensified since they play a central role in many technologies like 3D imaging and computer vision and because depth image sensors are widespread now. Since depth images possess different types of redundancy than the natural images it turns out that standard lossy and lossless compression methods can be easily surpassed over the class of depth images by taking into account the specific sources of redundancies. For natural images the luminance has quite high variance over most objects (due to their texture, coloring, visual details) and as such the lossless compression by the most efficient methods can reduce the necessary size to 2 to 4 bits per pixel, while in the case of depth images the variability of the depth is more reduced, and we show in here that we can obtain by lossless compression quite often under 1 bit per pixel. The typical redundancy for depth images is due to the existence of relatively large regions of tens to hundreds connected pixels having the same depth value, or having a very regular law of variation of the depth (e.g., accounting for planar regions or other regular geometrical surfaces). It is thus natural to attempt for depth images first a separation of the image into regions having specific forms of redundancy and subsequently to exploit suitable models inside each region. This approach was utilized in the past in a number of papers, where typically the models of a region were considered in the form of first order or second order polynomials in the (x, y) coordinates and the partitioning problem was to find the best split of the image into regions so that to minimize overall cost of transmitting the contours, the polynomial parameters for each region and the modeling residuals. Such approaches based on segmentation followed by predictive coding were discussed for example in the papers introducing MDL segmentations for color images in [1, 2, 3], and for the case of depth images in [4].

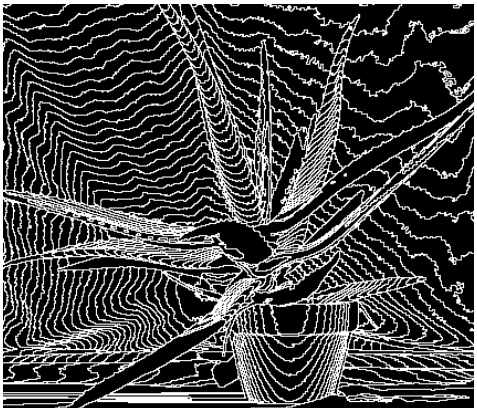


Fig. 1. Example of constrained variability segmentation of the (427×370) image Aloe(d5). The initial 5016 constant depth regions of the image are grouped by a 3 stage segmentation algorithm into $n_r = 574$ regions, when using the upper bounds on region variability $\Lambda = \{1, 2, 3\}$ and the lower bounds on region size $\{55, 45, 30\}$. The 790 constant regions of size up to 4 pixels are encoded separately and not shown in this segmentation.

Instead of the typical models of regions based on first or second order surfaces, in here we consider a different approach, where the predictive coding inside a region is realized at each pixel by a nonlinear predictor applied to a causal prediction mask. The local predictor is well suited for predicting first order variations inside the prediction mask, and will work well even for second order surfaces if their curvature is not strong and they can be approximated well by a first order surface inside the prediction mask of about 4 neighbor pixels. The prediction mask is not constant, it is carved to contain only pixels belonging to the current region, but otherwise the predictor has fixed parameters.

We follow a similar algorithmic structure as our preliminary conference paper [5], but in here we better optimize all involved stages of segmentation, prediction, encoding of contours, and encoding of residuals, improving consistently the performance as compared to [5]. Finally, we study the compression performance of our

scheme over a large set of images and observe regular groups of patterns obeyed by the tunable parameters of the algorithm, which makes possible the use of specific parameter sets for each image.

The depth image contains the depth value $I(x, y) \in \{0, 2^B - 1\}$ for each pixel (x, y). In our experiments the number of depth-planes is $B = 8$. We define the segmentation (see Figure 1) as the union of all regions, $\Omega_1, \ldots, \Omega_{n_r}$, in which the depth image is partitioned.

2. DESCRIPTION OF THE METHOD

2.1. Segmenting into variability constrained regions

The image is partitioned into regions such that the combined cost of encoding the contours of regions and the cost of encoding the prediction residuals inside each region is minimized. Partitioning the image into too many very small regions will lead to a too high cost of contours, while partitioning only into a few very large regions, each containing inside pixels with high variability of their depth, will lead to a very high cost for encoding the residuals. The best compromise is obtained by an iterative segmentation, where at consecutive iterations the allowed variability λ inside regions is increased. At a given iteration all connected components with variability at most λ are declared regions of the segmentation if each contains more than N_λ pixels. During the iterative process the important parameters will be the allowed variability λ and the minimum size N_λ.

For a current pixel (x_t, y_t) the set of the four neighbors in 4-connectivity is denoted $\mathcal{N}_4(x_t, y_t)$. We define the variability of a current pixel (x_t, y_t) inside a given region Ω_j as the minimum value of the absolute differences of the depth values between the current pixel and those neighbors which are part of the region, as follows:

$$V(x_t, y_t) = \min_{(x_i, y_i) \in \mathcal{N}_4(x_t, y_t) \cap \Omega_j} |I(x_t, y_t) - I(x_i, y_i)|.$$

The variability based segmentation algorithm is initialized by finding the sets of connected pixels having variability $\lambda = 0$ and declaring them candidate regions. We iterate for successive thresholds on variability $\lambda \in \Lambda = \{1, 2, 3\}$. At the iteration marked as Step λ, each candidate region from the previous step having at least N_λ pixels is declared a region Ω_j of the segmentation, and the remaining pixels not yet in already decided regions are then grouped in connected components and all such components with variability not exceeding λ are declared candidate regions for the next step. At the last step all candidate regions are automatically declared regions of the segmentation.

The obtained segmentation is further refined by a merging stage, where two regions smaller than $\delta = 20$ pixels can be merged, or a region smaller than δ pixels can be merged with one of the neighboring regions, larger or equal to δ pixels and having $\lambda \geq 1$. Merging is decided based on the following rule: *merge two regions if the prediction residuals' cost over the merged region is less than the sum of the prediction residuals' costs over the two separated regions and of the eliminated contour cost*.

For the constant regions with size smaller than 4 pixels an anchor point is transmitted followed by the code in an enumeration of all possible shapes. For all other regions the contour is transmitted in the 3OT representation [6] and then the prediction residuals are transmitted using entropy coding. The prediction process is described next.

2.2. Local Nonlinear Prediction

We predict the depth $I(x_t, y_t)$ for a current pixel $(x_t, y_t) \in \Omega$ by using the values $I(x_i, y_i)$ of the pixels $(x_i, y_i) \in \Omega$ which also belong to a causal neighborhood $\mathcal{N}_P(x_t, y_t)$ of the pixel (x_t, y_t), depicted at the bottom of Table 1. For each region the horizontal scanning (with causal neighbors a, c, d, e) and the vertical scanning (with causal neighbors a, c, d, f) are tested, and the one giving better performance is selected and announced to the decoder by one bit per region.

n	row scanning	column scanning
$b_0 = 1$	$z_0 = c + a - d$	$z_0 = c + a - d$
$b_1 = 1$	$z_1 = a - c + e$	$z_1 = c - a + f$
$b_2 = 1$	$z_2 = a + \frac{e-d}{2}$	$z_2 = c + \frac{f-d}{2}$
$b_3 = 1$	$z_3 = [c\ a\ d\ e]w_r$	$z_3 = [c\ a\ d\ f]w_c$

d	c	e
a	x	
f		

Table 1. Elementary predictors, $z_0, \ldots, z_3$, used for forming the final fifteen mixtures of predictors $\mathcal{P}_1, \ldots, \mathcal{P}_{15}$ where the mixture with index $n = 2^{b_3} + \ldots + 2^{b_0}$ is defined as $\mathcal{P}_{b_3 b_2 b_1 b_0}(\mathcal{N}_P(x_t, y_t)) = \{a, c, z_k | b_k = 1, k = 0, \ldots, 3\}$. The first three predictors z_0, z_1, z_2 are best linear fits using only neighbor values, while the fourth is the best linear fit over the whole image. At the bottom is depicted the prediction neighborhood $\mathcal{N}_P$ of the current pixel (x_t, y_t), which is marked by "x" and also shown are the names used for the depth values of the neighbors.

For each scanning order the optimal predictor is selected among 15 mixture predictors, $\hat{I}_n(\mathcal{N}_P(x_t, y_t)), n = 1, \ldots, 15$, which are evaluated in turn, estimating the cost, in bits, required for the residuals compression. The predictor having the smallest cost of the residuals is selected to be applied for the full image and its index n^* is sent as side information once for the whole image. Some regions will be scanned horizontally and others vertically, depending on which scanning provides better performance. In the following the prediction in horizontal scanning is described, since the principles for the two scanning orders are identical.

Each nonlinear predictor $\hat{I}_n(\mathcal{N}_P(x_t, y_t))$ is indexed by $n \in \{1, \ldots, 15\}$ or, when more informative, the index is written in binary form $b_3 b_2 b_1 b_0$ with $n = 2^{b_3} + \ldots + 2^{b_0}$. The nonlinear predictor performs the median of a collection of elementary predictions, which we called a prediction mixture $\mathcal{P}_n(\mathcal{N}(x_t, y_t))$, as follows: $\hat{I}_n(\mathcal{N}(x_t, y_t)) = median\{\mathcal{P}_n(\mathcal{N}(x_t, y_t))\}$.

Table 1 shows the elementary predictors appended in the mixture depending on the value of the bits $b_3 b_2 b_1 b_0$, which form the binary representation of n, and also depending on which scanning order is used. The elementary predictor z_i is used in the mixture $\mathcal{P}_n$ only when the ith bit of n is $b_i = 1$ and when the pixels involved in its computation exist in the neighborhood, otherwise it is left out of consideration. The weights w_r used in the globally designed predictor z_3 are obtained by solving a LS problem, where the data matrix Φ_r collects as rows the neighbor values $[c\ a\ d\ e]$, whenever they all fall inside the region of the current pixel x. Denoting by ψ_r the vector collecting all corresponding current pixels x, the LS parameters are obtained as $w_r = (\Phi_r^T \Phi_r)^{-1} \Phi_r^T \psi_r$. A similar design is performed for the column-wise scanning optimal predictor parameters w_c. The elements of the vectors w_r and w_c are sent quantized using 10 bits each in the header of the whole file, if the bit three, b_3, is set on in n^*.

2.3. Encoding of prediction residuals

Once the optimal predictor $\hat{I}_{n^*}(\mathcal{N}_P(x_t, y_t))$ is selected, the encoding of the pixels in any region Ω_i having K_i pixels is performed as follows. First determine the prediction value $\hat{I}(x_t, y_t)$ for each pixel (x_t, y_t) in the region, with $t = 1, \ldots, K_i$. We define the residuals $\epsilon(x_t, y_t) = I(x_t, y_t) - \hat{I}(x_t, y_t)$ for all pixels $(x_t, y_t) \in \Omega_i$. In the next stage we determine the minimum and maximum residuals, denoted by m_i and M_i for each $\Omega_i \in I$ and encode them along with all auxiliary information. We form a stream of symbols by concatenating the shifted residuals $\epsilon'(x_t, y_t) = \epsilon(x_t, y_t) - m_i$ for all regions and encode it by applying adaptive Markov arithmetic coding with order one. For a better compression, when a shifted residual $\epsilon'(x_t, y_t)$ is larger than an optimally determined value, M_{Res}, we encode the sequence $\{M_{Res}, s, r\}$, where s and r are the quotient and remainder of the division $\frac{\epsilon'(x_t, y_t)}{M_{Res}}$, respectively.

2.4. Encoding of region contours

The segmentation of an image can be transmitted by sending the contours separating the regions. The contour is transmitted using the 3OT chain-code encoded with adaptive-order Markov models [6].

Generally the collection of all borders between regions is formed of open contours which need to be grouped in order to obtain a total number of vertex chains as small as possible, because the transmission of starting points of a chain (which we also call anchors) is done with a high cost (assuming independent uniform distribution for their location in the image).

Here we introduce 5 different options of contour compression. The first option is the algorithm presented in [5]. The second option uses an algorithm which divides the contours into closed and open contours. For the segmentation obtained for an image not all contour points are interconnected because a region can be inside another region. For each 3OT chain-code (closed and open contour), we apply the adaptive Markov arithmetic coding algorithm with the determined optimal order. For the third option we use the idea of compressing only sequences of symbols from the alphabet $\{0, 1\}$. For that, from the initial sequence of 3OT chain-code representation we obtain two sequences of symbols: the first sequence encodes the position of the 3OT symbols $\{1, 2\}$, while the second sequence encodes the position of the 3OT symbol 2. Then for each sequence we apply the adaptive Markov arithmetic coding algorithm with the determined optimal order. The fourth and fifth options use the same principle as in the first and third options, with the difference that we apply the Arithmetic Coding Algorithm using the tree obtained by the *Context Tree Algorithm* [6]. Finally, the contour is compressed using the option which needs the lowest number of bits.

2.5. Algorithmic settings

In the version of the algorithm presented in [5], denoted here *Old Version* (OV), the lower bounds on region size are $\mathbf{N} = [N_1\ N_2\ N_3] = [55\ 45\ 30]$, and we keep these parameter set as a first alternative. Our extensive tests on a set of more than 1000 images have shown that for some images the compression ratio is better if one uses a second alternative set of parameters, $\mathbf{N} = [100\ 100\ 100]$, while for others it is better to use a third set of parameters, determined by exhaustive testing the combinations of the values $N_1 \in \{20, 30, \ldots, 80\}$, $N_2 \in \{20, 30, \ldots, 40\}$, and setting $N_3 = 0$. For this reason we have created a more complex version, denoted *New Complex Version* (NCV), that determines the compression ratio for all 3 alternative parameter sets and chooses the one that gives the optimal compression ratio. Due to the high complexity of the algorithm, we created another version of the algorithm, denoted *New Fast Version* (NFV), having the following features: uses for segmentation only the lower bounds $\mathbf{N} = [55\ 45\ 30]$; instead of 15 possible mixture predictors considers only the predictor with index $n = 7$; and chooses the best encoding of the region contours only from the first three options.

2.6. Experimental results

We illustrated the segmentation algorithm by presenting an example of segmentation in Figure 1. We note that the image is oversegmented, in the sense that the meaningful objects are split into many regions, since this split is convenient for getting the best compression within our scheme.

For illustrating the improvements with respect to our old algorithm [5] we first present the results for the same set of images from [8]. The comparisons with the standard JPEG-LS [7] compressor (using the implementation provided in [9]), with the PNG compressor (PNG being the format normally used for storing depth images in the public databases), and with the old version of the algorithm presented in [5] are illustrated in Table 2. The table shows that both new versions, NCV and NFV, consistently achieve significant improvements of the compression ratio.

The dataset from [8], also used in [10][11], contains 162 images classified by size in: large, medium and small, each having 27 images taken with camera in two positions, denoted (d1) and (d5). This gave us the opportunity to observe the behavior of the algorithm for different sizes of depth images. Compared to the old version program [5], the NCV algorithm produces improvements of the compression ratio with 477 936 bits or 0.08% for the entire set of 54 large images, 356 528 bits or 0.23% for the 54 medium images set, and 266 688 or 0.38% for the 54 small images set. In Table 3 we present resulting file sizes and compression factors (CF) for a set of 6 images, for each image type, where CF is defined as compressed size over initial size in percentage. The table shows that both new versions achieved an improvement of the compression factor and also that NCV algorithm shows an average improvement of 6% compared with the results of the CharLS algorithm. All results are checked for perfect reconstruction of the original file after decoding.

3. REFERENCES

[1] T. Kanungo, B. Dom, W. Niblack, and D. Steele, "A fast algorithm for MDL-based multi-band image segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition*, Seattle, pp. 609–616, June 1994.

[2] Y.G. Leclerc, "Constructing simple stable description for image partitioning," *International Jounal of Computer Vision*, vol. 3, pp. 73–102, 1989.

[3] Q. Luo and T. M. Khoshgoftaar, "Unsupervised multiscale color image segmentation based on MDL principle," *IEEE Transactions on Image Processing*, vol. 15, no. 9, pp. 2755–2761, 2006.

[4] S. Milani and G. Calvagno, "A depth image coder based on progressive silhouettes", *IEEE Signal Processing Letters*, vol. 17, issue 8, pp. 711–714, 2010.

[5] I. Şchiopu, and I. Tăbuş, "MDL segmentation and lossless compression of depth images", Workshop on Information Theoretic Methods in Science and Engineering, WITMSE 2011, pp. 55–58, August 7-10, Helsinki, 2011.

Image Name (disply)	Initial size (bits)	PNG size (bits)	CharLS size (bits)	OV size (bits)	NFV size (bits)	NCV size (bits)	ΔL_F size (bits)	ΔL_C size (bits)
$Art(d1)$	1370480	194736	212704	107080	102752	**100592**	4328	**6488**
$Books(d5)$	1370480	159560	152720	104208	95488	**95288**	8720	**8920**
$Dools(d1)$	1370480	272136	232496	177840	168856	**165880**	8984	**11960**
$Laundry(d1)$	1323120	168080	157824	102640	99648	**96032**	2992	**6608**
$Moebius(d1)$	1370480	193848	170272	109440	104256	**103744**	5184	**5696**
$Reindeer(d1)$	1323120	187112	174224	113768	109280	**107256**	4488	**6512**
$Lampshade2(d1)$	11544000	611288	673368	245768	239232	**238192**	6536	**7576**

Table 2. Results for the set of 7 images presented in [5], where $\Delta L_F = L_{OV} - L_{NFV}$, $\Delta L_C = L_{OV} - L_{NCV}$. With bold text are presented the best results.

Image Type	Image Name (disply)	Initial size (bits)	PNG size (bits)	CharLS size (bits)	OV size (bits)	NFV size (bits)	NCV size (bits)	CF CharLS (%)	CF NCV (%)
Large images	$Aloe(d5)$	11384160	797344	859104	346576	341648	**337136**	7.55	**2.96**
	$Bowling2(d5)$	11810400	799216	838752	315904	301256	**298456**	7.10	**2.53**
	$Cloth4(d5)$	11544000	735504	841248	284704	281984	**281152**	7.29	**2.44**
	$Flowerpots(d5)$	11650560	727840	884336	287176	277856	**274704**	7.59	**2.36**
	$Monopoly(d5)$	11810400	575728	687288	261152	253680	**250416**	5.82	**2.12**
	$Plastic(d5)$	11277600	466536	677912	198712	195752	**195728**	6.01	**1.73**
	$Wood1(d5)$	12183360	515632	793960	235496	227760	**223056**	6.52	**1.83**
Medium images	$Aloe(d5)$	2846040	330440	334592	167440	158824	**152048**	11.76	**5.34**
	$Bowling2(d5)$	2952600	326576	326536	148520	140032	**137376**	11.06	**4.65**
	$Cloth4(d5)$	2886000	303272	307432	134672	133304	**132528**	10.65	**4.59**
	$Flowerpots(d5)$	2912640	304320	344448	140000	134856	**132432**	11.83	**4.55**
	$Monopoly(d5)$	2952600	247776	280016	118416	113576	**112944**	9.48	**3.82**
	$Plastic(d5)$	2819400	204968	243136	99200	97024	**96216**	8.62	**3.41**
	$Wood1(d5)$	3045840	207984	306448	100640	95648	**92856**	10.06	**3.05**
Small images	$Aloe(d5)$	1263920	187472	194512	104808	94952	**93752**	15.39	**7.42**
	$Bowling2(d5)$	1311280	175632	178776	85888	81040	**79912**	13.63	**6.09**
	$Cloth4(d5)$	1281680	169352	168608	86440	85216	**84632**	13.16	**6.60**
	$Flowerpots(d5)$	1293520	168032	185944	85440	82584	**80288**	14.38	**6.21**
	$Monopoly(d5)$	1311280	130600	156864	66648	65176	**64264**	11.96	**4.90**
	$Plastic(d5)$	1252080	109544	129256	60360	57352	**55824**	10.32	**4.46**
	$Wood1(d5)$	1352720	107760	145096	55688	52880	**51208**	10.73	**3.79**

Table 3. Results and compression factors (CF) for a set of 6 images . With bold text are presented the best results.

[6] I. Tăbuş and S. Sârbu, "Optimal structure of memory models for lossless compression of binary image contours", IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2011, May 22-27, Prague, 2011.

[7] 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, issue 8, pp. 1309–1324, 2000.

[8] Repository *vision.middlebury.edu*: 2005 and 2006 Stereo Datasets [Online]. Available: *http://vision.middlebury.edu/stereo*.

[9] Source files of CharLS arhive [Online]. Available: *http://charls.codeplex.com/* Version 1.0.

[10] D. Scharstein and C. Pal, "Learning conditional random fields for stereo", IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2007, June 2007, Minneapolis, MN.

[11] H. Hirschmller and D. Scharstein, "Evaluation of cost functions for stereo matching", IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2007, June 2007, Minneapolis, MN.

Publication 2

First published in the Proceedings of the 20th European Signal Processing Conference (EUSIPCO-2012) in 2012, published by EURASIP. Reprinted, with permission, from

[P2] I. Schiopu and I. Tabus. Lossy and Near-Lossless compression of depth images using segmentation into constrained regions. In *Proc. European Signal Processing Conference (EUSIPCO)*, pages 1099–1103, Bucharest, Romania, August 2012.

Copyright ©EURASIP, 2012.

LOSSY AND NEAR-LOSSLESS COMPRESSION OF DEPTH IMAGES USING SEGMENTATION INTO CONSTRAINED REGIONS

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

This paper presents a lossy coding method for depth images using a segmentation constructed by selecting regions of pixels having the depth values obeying constraints defined in terms of some bounds, which are tuned in order to obtain the target distortion. The contours describing the segmentation are transmitted using an efficient chain coding method and are thus available also at the decoder for the next stage, which is region based predictive coding with a tunable precision level. The rate comprises the cost of losslessly transmission of the contours and the cost of transmitting the residuals with the decided precision, which is the main factor influencing the distortion. We introduce a procedure optimizing the parameters involved in the segmentation and in the prediction for a given image. As a side result, the segmentations residing on the convex hull of the RD curve can be seen as optimal segmentations with various granularity.

Index Terms— lossy compression, near-lossless compression, depth image, segmentation, rate-distortion

1. INTRODUCTION

The subject of depth compression has received increased attention recently, mostly due to the wide range of applications for 3D representations, in computer vision, 3DTv, and computer games. Applying the same compression methods for depth images as for the gray-level or color pictures is not as efficient as designing new methods, dedicated to the types of regularities present in depth images. In [1] we have shown that dedicated lossless compression methods can reduce to half the size of compressed files produced by the standard JPEG-LS image compressor.

The literature on lossy depth image compression is wide, mostly in connection to the compression of multiview images, where the interesting bit-rates are in the very low end, even below 0.1 bpp, see e.g. [2][3] and reference lists therein. Our method addresses higher rate ranges aiming at near-lossless coding, with prior work existing in, e.g., [4] [5].

2. THE PRINCIPLE OF THE METHOD

The basic principle of constructing the segmentation is to split the image into two types of regions: for the first type, a region "with local variability λ" should contain any pixel which has inside the region at least one neighbor such that the difference between the depth values of the pixel and its neighbor is at most equal to a given threshold λ, and additionally the size of the region is also constrained. In the case of regions of second type, from the regions with local variability $\lambda = 1$ are selected those which have also global variability 1, i.e., they contain pixels with only two distinct depth values. The regions obeying the local constraint variability condition may have quite a diverse distribution of depth values, e.g., planar sections starting with one side close to the camera, with a low depth level and ending on the other side with a very high depth value; as a different example the regions are encompassing also second order surfaces, typical in the case of round, spherical, cylindrical, or conical objects. By varying the threshold λ the sizes of the regions will change. The process of finding the regions is iterative, starting with finding the connected regions at small thresholds and if they are large enough they are declared regions, and then the process continues at larger thresholds. The values of the thresholds and the lower bound of the region size for declaring a region are parameters determining many possible partitions of the image into regions. When the parameters determine a rough granularity, the cost of transmitting the contours of regions is small. Here we use chain codes which are very cheap ways of transmitting losslessly the contours, and hence we do not resort to parametric models for coding the contours in a lossy manner, which is the option followed in most of the previous lossy coding methods.

After the regions are defined, in each region we use lossy predictive coding, where the prediction is performed based on the reconstructed depth of the previously transmitted pixels and the quantization of the prediction residuals is uniform, with a tunable step size $2\eta + 1$, for all regions except those regions which contain two or three distinct depth values. The parameter η defining the quantization steps belongs to the set $\{0, 1, 2, 3\}$, where $\eta = 0$ means no quantization, in which

© EURASIP, 2012 - ISSN 2076-1465

case the compression is lossless for the involved regions. The regions having only two or three distinct depth values are treated differently; in each of them only one reconstruction value is chosen (the one minimizing the sum of square errors inside the region) and then is transmitted to the decoder.

When the targeted bitrate is in the low range (below 0.2 bpp), we introduce an additional preprocessing stage: before segmentation stage the last bit of the depth values is removed, the average of all these removed bits is computed, and if the average is larger than 0.5, a bit of one is appended as a least significant bit of the final depth values obtained after the reconstruction at the decoder. In this way the range of depth values is halved and only a rough reconstruction of the last bit is performed, by the majority bit.

In the following section we present the algorithmic design of the segmentation and the way to combine the lossless contour compression and the lossy prediction residuals encoding. The obtained results are compared with [4].

The depth image contains the depth value $I(x,y) \in \{0, 2^B - 1\}$ for each pixel (x,y). For illustrations we use as input image the same one used in [4], namely the first frame of view 1 from the *Breakdancing* sequence [6], which has the number of bit-planes $B = 8$. We will define the segmentation as the union of all regions, $\Omega_1, \ldots, \Omega_{n_r}$, that make up the depth image.

3. ALGORITHM DESCRIPTION

3.1. Generating a segmentation

The main problem for obtaining the best results is to generate a suitable segmentation of the image so that after lossless contour compression the decoder knows enough distinct regions and with an additional bitstream containing the prediction residuals it can obtain small overall distortion using a low bitrate. Our solution is an iterative segmentation which allows a different variability inside some regions, while for other regions, beside imposing the fixed variability $\lambda = 1$, it is additionally required that the overall number γ of distinct depth-levels is small. At a given step of the algorithm all connected components which contain more than N_λ (or K_γ) pixels are declared regions. The process can be characterized by the maximum allowed variability λ with its associated minimum size N_λ of a region, and the constrained number γ of distinct depth-levels with its associated minimum size K_γ.

For a current pixel (x_t, y_t) the set of the four neighbors in 4-connectivity is denoted $\mathcal{N}_4(x_t, y_t)$. The variability of the current pixel (x_t, y_t) inside a given region Ω_j is defined as the minimum value of the absolute differences of the depth values between the current pixel and those neighbors which are part of the region [1], as follows:

$$V(x_t, y_t) = \min_{(x_i, y_i) \in \mathcal{N}_4(x_t, y_t) \cap \Omega_j} |I(x_t, y_t) - I(x_i, y_i)|.$$

The segmentation algorithm starts by finding the sets of connected pixels having variability $\lambda = 0$ and declaring them candidate regions. In the next stage each candidate region having at least K_γ pixels is declared a region Ω_j of the segmentation, and the remaining pixels, not yet in the already decided regions, are then grouped in connected components that have variability at most $\lambda = 1$ and a number of distinct depth-levels $\gamma = 2$. Using these constraints one can generate regions that have only two consecutive depth-levels that are compressed using a single reconstruction depth level, the one which gives the minimum distortion. In the next step we iterate for successive thresholds on variability $\lambda \in \Lambda = \{1, 3\}$. At each iteration step, each candidate region from the previous step having at least N_λ pixels is declared a region Ω_j of the segmentation, and the remaining pixels not yet in already decided regions are then grouped in connected components and all such components with variability not exceeding λ are declared candidate regions for the next step. At the last step all candidate regions are automatically declared regions of the segmentation. According to the definition of the variability constrained regions, the obtained segmentation is unique. The regions with size smaller than 5 pixels, which make a large proportion of the whole number of regions, are merged with the largest neighbor region because of the high cost of transmitting them separately. This ensures the reduction of the contour length, and even more importantly, the elimination of some points in the contour with more than two contour-edge intersections, points that are required to be transmitted separately as anchor points and which require a large number of bits for encoding.

We consider in the experiments two versions of the segmentation, the first L-CRS using the merging of the very small regions, the second, NL-CRS keeping the small regions as part of the segmentation.

3.2. Quantization and encoding of regions with almost constant depth

The obtained regions which have a small number, γ, of distinct levels are quantized and encoded in a simpler manner than the rest of the regions. This simple quantization and encoding is used because a near-lossless quantization could have set the regions with a depth-level that produces a large distortion.

In each region that has $\gamma = 2$ distinct depth levels, g and $g+1$, the quantized depth value is set to that value, g^*, which occur the most often among the two consecutive levels, this process being equivalent to the reconstruction minimizing the distortion.

In each region that has $\gamma = 3$ distinct depth levels, first the mean square distortion after quantizing by the optimal level is computed, and if the resulting PSNR is smaller than a threshold T_3 then the region is quantized and encoded using the predictive method presented in the next section, otherwise it

d	c	
a	x	

Table 1. The prediction neighborhood $\mathcal{N}_P$ of the current pixel (x_t, y_t), which is marked by "x". Also shown are the letters used for the depth values of the neighbors.

is processed similarly to the case $\gamma = 2$, where only the optimal quantization level is encoded and used as a reconstruction at the decoder.

3.3. Local nonlinear prediction

We predict the depth $I(x_t, y_t)$ for a current pixel $(x_t, y_t) \in \Omega$ by using the reconstructed values $\tilde{I}(x_i, y_i)$ of the pixels $(x_i, y_i) \in \Omega$ which also belong to a causal neighborhood $\mathcal{N}_P(x_t, y_t)$ of the pixel (x_t, y_t), depicted in Table 1. We denoted by $\tilde{I}(x, y)$ the value available at the decoder, obtained using the quantized prediction residuals. Similarly as in [1], for each region the horizontal scanning and the vertical scanning are tested, both with causal neighbors (a, c, d), and the one giving better performance is selected and announced to the decoder by one bit per region. Although both scanning orders use the same causal neighborhood, different quantized residues are obtained for each scanning order and hence the two compression ratios for a region are different.

In [1] we used an optimal predictor selected among 15 mixture predictors, $\hat{I}_n(\mathcal{N}_P(x_t, y_t)), n = 1, \ldots, 15$. Here the optimal predictor is taken the one with index $n = 1$, which gave the best results in our tests. Hence, the collection of elementary predictions of the nonlinear predictor, denoted $\mathcal{P}(\mathcal{N}(x_t, y_t))$, is $\mathcal{P}(\mathcal{N}(x_t, y_t)) = \{a,\ c,\ c + a - d\}$. If one of the used neighbors are not in the causal neighborhood, the elementary prediction is eliminated from $\mathcal{P}(\mathcal{N}(x_t, y_t))$.

Consequently, in this paper the prediction is calculated as follows: $\hat{I}_n(\mathcal{N}(x_t, y_t)) = median\{\mathcal{P}_n(\mathcal{N}(x_t, y_t))\}$.

3.4. Encoding of quantized prediction residuals

The encoding of the pixels in any region Ω_i having k_i pixels is performed as follows. First determine the prediction value $\hat{I}(x_t, y_t)$ for each pixel (x_t, y_t) in the region, with $t = 1, \ldots, k_i$. We define the residuals $\epsilon(x_t, y_t) = I(x_t, y_t) - \hat{I}(x_t, y_t)$ for all pixels $(x_t, y_t) \in \Omega_i$. In the next step we quantize the prediction residuals $\epsilon(x_t, y_t)$ using uniform quantization. Same as in [7], the quantizer is defined as:

$$Q(\epsilon) = sign(\epsilon) \left\lfloor \frac{|\epsilon| + \eta}{2\eta + 1} \right\rfloor,$$

where the signum function returns 1 for positive argument, -1 for negative and 0 for 0 argument. The reconstructed value, used also by the encoder, is obtained as follows:

$$\tilde{I} = \hat{I} + Q'(\epsilon) \cdot (2\eta + 1),$$

where the reconstruction level is biased from the midpoint of the quantization interval towards zero, to account for the typical monotonic decreasing pdf of the absolute value of the residual:

$$Q'(\epsilon) = Q(\epsilon) - \frac{sign(Q(\epsilon)) \cdot \mu(\eta)}{2\eta + 1}.$$

The tests showed that the best results are obtained for the bias term corresponding to $\mu(\eta) = \eta$.

In the next stage we determine the minimum and maximum quantized residuals, denoted by m_i and M_i for each $\Omega_i \in I$ and encode them along with all auxiliary information. We form a stream of symbols by concatenating the shifted residuals $\varepsilon'(x_t, y_t) = Q(\epsilon(x_t, y_t)) - m_i$ for all regions and encode it by applying adaptive Markov arithmetic coding with order one. Like in [1], for a better compression, when a shifted quantized residual $\varepsilon'(x_t, y_t)$ is larger than an optimally determined value, M_{Res}, we encode the sequence $\{M_{Res}, s, r\}$, where s and r are the quotient and remainder of the division $\frac{\varepsilon'(x_t, y_t)}{M_{Res}}$, respectively.

3.5. Encoding of region contours

The segmentation of an image is defined by contours separating the regions. The contour is transmitted using the 3OT chain-code representation [8].

From the five options presented in [1], option four obtained the best results in our tests. For other images the optimal option could be any other. The fourth option was encoding the chain code by applying the Arithmetic Coding Algorithm using the optimal context tree obtained by the *Context Tree Algorithm* [8]. Depending on the required bitstream, the algorithm generates a specific segmentation which has a different contour. For each segmentation we usually obtain a different tree-depth for the context tree: a height tree-depth $(18 \div 20)$ for high bitrate (almost lossless), and a lower tree-depth $(14 \div 16)$ for low bitrate, see column 5 of Table 2.

3.6. Summary of the algorithm

The segmentation algorithm implemented, denoted Lossy Constrained Region Segmentation (L-CRS), can be summarized in a few steps:

1. Smooth the contour by eliminating some edges between pixels with similar values as follows: if at least 3 neighbors in $\mathcal{N}_4(x_t, y_t)$ have the same depth value, v, and if $|I(x_t, y_t) - v| \leq 2$, then set $I(x, y) = v$;
2. Determine the connected sets of pixels having variability $\lambda = 0$ (thus having constant depth);
3. Select the sets of pixels with cardinality larger than K_2 and declare them regions; out of the remaining pixels, determine connected sets of pixels having variability at most $\lambda = 1$ and having the maximum number of distinct depth-values $\gamma = 2$;

4. Select the sets of pixels with cardinality larger than N_1 and declare them regions; out of the remaining pixels, determine connected sets of pixels having variability at most $\lambda = 1$;
5. Similar to (3), but with cardinality larger than $N_3 = 100$ and using variability $\lambda = 3$;
6. Declare the remaining connected sets of pixels regions;
7. Set pixel regions with size smaller than 5 pixels with the depth level of the biggest neighboring region;
8. Encode the region contours using 3OT chain-codes;
9. Quantize and encode the regions with almost constant depth, using $T_3 = 50$ ($T_3 = 45$ if $low\_bitrate = 1$);
10. For the remaining regions use the near-lossless predictive compression as explained in Sections 3.3 and 3.4.

Besides the fixed parameters, having the value specified in the algorithm, the values of the parameters M_2 and N_1 are the most important and have to be chosen according to the desired bitrate. Both of them can take values from 50 to the size of the biggest region in the image. For example to obtain a compression with a large PSNR, one can set $M_2 = 50$, $N_1 = 50$ and use $eta = 0$, so that only a few regions contains distortion, while for a low bitrate one can set $low\_bitrate = 1$, $M_2 = 1000$, $N_1 = 14000$ and use $eta = 2$.

For a near-lossless compression, meaning that the absolute value of the error is smaller or equal than $2\eta + 1$, we introduced another method, denoted Near-Lossless Constrained Region Segmentation (NL-CRS), using the same algorithm as L-CRS with the difference that we eliminate step (7).

3.7. Experimental results

The segmentation algorithm is illustrated by presenting the segmentation result in Figure 1 (b) for the depth image from Figure 1 (a). We note that the image is oversegmented, in the sense that the meaningful objects are split into many regions, since this split is optimal for our rate-distortion optimization scheme.

We present the results in a rate-PSNR plot, where the bitrate is calculated as bits per pixel (*bpp*),

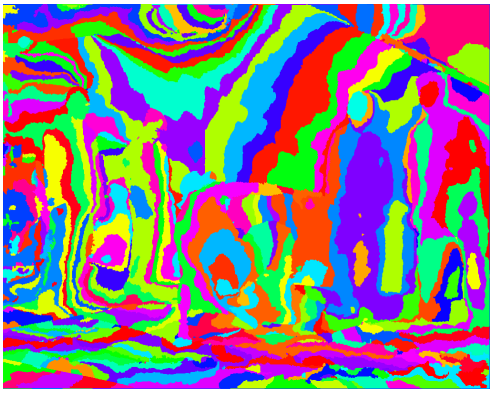
$$bpp = 8 \cdot \frac{compreesed\_file\_size}{initial\_file\_size},$$

and the peak signal-to-noise ratio, $PSNR$, is computed as:

$$PSNR = 10 \cdot \log_{10} \frac{255^2}{MSE}.$$

We compared the results for the two methods introduced, L-CRS and NL-CRS, with JPEG2000 and the two other methods from [4] and [5], denoted here Method 1 and Method 2. Figure 2 shows the results for the five methods using one image from the Breakdancing sequence. One can see that our

(a) Initial depth image



(b) Obtained segmentation

Fig. 1. Example of segmentation for a low bitrate of first frame of view 1 of Breakdancing sequence.

methods obtain better results compared with the best existing results, Method 1 [4]. Because L-CRS is generated from a lossless method, the transition from lossless to lossy is steep. Another factor is that the bitrate has two parts: we compressed losslessly the region contours and lossy the depth-levels inside the regions, that is why for a low bitrate the results will asymptotically reach the point where most of the bitrate will be composed of contour lossless bitrate.

Figure 2 presents also the result of lossless compression using the more complex algorithm from [1]. The result is presented using a vertical asymptote at $bitrate = 0.3933$ *bpp*, which is the point where $PSNR = \infty$.

In Table 2 we take a closer look at some statistics of the segmentation and the proportions of bitrates needed for lossless compression of contours and lossy compression of depth values for 8 functioning points from the L-CRS curve pre-

	Image Segmentation				Q.	LP	Contour comp.		Depth-level comp.		L-CRS	
Nr.	low bitrate	final nr. of reg.	contour length	maximum tree-depth	η		bitrate (bpp)	% of total	bitrate (bpp)	% of total	Total bitrate	PSNR (db)
1	0	2575	142949	18	0	461	0.2265	64.48	0.1248	35.52	0.3513	61.9048
2	0	1635	140289	20	1	579	0.2223	82.80	0.0462	17.20	0.2685	57.9007
3	0	1483	134155	20	1	568	0.2120	84.39	0.0392	15.61	0.2512	56.9242
4	0	1269	124382	18	1	474	0.1957	84.86	0.0349	15.14	0.2307	55.3445
5	0	1133	112289	18	2	477	0.1775	88.31	0.0235	11.67	0.2010	53.1608
6	1	598	80145	18	1	193	0.1223	91.37	0.0116	8.63	0.1339	48.2914
7	1	536	70201	18	1	147	0.1078	92.52	0.0087	7.48	0.1166	46.7343
8	1	432	62313	17	3	147	0.0952	93.19	0.0070	6.81	0.1022	45.0466

Table 2. Examples of functioning points on the L-CRS curve from the rate-PSNR plot, with their details regarding the segmentation, the quantization, and the lossy and lossless composition of compressed image. The size of the images is 1024×768 and has 6690 initial constant regions.

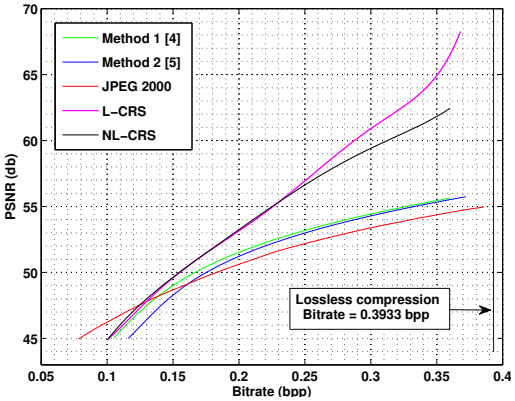


Fig. 2. Lossy depth image compression comparison of our two implemented methods, L-CRS and NL-CRS, with JPEG 2000 and the two methods from [4] and [5], for the first frame of view 1 (camera 0) of the Breakdancing sequence.

sented in Figure 2. The table presents also the number of pixels, denoted lossy pixels (LP), for which we do not impose a near-lossless compression which means that the absolute value between initial depth-level and the reconstructed value is greater than $2\eta + 1$.

In Figure 1 (b) we presented the segmentation obtained for the 6^{th} point in Table 2. One can see that some of the 598 final regions are very small, especially in the bottom of the image. The final segmentation contains these regions because the depth has a great variation in this area and putting them together produces worse results, by increasing significantly the distortion, with just a small decrease of the bitrate.

4. REFERENCES

[1] I. Schiopu and I. Tabus, "Depth image lossless compression using mixtures of local predictors inside variability constrained regions," in *Int. Symposium on Communications, Control, and Signal Precessing*, Rome, May 2012.

[2] 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 Int. Conf. Image Processing*.

[3] W.-S. Kim, A. Ortega, P. Lai, D. Tian, and C. Gomila, "Depth map distortion analysis for view rendering and depth coding," in *Proc. 16th IEEE Int Image Processing*, 2009, pp. 721–724.

[4] C. Dal Mutto, P. Zanuttigh, and G. M. Cortelazzo, "Scene segmentation by color and depth information and its application," in *STreaming Day*, Udine, Italy, Sept. 2010.

[5] P. Zanuttigh and G.M. Cortelazzo, "Compression of depth information for 3D rendering," in *3DTV Conference: The True Vision-Capture, Transmission and Display of 3D Video*, Potsdam, Germany, May 2009, pp. 1–4.

[6] 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*, Los Angeles, 2004, pp. 600–608.

[7] 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, 2000.

[8] I. Tabus and S. Sarbu, "Optimal structure of memory models for lossless compression of binary image contours," in *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Prague, May 2011, pp. 809–812.

Publication 3

Copyright ©2013 IEEE. Reprinted, with permission, from

[P3] 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 (IEEE TIP)*, 22(11):4195–4210, November 2013.

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Tampere University of Technology's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

Context Coding of Depth Map Images under the Piecewise-Constant Image Model Representation

Ioan Tabus*, *Senior Member, IEEE,* Ionut Schiopu, *Student Member, IEEE* and Jaakko Astola, *Fellow, IEEE*

***Abstract*—This paper introduces an efficient method for lossless compression of depth map images, using the representation of a depth image in terms of three entities: the crack-edges, the constant depth regions enclosed by them, and the depth value over each region. The starting representation is identical with that used in a very efficient coder for palette images, the piecewise-constant image model (PWC) coding, but the techniques used for coding the elements of the representation are more advanced and especially suitable for the type of redundancy present in depth images. First the vertical and horizontal crack-edges separating the constant depth regions are transmitted by two-dimensional context coding using optimally pruned context trees. Both the encoder and decoder can reconstruct the regions of constant depth from the transmitted crack-edge image. The depth value in a given region is encoded by utilizing the depth values of the neighboring regions already encoded, exploiting the natural smoothness of the depth variation and the mutual exclusiveness of the values in neighboring regions. The encoding method is suitable for lossless compression of depth images, obtaining compression of about 10 to 65 times, and additionally can be used as the entropy coding stage for lossy depth compression.**

EDICS Category: COM-LLC Lossless Coding of Images and Video

I. Introduction

Recently there has been an increased interest in the compression of depth map images, especially due to the appearance of many types of sensors or techniques for acquiring this type of images, and also due to their wide range of applications, starting from generating multi-view images in 3DTv, to computer vision applications, and gaming. Since the quality of the acquired images improves all the time, there is an interest in developing techniques which can compress these data preserving all the information contained in the originally acquired images.

The depth map images are formally similar to the natural gray level images, at the pixel with coordinates (x, y) being stored an integer value $D(x, y) \in \{0, \ldots, 2^B - 1\}$ using B bits, with the major difference being the significance of the image values. For natural gray level images $D(x, y)$ represents the luminance of the object area projected at the (x, y) pixel, while for depth images $D(x, y)$ represents the value of the depth or distance from the camera to the object area projected in the image plane at the coordinates (x, y).

Manuscript accepted 14 June 2013

Copyright (c) 2013 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

I.Tabus, I. Schiopu, and J. Astola are with the Department of Signal Processing, Tampere University of Technology, Tampere, Finland. Corresponding author: Ioan Tabus, e-mail: ioan.tabus@tut.fi (see http://www.cs.tut.fi/~tabus)

Unlike the case of natural gray-scale images, which represent the luminance of natural images, in most depth images there are many large regions of constant values, hence leading to a higher redundancy, and thus to potentially much better compression factors for depth images as compared to natural gray scale images. Apart of the large constant regions, depth images may contain also medium size, and even very small regions. Some of the small regions may represent only noise, some of them may represent also valuable geometric information about the contours of the objects present in the depth image. When using lossy compression of the depth images for such applications as multi-view generation from depth-plus-one-view, it was noticed that preserving the geometry of the object contours is very important for achieving good quality of the multi-view reconstructions. Hence the removing of the small objects, or redrawing of the contours due to use of quantization (possibly after applying an image transform), can lead to disturbing artifacts in the multi-view reconstruction. One possible remedy is the transmission of the lossless version of the depth image, if it compresses well, or the transmission of lossy versions that are as close as possible to the original. In this paper we propose an efficient lossless compression method, which can also be used as entropy coder for several lossy approximations of the image, providing a wide range of available rates in the rate-distortion plane, wider than most existing lossy compression methods, bringing potentially more flexibility in a multi-view generation system.

A. Preview

This paper provides a tool for encoding the representation of a depth image in terms of the underlying partition of constant regions (which we call patches in the rest of the paper), focusing on efficient methods to encode the contours and the depth-value of each patch.

Our method has a number of similarities with the method named "the piecewise-constant image model" (PWC) [1] which was shown to be most suited for palette images. Both PWC and our method, CERV, start from the initial representation of the image, in terms of binary vertical and horizontal edges of the region contours, and in terms of the depth value over each region. We dedicate Section VI to present the algorithmic similarities and differences between the two methods, after presenting in detail the content of our method.

A crack-edge can be seen as a virtual line segment separating two pixels, which are vertical or horizontal neighbors (see the green and red segments in Figure 1 and further details in

Section II-A). It has associated the value zero if the separated pixels have the same value and one otherwise, and hence can be used to describe the contours between constant regions. We use two-dimensional contexts for encoding all crack-edges in the image. Out of all crack-edges only the active crack-edges (those which are set to one) are important because they form the contours of the constant regions.

Once the constant regions in the depth map image are reconstructed from the encoded contours, filling in a depth value for each constant region is done exploiting the smoothness of variation of depth value from one region to another. However, unlike in the case of natural images, where one pixel is predicted from its (small) causal neighborhood, which is bound to a few of its neighbors in the pixel grid, now a region may have a more varied neighborhood of regions, sometimes one region being engulfed into another one, and thus having just one single neighbor, or in the other extreme, one large region may have hundreds of neighboring small regions. Prediction in this network of regions, with variable structure of their neighborhood, will lead to encoding distributions with very low entropy.

The depth value at a current patch is encoded in several different ways, depending on the values at the neighbor patches. The efficiency comes especially from the situations when the depth values in the already known patches are close to the depth of the patch to be encoded, which additionally ought to be different of all depth values of its neighboring patches. In these situations a list of possible candidate values for the depth of the current patch is constructed, and the rank of the current depth in the list is transmitted, taking into account the exclusions built in the list. It turns out that the current depth value comes most of the times in the top few positions, and the rank information can thus be transmitted very efficiently.

In this paper we attempt to split the image into elements denoted generically $\{\xi_i\}$ which can be: 1) vertical crack-edges, $V_{i,j}$, 2) horizontal crack-edges, $H_{i,j}$ and 3) depth values $D_{i,j}$. The challenge is to utilize in the most efficient way the redundancy between these elements and find the most effective strategy of transmitting them. The order in which we transmit is essential and both the encoder and decoder should be able to construct and use in the arithmetic coding [2], [3] the conditional probabilities $\mathbb{P}(\xi_i|\xi_{j_1}, \xi_{j_2}, \ldots; j_1, j_2, \ldots < i)$ in a causal way with respect to the order of transmitting the elements. Out of the transmitted elements we may also construct entities like regions of pixels, or clusters of available neighbor values, which can be used when defining the conditional probabilities.

We introduce a family of algorithms for encoding depth images, dubbed CERV (from crack-edge–region–value), having configurable algorithmic complexity, where at one end the encoding is done very fast, and on the other end the encoding takes longer time in order to exploiting more intricate redundancies. In the fast encoding variant, CERV-Fast, the encoding of the contours and depth values are done intertwined, line by line, in a single pass through the image, while in the higher compression (CERV-HiCo) variant (which also has higher complexity) the patches over the whole image are found and encoded first, followed by encoding the depth values. The two variants differ also in the type of regions they use: the CERV-HiCo variant uses patches, which are globally maximal regions, while the CERV-Fast variant uses locally determined constant segments as regions, for each such a region one depth-value being encoded. Hence the essential ingredients in the CERV representation and in the lossless coding scheme are: the crack-edges (CE), the regions (R), and the values over regions (V).

The algorithms can be used directly for lossless coding of depth images. Additionally, the CERV algorithms can be used as entropy coding blocks for lossy coding methods, where one has to design a first processing block having the task to build a lossy version of the image, suitable for the type of encoding we propose. One very simple version of such a scheme is illustrated in this paper, which together with the CERV algorithm constitute a lossy coding method, displaying very good results at high bitrates, and, for some images, surprisingly competitive results even at low bitrates.

B. Related work

The recent work relevant for lossless depth image compression has proposed several algorithms specifically conceived for depth images and additionally it considered modifications of the methods currently used for color or gray-scale images. In [4] the image is first split into blocks, and the initial binary planes are transformed according to Gray coding and then encoded using a binary compression scheme. Encoding by bit-planes is further developed in [5], where the transformed binary planes are encoded by JBIG and, additionally, the problem of efficiently encoding a pair of left- and right-disparity images is solved. In [6], [7] only the contours of the large regions resulted from a segmentation of the image were transmitted, using chain-codes, after which predictive coding of the various depth-values was used inside each region. Other line of research in lossless depth coding refers to modifying the traditional lossless image coders for making them more suitable for depth images coding. The lossless mode of the H264/AVC standard was modified in order to cope better with smoother images, with results presented in [8] and [9].

As a small departure from proper lossless compression, which ensures perfect recovery of the original, there are also several recent papers that discuss lossy encoding with near-lossless [10], [11], or rendering lossless capabilities [12].

Lossless depth image compression is essentially related to other areas of image coding, perhaps the closest being the coding of palette images, of color map images, and of region segmentations.

The most efficient coder for palette images is the already mentioned method PWC [1], while another coder specially designed for palette images [13] was also used in the past for encoding segmentation images.

The particular technical solutions used in CERV can be traced to a number of past contributions. The context coding of the binary crack-edges can be seen as a constrained subclass of encoding a binary image. General encoding of binary images was dealt with by the ISO standard Joint Bilevel Image Experts

Group (JBIG [14] and JBIG2 [15]) providing good performance for a wealth of applications. More elaborated coding schemes have proposed context trees where the selection and the order of the pixels in the context is subject to optimization, in an adaptive way [16]. Using context trees in a semi-adaptive way was proposed in [17] by using a fixed context, similar to the way it is done in the algorithm CERV.

Encoding the boundaries of regions is one of the problems needed to be solved in seemingly distant applications: in MDL based segmentation of images [18] [19], where the code length for transmitting the segmentation is one term in the MDL criterion; in object based coding, for transmitting the boundary of objects or the regions of interest [20] [21]. Chain codes representations and their associated one-dimensional memory models were very often used for the specific problem of contour coding of segmentations. Context tree coding of contours in map images was discussed in [17]. The predictive coding of the next chain-link based on a linear prediction model was presented in [22]. A general memory model based on semi-adaptive context trees was presented in [23]. PPM using a 2D template was used in [24]. The precursor of PWC method for contour coding is the 2D context coding of [25].

The more general topic of lossy depth map coding received a lot of attention recently. Two of the most efficient techniques were published in [26] and [27]. The precision of the boundary information was seen as an important factor which needs to be improved in [28]. Piecewise linear approximations were used for selecting the important contours in [29], while encoding the contours was realized by JBIG2.

Embeded coding of depth images was recently introduced in [30], where R-D optimization is used for combining sub band coding with a scalable description of the geometric information.

II. Representation by crack-edges plus depth over constant regions

A. Depth image, horizontal crack-edge image, vertical crack-edge image

The depth image can be conveniently represented by two groups of elements, possessing very specific redundancies: first the set of crack-edges (which defines the contours enclosing sets of pixels having the same depth value) and second, the collection of depth values over all constant regions.

The depth image to be compressed is $\mathcal{D} = \{D_{i,j}\}_{i=1,\ldots,n_r,j=1,\ldots,n_c}$, where n_r is the number of rows and n_c is the number of columns. The crack-edges can be defined using indexing in the rectangular $n_r \times n_c$ grid, and we introduce a binary image of vertical crack-edges, $\mathcal{V} = \{V_{i,j}\}_{i=1,\ldots,n_r,j=2,\ldots,n_c}$ with $V_{i,j} = 1$, if $D_{i,j-1} \neq D_{i,j}$, and $V_{i,j} = 0$ otherwise. Similarly, the horizontal crack-edge image $\mathcal{H} = \{H_{i,j}\}_{i=2,\ldots,n_r,j=1,\ldots,n_c}$ is defined by $H_{i,j} = 1$ if $D_{i-1,j} \neq D_{i,j}$ and $H_{i,j} = 0$ otherwise. Hence, a vertical crack-edge refers to the left side of the current pixel, telling if the depth of the current pixel and that of its left neighbor are different. For illustration purposes we visualize the pixel $D_{i,j}$ as the interior area of a square (see Figure 1), the crack edge $V_{i,j}$ as the left side of the square, and $H_{i,j}$ as the top side of the square, remaining basically in the same $n_r \times n_c$ grid of indices, except that the first row of horizontal edges $H_{1,1}, \ldots, H_{1,n_c}$ and the first column of vertical edges $V_{1,1}, \ldots, V_{n_r,1}$ do not contain any useful information, and it is not necessary to be encoded. We consider here as type of connectivity for pixels only 4-connectivity, two pixels being neighbors only if they are neighbors on the same row ($D_{i,j}$ and $D_{i,j+1}$) or on the same column ($D_{i,j}$ and $D_{i+1,j}$). The crack-edge images are obtained by scanning the image in *row-wise* order, and checking the inequalities $D_{i,j-1} \neq D_{i,j}$, which results in the $n_r \times (n_c - 1)$ crack-edge image $\mathcal{V} = \{V_{i,j}\}$ and checking the inequalities $D_{i-1,j} \neq D_{i,j}$, which results in the $(n_r - 1) \times n_c$ horizontal crack-edge image $\mathcal{H} = \{H_{i,j}\}$, both images being binary valued.

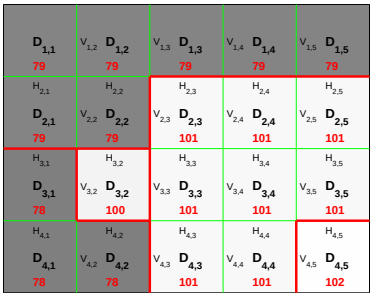


Fig. 1. The lattice of pixels with depth values $D_{i,j}$ (with numerical value printed in red), having associated vertical crack-edges $V_{i,j}$ and horizontal crack-edges $H_{i,j}$. The crack-edges that are set to one are called active and are represented in red, the crack-edges set to zero are inactive and represented in green. In this example there are five patches separated by the active crack-edges.

B. Splitting the image into patches (regions with same depth)

A patch P is defined as a maximal region connected in 4-connectivity in the initial image $\mathcal{D}$, containing pixels with the same depth value, D, which is called the depth value of the patch, D_P. The interior part of the patch is separated from the exterior part by crack-edges set to 1 (dubbed active crack-edges), which form an uninterrupted chain. In Figure 1 there are five patches, e.g. the patch $P_1 = \{D_{1,1}, D_{1,2}, \ldots, D_{1,5}, D_{2,1}, D_{2,2}\}$ is separated from the rest of the image by the chain $\mathcal{C}(P_1) = [H_{3,1}, H_{3,2}, V_{2,3}, H_{2,3}, H_{2,4}, H_{2,5}]$ which is called contour of the patch (not including the outer crack-edges $H_{1,1}, \ldots, H_{1,5}, V_{1,1}, V_{1,2}$, which need not be encoded).

The collection of all patches in the image is $\mathcal{P} = \{P_1, \ldots, P_{n_P}\}$ and can be constructed at the decoder starting from the images $\mathcal{H}$ and $\mathcal{V}$, while at the encoder the set of patches can be obtained already while scanning the image for setting the values in $\mathcal{H}$ and $\mathcal{V}$. Algorithmically, constructing the patches is done by checking the four neighbors (in 4-connectivity) of each pixel belonging to a patch, and labeling them as members of the same patch if they have the same depth-value (or at decoder, testing if the candidate neighbors are not separated from the patch by active crack-edges). When all pixels in a patch are tested and no more growing of the

patch occurs, a pixel not belonging yet to any patch is used to start a new patch and the growing of the new patch is continued in a similar manner.

C. Characterizing the set of patches

The patches are indexed in such a way that both the encoder and the decoder can identify and process the patches in the increasing order of their indices, e.g in the order of reaching the patches when we scan in row-wise order the pixels in $\mathcal{D}$.

For the following descriptions each patch P_ℓ has associated a number of features: depth, D_{P_ℓ}, patch size, $|P_\ell|$ (i.e. number of pixels in the patch), its contour $\mathcal{C}(P_\ell)$, and its set of neighbor patches $\mathcal{N}(P_\ell)$ having cardinality $|\mathcal{N}(P_\ell)|$.

A patch P_{ℓ_2} is a neighbor of a patch P_{ℓ_1} ($P_{\ell_2} \in \mathcal{N}(P_{\ell_1})$) if their contours have a common active crack-edge. The depth values of the two neighbor patches are necessarily distinct, $D_{P_{\ell_1}} \neq D_{P_{\ell_2}}$.

Some typical statistics concerning the crack-edges, the patch sizes, and number of neighboring patches are shown in Table I for a set of six depth images, which will be used for exemplifications throughout this paper and will be described in more details in the experimental section.

III. Encoding the images of crack-edges

The two images of crack-edges $\mathcal{H}$ and $\mathcal{V}$ can be encoded in two alternative ways. The first one, which provides the best results over the public data-sets over which we experimented, sends in row-wise order and in interleaved manner the rows of $\mathcal{H}$ and $\mathcal{V}$, as explained in Subsection III-A. The second method encodes first a header which specifies all anchor points for the chain-codes, followed by encoding the chains formed by the active crack-edges, as explained in Subsection III-B.

A. Row-wise encoding the crack-edges by using context trees with two-dimensional contexts

In this method all the crack-edges (both active and inactive) stored in the images $\mathcal{H}$ and $\mathcal{V}$ are transmitted by scanning the images row-wise and in an interleaved way: one row from $\mathcal{H}$ is followed by the row with the same index from $\mathcal{V}$. Encoding is done using context arithmetic coding, [16], [31], where the template used for defining the encoding context of a vertical crack-edge, $V_{i,j}$, is formed from both horizontal and vertical crack-edges transmitted up to the current vertical crack-edge. The template is shown in Figure 2. One can select from the template a coding context containing any total number $n \leq 17$ of crack-edges, $T^v(V_{i,j}) = [T^v_1 \dots T^v_{n_v}] \in \{0,1\}^{n_v}$, in the order specified by the template. For example, in the middle top row of Figure 5 the encoding context is $T^v(V_{i,j}) = [T^v_1 \dots T^v_5] = [1\ 1\ 0\ 0\ 1]$.

Similarly, the template $T^h(H_{i,j})$ for the horizontal crack-edge, $H_{i,j}$, is shown in Figure 3. Each of the crack-edges marked by $1, 2, 3$ in Figure 3 shares one vertex with the current horizontal crack-edge (marked by "?"), and hence they are the most relevant. Next in order of relevance are the crack-edges $4-9$, sharing vertices with the crack-edges $1, 2$ and 3. The template ordering was optimized over two Middlebury files (Art and Reindeer, right view, one third resolution [32]) through a greedy iterative process. Initially the template was taken to contain the crack-edges marked $1-3$ and then, at each step of the greedy optimization, the template size was increased by one. The new crack-edge included in the template was the one leading to the best compression over the two images, when considering as candidates all the neighbor crack-edges of the crack-edges already existing in the template, obeying the causality constraint for the overall template. The compression performance was taken to be the one after optimally pruning the trees built with the current template (the pruning technique is presented in the following subsection). The process was stopped when the size of the template became 17, mostly for complexity reasons, but also because further enlarging the template did not improve significantly the compression performance.

The crack-edge indices shown in Figures 2 and 3 are recording the order in which the crack-edges were included in the template by the greedy algorithm. The template and the indexing of the crack-edge in the template, thus optimized for two images, were then fixed and used as such in all experiments of this paper. Interestingly, trying to change the order for the crack-edges in the template for some of the depth images did not result in significant improvements of the compression results, and hence the ordering of the crack-edges in the vertical and horizontal templates shown in Figures 2 and 3 seem to reflect well the natural order of importance for crack-edge neighboring in depth image contours.

We have used the context trees in the configuration requiring that the coding contexts are leaves in an overall binary context tree. In the fast variant, the context tree $\mathcal{T}^B_{n_T}$ for the horizontal crack-edges is a balanced tree having 2^{n_T} leaves, all at tree-depth n_T, and similarly the context tree for the vertical crack-edges is a balanced binary tree of same depth, in which case the data structure for storing the contexts and their counts is simply a table indexed by the binary contexts. If the common tree-depth n_T is too large, some of the contexts will not be seen at all, or the number of occurrences will be small, so that their statistical relevance will also be modest. The value of n_T optimizing the overall compression for a balanced tree was found experimentally to vary between 10 and 15. We choose as a default value in the CERV-Fast variant the value $n_T = 15$.

In the case of high-compression variant of the CERV method, the context trees are optimized by pruning the contexts which do not perform better than their parents, as described in the next subsection.

1) Optimal pruning the context tree: Context tree coding is a mature field, containing a rich literature proposing various adaptive and semi-adaptive algorithms and their applications, see e.g., [16], [31], [33], [34]. In the HiCo variant we use a semi-adaptive version requiring two-passes through the image. Further optimization of the extent of the context template and of the order of its pixels may add some improvements in compression efficiency, but will also slow down the execution of the program. In the high compression variant each tree is initialized as the balanced tree $\mathcal{T}^B_{n_T}$ and then pruned to an optimal tree at the end of the first pass through the image.

TABLE I
STATISTICS ABOUT THE ACTIVE CRACK-EDGES AND ABOUT THE PATCHES ENCLOSED BY THE ACTIVE CRACK-EDGES.

Image	Format	$n_r \times n_c$	Number of CE set to 1		Number of patches						Histogram of n_{neigh} (number of neighboring patches) using bins $[2^j, 2^{j+1})$					
					Total	Patches of size $s \in [10^i, 10^{i+1})$					Intervals for n_{neigh}					
						Intervals for $s = \lvert P \rvert$										
			Vert.	Hor.		1	$[2, 9]$	$[10, 99]$	$[100, 999]$	$[1000, \infty)$	1	$[2, 3]$	$[4, 7]$	$[8, 15]$	$[16, 31]$	$[32, \infty)$
Art	1/2	385725	47543	61841	5893	2062	2352	987	418	74	703	1999	2493	487	162	49
Art	1/3	171310	29054	32850	4306	1706	1562	744	275	19	386	1416	1973	419	91	21
Dolls	1/2	385725	63117	78335	9581	3988	3287	1594	641	71	670	4465	3273	880	254	39
Dolls	1/3	171310	38259	46596	8588	3821	3069	1325	365	8	388	3379	3808	807	183	23
Plastic	1/2	352425	47249	21763	1829	419	804	374	139	93	789	444	445	117	18	16
Plastic	1/3	156510	29589	11746	1459	452	572	273	109	53	519	315	511	83	20	11

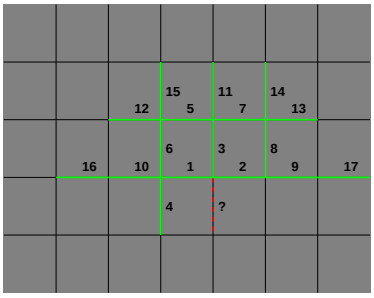


Fig. 2. The template of contexts for vertical crack-edges. The vertical crack-edge marked in red is the current one, to be predicted from the seventeen vertical and horizontal crack-edges marked in green. The index in the template given to each crack-edge is marked near it, the most significant being crack-edge one.

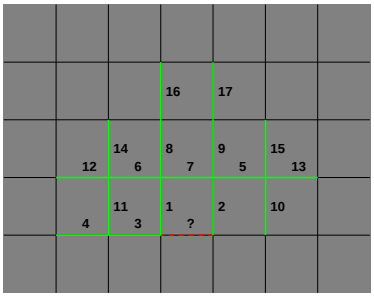


Fig. 3. The template of contexts for horizontal crack-edges. The horizontal crack-edge marked in red is the current one, to be predicted from the seventeen vertical and horizontal crack-edges marked in green.

The bitstreams describing the structure of the obtained vertical and horizontal optimal trees are sent as a side information to the decoder, so that both encoder and decoder use the same trees, in an adaptive manner. After optimization, during the encoding process, the counts of the symbols 0 and 1 are initialized at each leaf (but not anymore at the interior nodes) and they are updated at each visit of a leaf, leading to adaptive coding distributions. The pruning process used in this paper is presented for completeness in detail in an Appendix in the file with additional information and at the website created for the paper at http://www.cs.tut.fi/~tabus/CERV/Appendix.pdf.

The bitstream for transmitting the structure of an optimal tree encodes in each bit the decision $B(i)$ to split or not the node i, starting from the root and concatenating $B(\emptyset)B(0)B(1)\ldots$, by scanning at each tree-depth level the nodes which resulted from the splits of the previously encoded tree-depth level and including only for them the information about being split or not. The total number of nodes in the tree having n_{leaves} leaves is $2n_{leaves} - 1$, but the length of the bitstream for encoding the tree structure is possibly smaller, since for the leaves located at the maximal possible tree-depth n_T there is no need to transmit decisions to split. The binary bitstreams are encoded using arithmetic coding, with probabilities assigned by an adaptive first-order Markov model.

2) Encoding the crack-edges by using context trees: The vertical edges and the horizontal edges are encoded sequentially, row by row starting from the first row of vertical edges $V_{1,2}, \ldots, V_{1,n_c}$, and continuing in an interleaved manner of sending each row of horizontal edges $H_{i,1}, \ldots, H_{i,n_c}$ followed by a row of vertical edges, $V_{i,2}, \ldots, V_{i,n_c}$, with the last row encoded being a row of vertical edges.

The values of the vertical and horizontal crack-edges are transmitted using arithmetic coding, with their coding distribution given by the counts collected in the two context trees.

The two context trees for encoding the $\mathcal{V}$ and $\mathcal{H}$ images meet special situations at the boundary of the image, where some crack-edges required in the template are not available. In that case all the needed values which are not available are considered to be 0, which simplifies the encoding and decoding routines by avoiding treating separately the crack-edges close to the border. Only the first row of vertical edges $V_{1,2}, \ldots, V_{1,n_c}$ is encoded directly as a separate context, without using the optimal vertical context tree.

In Figure 4 (a) and (b) are shown a depth map image and a detail of the image, where the active crack-edges are overlaid, drawn with green lines. In the lower row, left panel of Figure 5 it is shown the zoomed area Z_1, where green lines are representing the active crack-edges, while blue and red lines are showing the crack-edges having as context the binary all-zero vector $T^h(H_{ij})$, which as an integer reads $T^h(H_{ij}) = 0$, and for short we call it context 0. Context 0 for the particular case of tree-depth 17 is shown in the upper row, left panel of Figure 5 (for some images the all-zero vector $T^h(H_{ij})$, called here context 0, will have length smaller than 17 after pruning).

In regions of the image similar to Z_1, with high density of active crack-edges, the context 0 is rarely found, while in the large patches the context 0 will appear very often, being the most frequent non-deterministic context in the overall image. The next two-most frequent contexts are illustrated in a similar manner on the middle and right panels of Figure 5. The next most frequent 15 contexts are shown in the files containing additional information at http://www.cs.tut.fi/~tabus/CERV/.

3) Deterministic crack-edges: In general the two images of crack-edges, $\mathcal{H}$ and $\mathcal{V}$ can be seen to form a particularly constrained pair of bi-level images. The most specific property is the fact that the crack-edges are forming chains that can terminate only when meeting other chains (including themselves) or when reaching the boundaries of the image, as can be seen from Figure 1. This property can be utilized for deriving deterministic connections between the crack-edges from the template and the current crack-edge, and was used for the first time in [1], for its slightly differently defined vertical and horizontal contexts (where some contexts of the horizontal crack-edges resulted to be deterministic).

The context template of the current vertical crack-edge $V_{i,j}$ has a remarkable property, allowing to determine a unique value of the current vertical crack-edge for some particular values of the crack-edges in the template. The template contains all three crack-edges, $T_1^v = H_{i,j-1}, T_2^v = H_{i,j}$, and $T_3^v = V_{i-1,j}$, marked 1,2 and 3 in Figure 2, having a common vertex to the current vertical crack-edge $V_{i,j}$. The following configurations are deterministic:

1) If none of the crack-edges T_1^v, T_2^v, and T_3^v is active, the current vertical crack-edge $V_{i,j}$ is not active. To see this, $H_{i,j-1} = 0$ implies $D_{i,j-1} = D_{i-1,j-1}$; $V_{i-1,j} = 0$ implies $D_{i-1,j-1} = D_{i-1,j}$; and $H_{i,j} = 0$ implies $D_{i-1,j} = D_{i,j}$. Hence $D_{i,j-1} = D_{i-1,j-1} = D_{i-1,j} = D_{i,j}$ which makes $V_{i,j} = 0$.
2) If a single one of the crack-edges $H_{i,j-1}, H_{i,j}$, and $V_{i-1,j}$ is active, the current crack-edge $V_{i,j}$ is active. There are three configurations in this class, e.g., $H_{i,j-1} = 0, H_{i,j} = 0$, and $V_{i-1,j} = 1$, which imply that $D_{i,j-1} = D_{i-1,j-1} \neq D_{i-1,j} = D_{i,j}$ and thus $V_{i,j} = 1$. The proof is similar for the other two configurations.

In cases where two or three of the crack-edges $H_{i,j-1}, H_{i,j}$, and $V_{i-1,j}$ are active, there is no deterministic conclusion about the value of the crack-edge $V_{i,j}$. The deterministic situations occur rather often, as seen in Table II, their proportion may vary from one third to 47% of all crack-edges (corresponding to the large majority of vertical crack-edges, which are about half of all crack-edges). The zoomed region Z_1 is shown in Figure 4 (c), where the active crack-edges are shown as green lines. With thicker lines are shown in blue or red those vertical crack-edges appearing in deterministic contexts: the marking is blue, if their value was 0, or is red, if their value was 1. In the whole picture from Figure 4 (a) there are 136860 deterministic contexts, out of 170940 contexts for vertical crack-edges (nearly 80%).

4) Accounting for the non-stationarity of context distributions: The conditional distribution collected at each context is updated on the fly, reflecting the data observed while being in that context. However, the depth images are non-stationary, e.g., for pictures taken inside a room the walls, floor, and foreground are resulting in different types of contours of the patches. It was found useful to introduce a form of forgetting in the updating process of the counts, in the form of halving the counts of zeros and ones at a given context, each time when the total number of zeros and ones exceeds a certain threshold, as is done in many cases for re-scaling the counts used in arithmetic coding [16] [35]. The optimal threshold, at which halving is done, varies for different images between a few tens and about one thousand. We have used in the experimental section a fixed threshold, by default equal to 250, at which to perform the halving of the counts for all contexts in all images, except the counts for context 0, which were left unchanged.

B. Encoding by chain-codes

When the active crack-edge density is low, it may become more efficient to encode the crack-edges along the chains of active crack-edges and to identify the next active crack-edge by a chain-code.

We consider an algorithm based on chain-codes, similar to the ones used in [7] and [23], which sends first the information about all anchors needed for defining chains, assuming that each point in the image can be an anchor, and also marks the anchors which are multiple start points. The encoder collects the 3OT codes for all resulted chains and concatenates them in a long string of symbols $0, 1$, and 2. The context tree optimal for the overall string is built and transmitted as side information and then the chain-codes are transmitted using the encoding distribution collected adaptively at the optimal contexts [23]. Two alternatives were tested, the one using 3OT chain-codes and the one using AF4 chain-codes, but very small differences were observed among the two alternatives (results not shown). The results for the current datasets are however overwhelmingly in favor of the coding using 2D contexts (e.g., in Table II the third and last columns show a very significant difference, in favor of 2D contexts) which is used in all the rest of experiments in this paper.

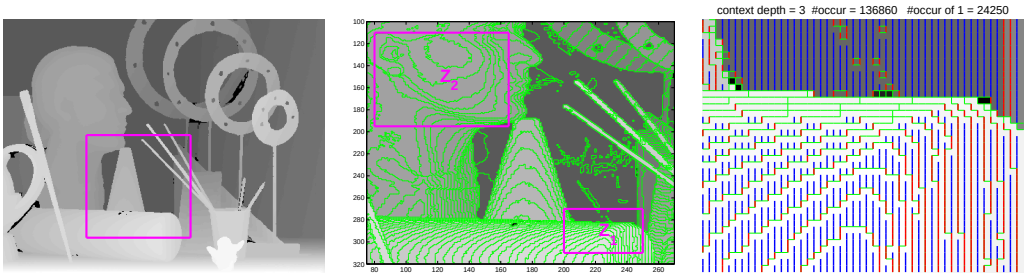
IV. Encoding the depth value of each patch

A. Depth dependencies across neighboring regions

In encoding the depth value of each patch the possible closeness between its value and the values of the neighbor patches is used for constructing suitable predictions, in the form of a list of most likely values taken by the depth. The fact that the depth values over neighboring patches ought to be distinct is used for excluding from the likely list the depth values of all neighboring patches.

The neighboring relationship between patches, which we define by specifying the set of neighbor patches $\mathcal{N}(P_\ell)$ for each patch P_ℓ, will be specific to every image, and both encoder and decoder will know it, since in the first stage the contour of the patches is transmitted. The set of all patches $\mathcal{P}$ and its cardinality $n_P = |\mathcal{P}|$ are also available to the decoder.

The values of the patch-depths are transmitted in a sequential order to the decoder, hence only the patch-depths already encoded can be used as conditioning variables. The encoding



(a) (b) (c)

Fig. 4. (a) The disparity image *Art* from view 5, in third resolution. (b) Overlaying the active crack-edges over the marked rectangle from image Art. The marked regions by rectangles are used for illustrations in Figures 5 and 9. (c) The vertical-crack edges from the zoomed rectangle Z_1 which are deterministically specified by their contexts are marked as follows: the ones which are active crack-edges are marked in red, the ones which are inactive crack-edges are marked in blue. The rest of active crack-edges (in non-deterministic contexts) are marked in green. Hence, in red and blue are shown all vertical crack-edges not needing to be encoded. The four contexts for deterministic vertical crack-edges, are all defined by the first three crack-edges in the template of Figure 2, by the condition $H_{i-1,j} + H_{i,j} + V_{i,j-1} \leq 1$.

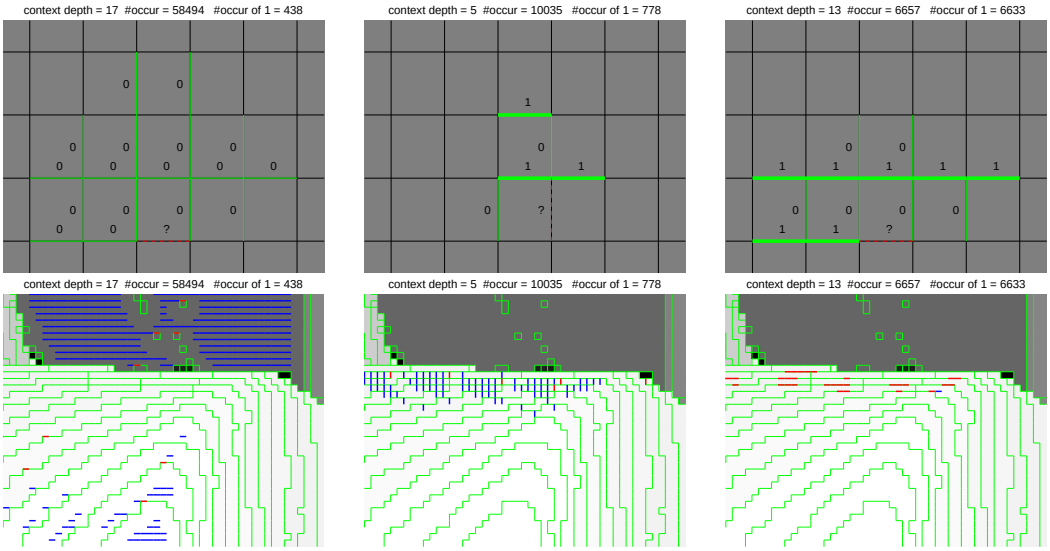


Fig. 5. The three most frequently occurring non-deterministic contexts in the pruned vertical and horizontal context trees obtained for the disparity image Art from view 5, in third resolution. The images in the first row show the pruned contexts, where red dotted line is for the crack-edge to be predicted, thin green line is for the template crack-edges which ought to be zero, and thick green line is for the template crack-edges which ought to be one. (Left) The context 0, the most frequent for horizontal crack-edges, maintaining all 17 crack-edges in the pruned context. (Middle) The most frequent context for vertical crack-edges, which was optimally pruned to context-depth 5. (Right) The next most frequent context for horizontal crack-edges, which was optimally pruned to context-depth 13. The second row of plots is a zoom in the rectangle marked Z_1 in Figure 4(b), and is marking the crack edges which were encoded using the contexts shown in the first row. The crack-edges which were encoded in the context marked in the above row are marked as follows: with blue are marked the inactive crack-edge and with red the active crack-edges, while with green are marked all other crack-edges active in the zoomed image. The given numbers of occurrences refer to the whole image Art, not only to the region Z_1.

order of the patch-depths is denoted here for notational simplicity, as $1, 2, \ldots, n_P$. The enumeration used here consists in scanning the initial image line by line, and transmitting the depth values of the patches in the order in which they are first met during the scanning.

We considered also a second enumeration, by sorting the patches in the decreasing order of their number of neighboring patches, so that first are specified the depth-values of those patches having many neighbors, adding at each instant the depth information that is relevant for as many unknown yet patch-depth values as possible. However, the second enumeration performed slightly worse than the first one, hence in the rest of the paper we utilize only the first enumeration.

At the moment t the depth value $d^* = D_{P_t}$ of the patch P_t is encoded, making use of the depth values of the k_t neighboring patches forming the set $\mathcal{P}_t = \{P_{t(1)}, \ldots, P_{t(k_t)}\} =$

TABLE II
COMPRESSION OF CRACK-EDGES WHEN ENCODING BY 2D CONTEXTS (COLUMNS 3-9) AND WHEN ENCODING BY CHAIN-CODES (COLUMNS 10-11).

Image	Format	Encoding CE by 2D contexts							Encoding CE by 3OT chains	
		Cost CE [bpp]	Total CE	active CE [%]	deterministic [%]	Context 0 [%]	Entropy of Context 0	bit per active CE [bits]	bit per active CE [bits]	Cost CE [bpp]
Art	1/2	0.330	770200	14.2	41.8	22.9	0.068	1.16	1.46	0.413
Art	1/3	0.432	341787	18.1	40.0	17.3	0.067	1.20	1.58	0.570
Dolls	1/2	0.479	770200	18.4	39.5	16.8	0.127	1.31	1.76	0.645
Dolls	1/3	0.638	341787	24.8	35.9	12.8	0.167	1.29	1.91	0.945
Plastic	1/2	0.220	703660	9.8	46.8	26.1	0.041	1.13	1.37	0.268
Plastic	1/3	0.284	312227	13.2	46.0	19.9	0.055	1.07	1.38	0.364

$\mathcal{N}(P_t) \cap \{P_1, \ldots, P_{t-1}\}$, for which the depth values were already encoded. The vector of these depth values is $\underline{D}^*(t) = [D_{P_{t(1)}}, \ldots, D_{P_{t(k_t)}}]$ and its elements which are unique form the set of depth values $\mathcal{D}_t^*$.

Building directly conditioning contexts using for conditioning the random vector $\underline{D}^*(t)$ is not effective when the number k_t of known neighboring patches is large, because the depth value alphabet, $\mathcal{A}$, is large and the resulting contexts will be diluted. A more structured process of conditioning is needed for $k_t > 2$, where first the values from $\mathcal{D}_t^*$ are clustered, and the centers of the chosen clusters are used for conditional encoding, by using two lists in the encoding process. One list, $\mathcal{L}_t$, contains likely values, close to the cluster centers, ordered in such a way that the value d^* is very likely to be situated in the top positions of the list. Whenever d^* is present in the list, its rank is encoded, instead of the value d^* itself. A second list, $\mathcal{W}_t$, is a default list of values and it is used when the value d^* is not found among the elements of the list $\mathcal{L}_t$.

B. Clustering $\mathcal{D}_t^*$ and constructing the likely list of values $\mathcal{L}_t$

The clustering algorithm performs grouping of the values $\mathcal{D}_t^*$ around centers selected sequentially. The goal of the clustering algorithm is to distinguish between two often met cases: one case is when all neighbor values in $\mathcal{D}_t^*$ are very close together, and most likely the value d^* will be close to all of them as well; this happens when the neighboring patches belong to the same object in the image. The second situation is when the neighboring patches belong to two different objects, situated at very different depths in the image, and the values in $\mathcal{D}_t^*$ form two clusters. Other, more complex, situations are certainly possible, but their occurrence is quite rare and we prefer a fast clustering algorithm, which selects sequentially centers of clusters using a threshold Δ for deciding which values to include to the already existing centers. A fixed value of $\Delta = 5$ is used throughout the paper.

The values in $\mathcal{D}_t^*$ are arranged in the order they are met along the contour of the current patch. The clustering algorithm takes the first value from $\mathcal{D}_t^*$ as center of the first cluster. All other values from $\mathcal{D}_t^*$ that are within a distance of Δ from the center of the cluster are marked and allocated to the cluster, and the center of the cluster is recomputed as the mean of all values in the cluster. The next value from $\mathcal{D}_t^*$ which is not marked yet is then taken as initial center of a new cluster, and this cluster is grown in a similar way, including the values of $\mathcal{D}_t^*$ not yet marked and situated closer than Δ from the cluster center. The process ends when all values from $\mathcal{D}_t^*$ are allocated to one of the created clusters. If the number of resulted clusters, n_Q, is larger than two, we set $n_Q = 2$ and only the two most populated clusters are kept, having centers denoted Q_1 and Q_2. If the distance between the centers of the two clusters is smaller than Δ, then n_Q is set to 1 and a single center is computed as the mean of the values in the two clusters. Each cluster center, Q_i, is rounded to its closest integer value.

If the number of clusters is $n_Q = 1$ the list of likely values is initialized as $\mathcal{L}_t = [Q_1, Q_1+1, Q_1-1, Q_1+2, Q_1-2, \ldots]$, while when the number of clusters is $n_Q = 2$, the list is initialized as $\mathcal{L}_t = [Q_1, Q_2, Q_1+1, Q_1-1, Q_2+1, Q_2-1, Q_1+2, Q_1-2, \ldots]$. Then the values in $\mathcal{D}_t^*$ are excluded from the list, after which the first $2\Delta+1$ elements in the list are kept to form the final list $\mathcal{L}_t$. It is expected that d^* will be located often in top ranks of the list $\mathcal{L}_t$, and therefore its rank in the list will be encoded, instead of its value. In order to separate the different situations regarding the number of values in $\mathcal{D}_t^*$ and the number of clusters, we found five context to be relevant for collecting statistics about the rank of d^* in the list $\mathcal{L}_t$, as follows:

$$i_C = \begin{cases} 1 & \text{if} \quad |\mathcal{D}_t^*| = 1 \\ 2 & \text{if} \quad |\mathcal{D}_t^*| = 2, n_Q = 1 \\ 3 & \text{if} \quad |\mathcal{D}_t^*| = 2, n_Q = 2 \\ 4 & \text{if} \quad |\mathcal{D}_t^*| > 2, n_Q = 1 \\ 5 & \text{if} \quad |\mathcal{D}_t^*| > 2, n_Q = 2 \end{cases} \tag{1}$$

In the context $i_C = 1$ the value of one single neighbor is known, in context $i_C = 2$ two neighbors are known and their distance is smaller than Δ, in context $i_C = 3$ two neighbors are known and are further apart than Δ, in context $i_C = 4$ the clustering process resulted in a single Δ-bounded cluster, and in context $i_C = 5$ the clustering resulted in two Δ-bounded clusters. If there is no neighbor yet known about a region, the list $\mathcal{W}_t$ will be used.

The list $\mathcal{L}_t$ can be constructed identically at the decoder. If the true value is contained in the list, then $d^* = \mathcal{L}_{tk}$, where k is its rank in the list. A binary switch S_t is transmitted first, telling if d^* belongs to the list. If $S_t = 1$ then the rank k will be encoded, using for driving the arithmetic coder the statistics of the ranks collected at the context i_C. If $S_t = 0$, d^* was not among the values in the list $\mathcal{L}_t$ (we deal with a patch having very different value than those in the clusters constructed from the neighboring patches), and then the value of d^* is sent using the statistics collected using the default list $\mathcal{W}_t$ (we label this distribution by the label $i_C = 6$).

Y0. Given the set $\mathcal{P}$ of global patches obtained from the crack-edges transmitted in the first stage.
Y1. Initialize counters $N^I_{j,i_C} = \delta_C$ for the contexts $i_C = 1, \ldots, 5$ and indices $j = 1, \ldots, 2\Delta + 1$ in the list of likely indices, where $\delta_C = \frac{1}{2\Delta+1}$. Initialize the switch counters $N^S_{j,i_C} = \frac{1}{2}$ for the switch $j \in \{0, 1\}$ in each of the contexts $i_C = 1, \ldots, 5$. Initialize the value counters $N^d_j = \frac{1}{2^B}$ for $j = 0, \ldots, 2^B - 1$.
Y2. Enumerate the patches from $\mathcal{P}$ as $P_1, \ldots, P_{n_P}$, in the order they are met along the row-wise scanning of the depth image.
Y3. For the current patch with index $t = 1, \ldots, n_P$ *(iterate until the depth values of all the patches were encoded)*
  Y3.1. The depth value to be encoded is $d^* = D_{P_t}$. The set of known neighbor patches is $\mathcal{P}_t = \mathcal{N}(P_t) \cap \{P_1, \ldots, P_{t-1}\}$ and the set of depth values over these patches is $\mathcal{D}^*_t = \{D_P | P \in \mathcal{P}_t\}$ *(set of distinct values, not a multi-set)*.
  Y3.2. Cluster the values in $\mathcal{D}^*_t$ and construct the list of likely values, $\mathcal{L}_t$ *(as in Subsection IV-B)*.
  Y3.3. If $d^* \in \mathcal{L}_t$, set the switch $S_t = 1$, otherwise $S_t = 0$. Encode S_t using $-\log_2 \frac{N^S_{S_t,i_C}}{\sum_i N^S_{i,i_C}}$ bits. Update the count $N^S_{S_t,i_C} \leftarrow N^S_{S_t,i_C} + 1$.
  Y3.4. If $S_t = 1$, encode the rank j^* of d^* in the list $\mathcal{L}_t$, (for which $\mathcal{L}_{t,j^*} = d^*$), using $-\log_2 \frac{N^I_{j^*,i_C}}{\sum_j N^I_{j,i_C}}$ bits. Update the count $N^I_{j^*,i_C} \leftarrow N^I_{j^*,i_C} + 1$.
  Y3.5. If $S_t = 0$, encode the value d^* (which is known to belong to the set $\mathcal{W}_t = \mathcal{A} \setminus \mathcal{L}_t \setminus \mathcal{D}^*_t$) using $-\log_2 \frac{N^d_{d^*}}{\sum_{j \in \mathcal{W}_t} N^d_j}$ bits. Update the count $N^d_{d^*} \leftarrow N^d_{d^*} + 1$.
  Y3.6. Move to the next patch in the list, P_{t+1}, and Goto Y3.1.

Fig. 6. The *Algorithm Y*, for encoding the depth value in each of the constant depth regions, using at the iteration t the set of already known depth values over neighboring patches, $\mathcal{D}^*_t$, the list of likely values, $\mathcal{L}_t$, and the default list, $\mathcal{W}_t$.

We illustrate the algorithm using the 4×5 image from Figure 1. When scanning row-wise the image, we find in order the following patches: P_1 having depth $D_{P_1} = 79$, P_2 having depth $D_{P_2} = 101$, P_3 having depth $D_{P_3} = 78$, P_4 having depth $D_{P_4} = 100$, and P_5 having depth $D_{P_5} = 102$. We consider here for simplicity of illustration $\Delta = 2$, although in all the experiments we have used only the value $\Delta = 5$. The Table III shows the relevant variables when processing the five patches. The coding distribution $\mathbb{P}(d^*|i_C)$, with $i_C = 6$ refers to coding using the default list $\mathcal{W}_t$, while the coding distribution $\mathbb{P}(k|i_C)$ refers to coding using the list $\mathcal{L}_t$, for contexts $i_C < 6$.

V. The variants of the CERV coding scheme

The generic CERV algorithm is shown in Figure 7. In the variant CERV-HiCo all steps of the algorithm are performed, providing the maximum compression of the scheme. In the variant CERV-2 the Step A4 of constructing global patches is omitted, but still two passes are needed for the overall encoding. In the variant CERV-3 the stage of optimizing the coding trees, consisting of Steps A1 and A2, is omitted, while the marking of global patches in Step A4 is performed. This variant eliminates the need of the first pass of collecting the counts in the coding trees and encodes with default balanced trees, but still needs one pass through the image for transmitting the crack-edges, only then the global patches

A0. Initialize the coding trees for the vertical and horizontal crack-edge as two balanced trees $\mathcal{T}^B_{n_T}$ having tree depth $n_T = 17$ *(default value)*.
A1. Do one pass through the images of crack edges and collect the counts in all the nodes of the context trees.
A2. Prune each of the two coding trees using dynamic programming and transmit to the decoder the structure of the optimal trees, as described in Subsection III-A1.
A3. Do one pass through the image for encoding crack-edges, using the two coding trees that are available at the decoder, as described in Subsection III-A2.
A4. Mark the global patches in the image, using the crack-edge information encoded in Step A3.
A5. Do one pass through the image and execute the Algorithm Y or Algorithm F2.3 for encoding the depth values of the patches.

Fig. 7. Generic *Algorithm A*, describing the overall structure of CERV coding algorithms. The algorithm CERV-HiCo contains all the steps and uses Algorithm Y in Step A5; the algorithm CERV-2 does not contain Step A4 and uses Algorithm F.2.3 in Step A5; while the algorithm CERV-3 does not contain the steps A1 and A2 and uses Algorithm Y in Step A5.

are found and finally in a second pass through the image the depth-values are transmitted by Algorithm Y. The variants CERV-2 and CERV-3 are introduced and exemplified solely for illustrating the gains of the main three parts of the CERV algorithm.

The gains of using the steps A1 and A2 (optimization of context trees for crack-edges) and A4 (determining the global patches) of the generic algorithm are almost equal, as seen from the comparative results in Figure 11.

The fourth variant is the Algorithm CERV-Fast, which omits the optimization of the context trees for encoding crack-edges (Steps A1 and A2) and also omits the construction of the global patches (Step A4). In this form, it can be executed in a single pass through the image, providing much better speed and smaller memory requirements than the CERV-HiCo algorithm.

We present a pseudocode of the algorithm CERV-Fast in Figure 8. Since the algorithm operates with the constant segments on the current line instead of global patches, it can infer that two non-connected constant segments belong to the same region by only utilizing the information acquired in the preceding lines of the image, while sometimes only the subsequent lines can clarify if two segments belong to the same patch or not. Thus, the fast variant will have to encode depth values for all *new* constant segments from the current line, that are not connected to a a known segment on the previous line, even though some of these new constant segments may belong to an already met patch. At the moment t the depth value $d^* = D_{P_t}$ is encoded, making use of the depth values of the k_t neighboring patches forming the set $\mathcal{P}_t = \{P_{t(1)}, \ldots, P_{t(k_t)}\}$ which is now only a subset of $\mathcal{N}(P_t) \cap \{P_1, \ldots, P_{t-1}\}$, differently than in the Algorithm Y, where the the two sets were equal.

In Figure 9 are shown cases where the fast algorithm treats two nonconsecutive constant-segments on the current row as belonging to two different patches, although the two segments belong to the same patch, as revealed by the image following the current row. The one-pass fast algorithm at the moment of encoding the current row will encode a depth value

TABLE III
ILLUSTRATION OF OPERATIONS IN ALGORITHM Y USING THE 4×5 IMAGE FROM FIGURE 1.

t	P_t	D_{P_t}	$\mathcal{N}(P_t)$	$\mathcal{D}_t^*$	$Q_1, \ldots, Q_{n_Q}$	i_C	$\mathcal{L}_t$	rank k	S_t	$\mathbb{P}_{\text{Encode}}$
1	P_1	79	$\{P_2, P_3, P_4\}$	$\emptyset$	$\emptyset$	6	$\emptyset$	–	–	$\mathbb{P}(d^*\|i_C) = \frac{1}{255}$
2	P_2	101	$\{P_1, P_3, P_4, P_5\}$	$\{79\}$	$\{79\}$	1	$[80, 78, 81, 77, 82]$	–	0	$\mathbb{P}(d^*\|i_C) = \frac{1}{250}$
3	P_3	78	$\{P_1, P_2, P_4\}$	$\{79, 101\}$	$\{79\}, \{101\}$	3	$[80, 78, 102, 100, 81]$	2	1	$\mathbb{P}(k\|i_C) = \frac{0.2}{5\times 0.2}$
4	P_4	100	$\{P_1, P_2, P_3\}$	$\{78, 79, 101\}$	$\{79\}, \{101\}$	5	$[80, 77, 102, 100, 81]$	4	1	$\mathbb{P}(k\|i_C) = \frac{0.2}{5\times 0.2}$
5	P_5	102	$\{P_2\}$	$\{101\}$	$\{101\}$	1	$[102, 100, 103, 99, 104]$	1	1	$\mathbb{P}(k\|i_C) = \frac{0.2}{5\times 0.2}$

TABLE IV
COMPRESSION OF DEPTH VALUE FOR EACH PATCH, WITH ENUMERATION OF PATCHES BY THE ORDER OF REACHING THEM IN ROW-WISE SCANNING, WHEN ENCODING WITH THE LIKELY LIST $\mathcal{L}_t$ AT A CONTEXT $i_C \in \{1, \ldots, 5\}$, OR ENCODING WITH THE DEFAULT LIST, $\mathcal{W}_t$ (CONSIDERED AS CONTEXT $i_C = 6$).

Image	CERV-Fast	CERV-HiCo														
		Probability of the contexts $\hat{p}(i_C) = \frac{m(i_C)}{n_P}$						Codelength per symbol in each context						Codelength per pixel for encoding:		
	$\frac{n_{LP}}{n_P}$	i_C						i_C								
		1	2	3	4	5	6	1	2	3	4	5	6	Patches	CEs	Total
Art $\frac{1}{2}$	1.14	0.30	0.34	0.21	0.02	0.04	0.10	1.81	0.82	2.53	3.37	0.98	6.60	0.035	0.330	0.365
Art $\frac{1}{3}$	1.09	0.30	0.30	0.22	0.02	0.07	0.09	1.83	1.63	2.57	3.71	1.33	7.69	0.064	0.432	0.496
Dolls $\frac{1}{2}$	1.21	0.36	0.28	0.18	0.02	0.03	0.12	2.03	1.87	2.80	3.52	2.53	6.45	0.066	0.479	0.545
Dolls $\frac{1}{3}$	1.11	0.32	0.35	0.17	0.02	0.04	0.10	2.07	1.77	2.79	3.50	2.19	7.41	0.132	0.638	0.770
Plastic $\frac{1}{2}$	1.34	0.71	0.10	0.09	0.02	0.00	0.08	1.47	1.81	2.95	3.86	4.83	6.86	0.011	0.220	0.231
Plastic $\frac{1}{3}$	1.22	0.60	0.21	0.09	0.02	0.01	0.06	1.40	1.25	2.94	2.65	3.27	8.86	0.019	0.284	0.303

for the constant segment, although the algorithm Y will not encode anything for this segment. This inefficiency of the fast algorithm is quantified in Table IV, by the second column, showing the number of constant segments for which the depth was encoded in the fast algorithm, divided by the number of patches n_P, which is the number of depth values encoded in the HiCo algorithm. The larger this ratio, the larger the relative difference in compression between the fast algorithm and the HiCo algorithm.

VI. DISCUSSION OF THE ALGORITHMIC SIMILARITIES AND DIFFERENCES BETWEEN CERV AND PWC

The piecewise-constant regions in PWC, called here patches, are separated by edges forming the edge map. In both CERV and PWC the initial representation of the depth image is formed of the two sets of variables: the binary crack-edge images $\mathcal{H}$ and $\mathcal{V}$, which are referred to as edge map in PWC, and the depth values inside the patches.

The coding of edge map in both PWC and CERV is done using 2D binary contexts. In PWC the contexts have a fixed size, taken as in [25], including the 8 causal neighbor edges for coding a vertical edge (those marked 1,3,4,5,6,7,8, and 13 in Figure 2) and 9 causal neighbor edges for coding a horizontal edge (i.e. when encoding the horizontal edge marked 2 in Figure 2 the context is formed of the edges marked 1,?,3,4,5,6,7,8, and 13). In CERV the whole line of horizontal edges is encoded first, and then the line of vertical edges is encoded, resulting in slightly different causal neighborhoods, but this difference is minor. The main advantage in CERV, concerning encoding edges, is the larger template used (it includes 17 neighbor crack-edges) and the use of variable length contexts, by employing a semi-adaptive context tree, designed and transmitted as side information after a first pass through the image.

F0. Initialize the memory of the algorithm: only buffers for three rows of vertical and horizontal crack-edges are needed. Initialize counters $N^V_{\underline{t},i} = 1/2$ and $N^H_{\underline{t},i} = 1/2$ for the binary symbols $i = 0, 1$ observed in the contexts $\underline{t}$ from the balanced trees $\mathcal{T}^V_{n_T}$ and $\mathcal{T}^H_{n_T}$. Initialize counters as in Step Y1 of the Algorithm Y in Figure 6.
F1. Transmit the vertical crack-edges $V_{1,2}, \ldots, V_{1,n_c}$ using a separate context.
F2. For i = 2 to n_r
  F2.1. Transmit the horizontal crack-edges $H_{i,1}, \ldots, H_{i,n_c}$ using the contexts in the balanced tree $\mathcal{T}^H_{n_T}$ having tree depth $n_T = 15$ (default value) and update the counters in each context, obtained with the template from Figure 3.
  F2.2. Transmit the vertical crack-edges $V_{i,2}, \ldots, V_{i,n_c}$ using the contexts in the balanced tree $\mathcal{T}^V_{n_T}$ having tree depth $n_T = 15$ (default value) and update the counters in each context, obtained with the template from Figure 2. If context is deterministic there is nothing to code.
  F2.3. Find the segments of constant pixels in each line and treat them in the order of discovery.
    F2.3.1. For a generic segment of constant pixels, $D(i_r, i_{start}), D(i_r, i_{start}+1), \ldots, D(i_r, i_{stop})$ at line i_r.
    The depth value to be encoded is $d^* = D(i_r, i_{start})$.
    If any of the pixels on top of the segment has the same depth d^*, then the current segment does not need to be encoded, so move to the next segment and Goto F.2.3.1.
    Else, encode the depth of the segment. The set of depths for known neighbors contains the left pixel (we set $D(i_r, i_{start}-1) = \emptyset$ if there is no left pixel) and all above pixels, i.e., $\mathcal{D}_t^* = \{D(i_r, i_{start}-1), D(i_r-1, i_{start}), D(i_r-1, i_{start}+1), \ldots, D(i_r-1, i_{stop})\}$.
    F2.3.2. Execute the Steps Y3.2 to Y3.5.

Fig. 8. The overall *CERV-Fast coding algorithm*

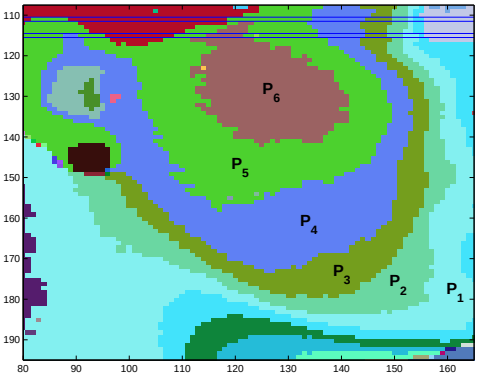


Fig. 9. The patches from the zoomed area Z_2 of image Art are represented in random colors. When processing the row 115 (marked with horizontal blue lines) the one-pass algorithm CERV-Fast will consider the gray segment (115,88)-(115,90) as a new patch and will encode its depth value, although this segment belongs together with the segment (115,133)-(115,144) to the patch P_4, for which the depth value is already known. Differently, in the two-passes algorithm CERV-HiCo, all pixels are marked with their patch index, so the depth values are encoded only once for each patch. Similarly, for the segment (111,82)-(111,88) in CERV-Fast there will be a depth value encoded, while CERV-HiCo will not need to encode anything, recognizing that the segment belongs to the patch P_5, already known as depth value at the time of scanning the row 111.

Both methods are taking advantage of the existence of deterministic contexts, which exist due to the constraints existing between the four crack-edges that are sharing a given vertex.

The quantitative gains obtained by CERV with respect to PWC due to the different type of contexts are similar to the gains of CERV-HiCo compared to the version CERV-3. In the variant CERV-3 the stage of optimizing the context tree is missing, and hence CERV-3 uses a fixed context, of size 15. The gains obtained due to the optimization of the tree are in average of about 1%, but occasionally they raise to 3% (see the black curve compared to the red line reference in Figure 11). Difference between CERV-HiCo and PWC will be higher, since PWC uses only a 8-9 long context, while CERV-3 uses a 15 long context.

One version of PWC uses a stage of run-length coding for repeated contexts which had a beneficial effect in the overall compression. In CERV such a stage is not used, especially because in CERV the contexts are much wider than in PWC. The only significantly repeating context in CERV is the "zero" context for horizontal crack-edges, $T^h(H_{i,j}) = 0$, which has an extremely skewed distribution, leading to very efficient encoding.

The second main task in both PWC and CERV, that of encoding the depth values over the piecewise-constant regions, is accomplished by starting from different goals and redundancy reduction techniques. In PWC the encoding of current region depth is obtained by mixing three hard decisions: diagonal connectivity, color guessing, and guess failure. The first decision, diagonal connectivity becomes active when two regions which are neighbors in 8 connectivity have the same depth. This may happen frequently in palette images containing diagonal lines of width one pixel, or when text is overlapped to natural content. CERV does not use anything similar to the diagonal connectivity primitive, depth images containing rarely thin lines.

The second primitive action in PWC, color guessing, consists in encoding the current unknown color conditionally on the color of a neighbor pixel, or on the colors of three neighbor pixels, where the conditional distribution is defined by an intricate dynamical structure which collects information about previous successful guesses. The list used in PWC implements a dynamical structure, containing memory cells, where the most recently used context gets the front position if the guess in that context is successful. Finally, when even the second primitive could not determine correctly the color value, a third primitive is used in PWC, where the the color is picked from the colors which are not included in the model, using zero order statistics of previous usage, or if the overall number of color values in the image is larger than 16, predictive coding is used. The predictive coding uses the predictor from JPEG-LS [36] and encodes the residuals as in ALCM [37].

By contrast CERV uses for color coding of the current region only the instantaneous information about the depth values in the neighboring regions. Since one region may have many neighboring regions, the conditioning values used by CERV are not only one or three, as used in PWC, but sometimes tens of regions may be neighbors of a given region and exploiting this network of regions results in skewed distributions. The clustering process in CERV described in Subsection IV-B may lead to several cluster centers, which can be seen as several different candidate predictions, the errors with respect to these predictions being encoded, after the important exclusion process is enforced. In order to keep track of the values with small errors, and at the same time to enforce exclusion, CERV uses the list construction from Subsection IV-B, which is a memoryless process applied to the network of region's neighbors at the moment of encoding one region's depth value. The exclusion process, enforcing that the current region should have a distinct value than the neighboring regions, combined with the predictive part seems to be very suitable for depth images.

The previous parallel of the methods for color coding in PWC and CERV reveals that the two coding techniques are starting from the same representation of the image into elements, but the techniques used for coding are different. We have tested the ability of the CERV techniques to cope with the redundancy present in general palette images. We have run CERV over the PWC corpus made up of several typical palette images and we found that the results of PWC were superior for almost all files (results shown in the additional information file), showing once more that the specific color coding used by PWC and by CERV are very distinct. Reversely, all results comparing CERV and PWC over the depth images in this paper are showing consistently better results of CERV for depth images. Consequently one can conclude that each method is well suited for the type of images for which each was intended in the first place.

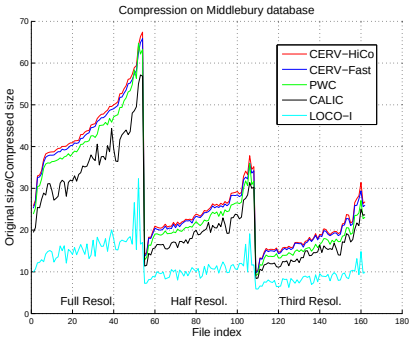


Fig. 10. Comparison of the compression ratio (compressed size over original size) for CERV algorithms, PWC, CALIC, and LOCO-I, over the Middlebury dataset (containing 54 images in three different resolutions).

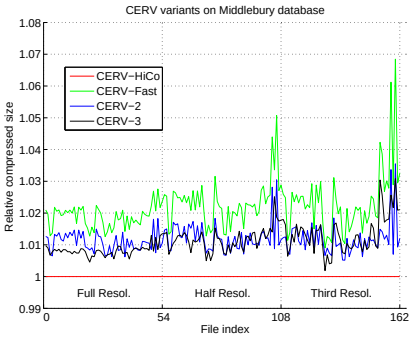


Fig. 11. Ratio of the compressed sizes achieved by each variant of the CERV methods, with respect to the compressed size obtained by the high-compression method, CERV-HiCo, which is taken as a reference.

VII. Experimental results

A. *Lossless compression of depth map images*

We consider here publicly available disparity images (which are in a one-to-one mapping with depth images). The Middlebury database [32] contains in total 162 disparity maps, for which we plot the results maintaining the grouping of images having the same resolution (54 at full resolution, 54 at half resolution, and 54 at one-third resolution). We use for the illustrative tables in the paper three images from the Middlebury dataset, Art, Dolls, and Plastic (typical for medium, low, and high compressibility) in the right view, and with half and one-third resolution (also denoted 1/2 and 1/3 in the tables).

The nature of the image, true depth image or disparity map (which is related to the depth by a simple invertible mapping) does not appear to make a difference for the compression performance of CERV, the only major place where it may have an impact is the depth value clustering and prediction, where the results may be different when working with depth or disparity values. The depth images obtained as disparity images from stereo pairs may have holes, e.g., due to occluded regions. Interestingly, the holes present in the disparity map images are not making the compression more difficult, being just another type of patches in the image. We could slightly improve the performance by taking into account the specific value of the hole patches (usually they are the only pixels having depth value equal to 0), and one can eliminate the patches with depth value 0 from the prediction of depth in Algorithm Y (i.e., by not including the value 0 in the set $\mathcal{D}_t^*$) but the improvements are usually not visible in the second significant digit of the compression ratio.

Some typical statistics of depth map images are shown in Table I. The number of patches (in four connectivity) is given in the 6th column (varying between 4000 and 10000). Couple of thousands patches contain just one pixel (column 7), a similar number of patches contain from 2 to 9 pixels (column 8) and only a few tens of patches are larger than 1000. The number of patches with just one neighbor (engulfed inside their neighbor patch) are shown in column 12, and the number

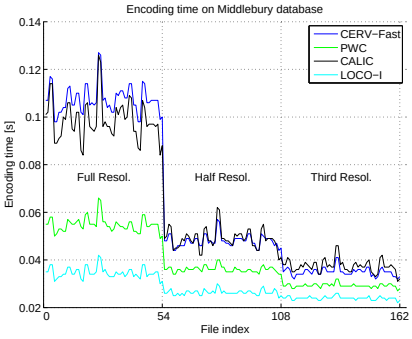


Fig. 12. Encoding time for CERV-FAST, CALIC, PWC, and LOCO-I, over the Middlebury dataset.

of patches with more than 32 neighbor patches are shown in column 17.

The algorithm for compressing crack-edges using 2D contexts is illustrated by presenting its inner variables, and also is compared against compression by chain-codes in Table II. The number of active crack-edges (in column 5) varies from a quarter to one tenth of the total number of crack-edges. The number of crack-edges which result deterministically from their contexts (in column 6) is in the range of 35%-47% of the total number of crack-edges and the number of horizontal crack-edges which are encoded in the context 0 (having T^h as a full zero vector) is in the range 12%-27%. We notice that the higher the proportion of crack edges having a deterministic context or Context 0, the better the compression of the crack-edges information (given in codelength for encoding crack-edges divided by the number of pixels in the image, in column 3 of Table II). Compression by chain-codes is illustrated in the last two columns, which show a very poor compression performance when compared to encoding using the 2D contexts. An explanation of the poor performance is the very large number of anchors needed as a header before transmitting the chain-codes; in particular, the 3OT chain-

codes require only 3 symbols, resulting without any entropy coding at about $\log_2(3) = 1.58$ bits per active link, while with the sophisticated context coding of [23] the cost reduces to about 1.1 bit per link. However, transmitting the very high number of needed anchors for the chain-codes raises the average cost to 1.4 or even 1.9 bits per active crack-edge (shown in column 10 of Table II).

The compression of depth value for each patch is illustrated in Table IV, where are given some statistics about the encoding process for the depth values. The patches are enumerated in the order of reaching them in row-wise scanning. The second column shows the ratio of the number of local patches over number of global patches (for each local patch CERV-Fast algorithm encodes the depth value, being less efficient than CERV-HiCo). The relative frequency of using each context ($m(i_C)$ is the number of occurrences of context i_C in all patches n_P) is shown in columns $3-8$ and the average codelength per pixel when encoding in each context (including the cost for encoding the switching S_t) is shown in columns $9-14$. The resulting codelength for patches per pixel is shown in 15th column, which added to the 16th column, representing the codelength for crack-edges per pixel, gives the total codelength per pixel for the depth image, in column 17th, hence columns $15-17$ are showing the split of the total necessary bitrate between coding crack-edges and coding depth values over patches.

In Figures 10-12 we present compression results over the whole dataset Middleburry. In Figure 10 we show the compression by CERV-HiCo, CERV-Fast, PWC [1], CALIC [35], and LOCO-I [36], where for better readability we order the images so that CERV-HiCo compression ratios are increasingly sorted over the group with full size resolution. The compression of the Fast and HiCo variants are not very far apart, and both are consistently better than the results of PWC, followed at a larger distance by CALIC, which in turn outperforms the LOCO-I results.

In Figure 11 we show the ratio between the compressed sizes obtained by the four CERV variants: CERV-HiCo, which is taken as a reference, achieves best compression for all files, CERV-Fast produces the largest compressed sizes, but is much faster than the other variants, and finally CERV-2 and CERV-3 are shown for clarifying the relative merit of the different steps in Algorithm A. The least favorable result of the fast variant is at 6% of the result of HiCo variant, but for most files the differences are less than 3%.

The encoding and decoding times of the CERV-Fast are very similar, since the compressor is almost symmetric. In Figure 12 are presented the encoding times for CERV-Fast over all Middlebury set, which are largely similar to the compression time of CALIC, while PWC is almost twice faster, while LOCO-I is several times faster. The tests were performed on a 64-bit system with 8 GB of memory and one Intel-i7 processor at 2GHz. The implementation of the most complex variant, CERV-HiCo, was not optimized for fast execution and is about three-four times slower than the CERV-Fast variant (results not shown). Also, in Table V are shown compression results over a subset of Middlebury dataset, from right view, comparing with results reported in previous publications or obtained with

R1. *Input:* The image $\{D_{i,j}\}$. *Output:* The image $\{Z_{i,j}\}$.
R2. Find the regions $\{R_\ell\}$ of the original image $\{D_{i,j}\}$.
R3. For each region R_ℓ find its size in number of pixels, $|R_\ell|$, and the counts $N_0(R_\ell)$ of how many times the horizontal context $T^h(H_{i,j}) = 0$ appears in the region.
R4. *Preserve large enough regions*: For all regions with $N_0(R_\ell) \geq \theta_1$ set $Z_{i,j} = D_{i,j}$ for all $(i,j) \in R_\ell$.
R5. *Re-quantize the values inside small regions*: For all regions with $N_0(R_\ell) < \theta_1$ set $Z_{i,j} = \theta_2 \lfloor D_{i,j}/\theta_2 \rceil$ for all $(i,j) \in R_\ell$.
R6. *Find new region borders and optimally assign the depth value inside them*: Find the regions $\{R'_\ell\}$ of the image $\{Z_{i,j}\}$. Find the average of the values $Z_{i,j}$ in each region R'_ℓ, round to closest integer the number and set all $Z_{i,j}$ in this region to this value.

Fig. 13. *Algorithm RQ* (θ_1, θ_2) for generating a lossy version of the original image, by preserving all regions in which the horizontal context $T^h(H_{i,j}) = 0$ appears more than θ_1 times, while for all pixels outside these regions, by quantizing the depth values with a quantization step θ_2, splitting the resulting image into constant regions, and setting optimally the depth value in each region, to the rounded average of the values in the region.

public programs, showing again the consistent performance of CERV-HiCo as the best of the methods, followed closely by CERV-Fast, while all other methods have lower compression factors. One can observe a number of regular features in Figure 10 and Table V. The compression factors obtained for full resolution images are much higher (in the range 25-65) than the compression factors obtained for small size images (in third resolution one obtain about half the compression factors of the high resolution). One can also notice that difficult images, where the general compressors obtain low compression factors, remain relatively more difficult also for the specialized depth compressors.

The sequences of depth map images Ballet and Breakdancers (in the view camera 0) have each 100 frames and we show in Table VI the average results over 100 frames, obtained by CERV, PWC, CALIC, and LOCO-I when encoding each frame independently (intra-mode) and also shown are results with the intra-schemes from [4] and [8]. Ballet and Breakdancers are similar "natural" depth images, while in the sequence Beer-garden the background is generated synthetically, making the intra-coding of it quite inefficient when compared to inter-coding. We use a very simple preprocessing along the sequence, by subtracting the current frame from the previous one and encoding the difference image by CERV, and the other lossless compressors. The results are shown for the first 100 frames of the sequence in Table VI, where we also show the results obtained for Beer-garden sequence with inter-coding schemes reported in [4] and [8].

In the file with additional results we present the complete list of results for all the images, frames, and view-points available in the mentioned data-sets. The CERV variants outperformed the CALIC and LOCO-I results for all 1862 tested images.

B. Utilizing CERV as an entropy coder for lossy depth map compression

In order to illustrate the potential usefulness of CERV beyond the lossless compression application discussed in the previous sections, we present a scheme where CERV can be used for the compression of several lossy versions of an

TABLE V
COMPRESSION RESULTS OF CERV, EXPRESSED AS *original size / compressed size*, COMPARED TO RESULTS OF PREVIOUS METHODS.

Image	Format	CERV-HiCo	CERV-Fast	PWC [1]	CALIC [35]	LOCO-I [36]	NCV [7]	Bit-plane-based + JBIG [5]
Art	1	**40.12**	39.30	37.71	28.09	12.08	33.06	30.38
Dolls	1	**27.17**	26.65	25.26	20.58	10.66	22.49	22.67
Lampshade	1	**47.78**	46.92	45.17	37.65	16.14	42.76	41.74
Moebius	1	**38.73**	37.94	36.40	29.51	14.78	33.21	31.41
Plastic	1	**66.34**	64.93	62.15	57.15	16.64	57.62	58.04
Reindeer	1	**36.94**	36.20	34.80	28.86	12.41	32.25	31.29
Art	1/2	**21.91**	21.44	20.22	15.79	8.01	18.74	16.77
Dolls	1/2	**14.69**	14.32	13.47	11.56	7.28	12.00	12.44
Lampshade	1/2	**25.91**	25.32	23.94	20.55	10.66	23.33	22.96
Moebius	1/2	**20.77**	20.25	19.11	16.56	9.94	18.03	17.19
Plastic	1/2	**34.56**	33.65	30.91	30.04	11.60	29.30	30.96
Reindeer	1/2	**20.94**	20.40	19.04	15.71	8.72	17.85	17.42
Art	1/3	**16.12**	15.75	14.58	11.45	6.26	13.93	12.04
Dolls	1/3	**10.38**	10.13	9.44	8.50	5.88	8.50	8.95
Lampshade	1/3	**20.10**	19.69	17.90	15.65	8.86	17.81	17.23
Moebius	1/3	**15.25**	14.88	13.74	12.11	7.93	13.14	12.60
Plastic	1/3	**26.38**	25.62	22.75	23.54	9.69	22.43	23.92
Reindeer	1/3	**15.59**	15.23	14.02	11.82	7.50	13.08	12.95

TABLE VI
COMPARING THE COMPRESSION RESULTS OF CERV, EXPRESSED AS BITS PER PIXEL (BPP), TO RESULTS ON THE SEQUENCES BALLET, BREAKDANCERS, AND BEER-GARDEN PREVIOUSLY REPORTED IN LITERATURE OR OBTAINED WITH PUBLIC PROGRAMS.

	Initial *bpp*	CERV-HiCo *bpp*	CERV-Fast *bpp*	PWC [1] *bpp*	CALIC [35] *bpp*	LOCO-I [36] *bpp*	Modified CABAC on H.264/AVC [8] *bpp*	CABAC on H.264/AVC [4] *bpp*	Bit-plane-based [4] *bpp*
Breakdancers	8	**0.338**	0.353	0.370	0.428	0.765	0.366	0.692	0.474
Ballet	8	**0.289**	0.302	0.317	0.383	0.750	0.346	0.627	0.437
Beergarden	8	**0.102**	0.104	0.106	0.115	0.170	0.586	0.175	0.155

original image. We use the *Algorithm RQ*, shown in Figure 13, to generate lossy versions of the original image, suitable to be encoded very efficiently by CERV. The goal is to simplify the original image, by preserving the large patches as they are, and by re-quantizing the depth values located in smaller patches, so that fewer contours and constant regions are formed after re-quantization. When deciding which patches to preserve, one can compare with a threshold either the number of pixels enclosed by the patch, $|R_\ell|$, or the number of occurrences $N_0(R_\ell)$ of the context $T^h(H_{i,j}) = 0$ inside the patch. The later was found to produce better results for the range of PSNR smaller than 70 dB, while the former is used to produce images with PSNR higher than 70 dB, using the threshold θ_1 in the range 2 to 14.

The algorithm, shown in Figure 13 involves simple selection operations of the large patches of the image, followed by quantization of the rest of the image with a uniform quantizer. The threshold θ_1, at which we select the patches to be preserved, is changed from one image to another, with values varying between 1 and the largest value of $N_0(R_\ell)$, while the quantization step θ_2 is varying between 1 and 4 for all these selection of patch sizes. When the threshold θ_1 is so large that no patches will be preserved in $\{Z_{i,j}\}$, the whole original image is re-quantized; in this case we use quantization step θ_2 between 5 and 14.

Each lossy image is then encoded by CERV and results in a point in the rate-distortion plane. After all points are obtained, we remove any point $(PSNR_i, L_i)$ performing strictly worse than another existing point, $(PSNR_j, L_j)$ (i.e., if $L_i > L_j$ and $PSNR_i \leq PSNR_j$). The remaining points in the RD plane form a smooth curve, which is plotted with square marked blue lines in Figure 14. Also shown, for the same lossy approximation images, are the results of PWC coder. Although the presented method for obtaining lossy approximations is not using advanced optimization techniques, the results of RQ+CERV, and even of RQ+PWC show very competitive results, being the best of all other tested methods for PSNR larger than 50-60 dB, and in the case of Aloe image the performance is the best over the whole rate range, except a single point where the method from [38] gives slightly better results.

The topic of lossy compression using results of CERV can be further investigated, to include a more involved optimization when generating the approximation images. However, from the presented results one can see that CERV is a very effective entropy coder of the suitable approximations of the original image for a wide range of rates, its use being promising also in lossy compression.

VIII. CONCLUSIONS

The introduced CERV algorithms are shown to consistently take better into account the specificities of depth map images when compared to recent methods specifically designed for lossless depth coding, and compared to the established algorithms CALIC and JPEG-LS for general lossless image compression. CERV performs better for depth images also when compared to the algorithm PWC, which was specifically designed for palette images. The similarities and differences between PWC and CERV algorithms are discussed, clarifying

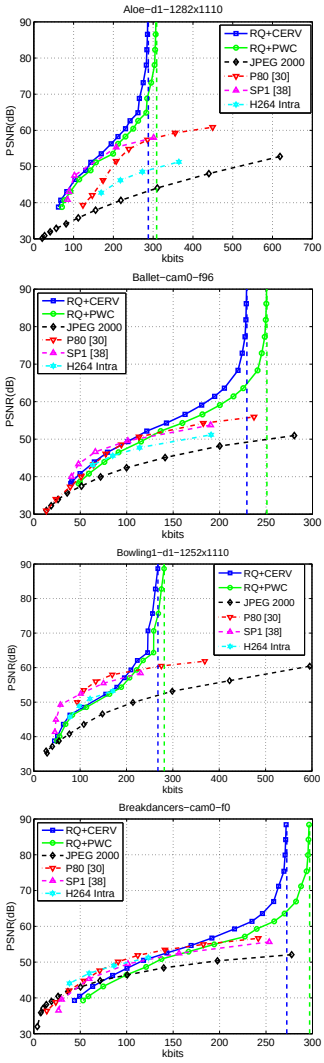


Fig. 14. Rate-distortion plots for lossy compression by several methods and by the simple RQ-CERV and RQ-PWC described in Section VII for all images from [30]. The losses compression values obtained by CERV and PWC are marked by vertical lines. The results from [30] are also cited here. The images are: Aloe (full resolution, view 1), Ballet (camera 0, frame 96), Bowling (full resolution, view 1), and Breakdancers (camera 0, frame 0).

the different ranking of the two methods for the type of images for which they are intended.

In lossless mode, typical average codelengths obtained by the coding scheme are about 1.1-1.4 bits per active crack-edge and about 2-2.7 bits per depth value of a patch, resulting in an overall compressed size of about 0.2-0.8 bits per pixels, or, equivalently, in reductions of the original image size of 10 to 65 times, depending on the density and shape of contours in the image. The usage of CERV as an entropy coder in a lossy compression scheme is illustrated using a very simple mechanism for generating lossy images, resulting in a lossy compressor having very good performance for the high PSNR range.

REFERENCES

[1] P. Ausbeck Jr., "The piecewise-constant image model," *Proceedings of the IEEE*, vol. 88, no. 11, pp. 1779–1789, Nov 2000.

[2] J. Rissanen and G. Langdon, "Arithmetic coding," *IBM Journal of Research and Development*, vol. 23, pp. 149–162, 1979.

[3] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, pp. 520–540, 1987.

[4] K. Kim, G. Park, and D. Suh, "Bit-plane-based lossless depth-map coding," *Optical engineering*, vol. 49, no. 6, pp. 067 403–1–10, 2010.

[5] M. Zamarin and S. Forchhammer, "Lossless compression of stereo disparity maps for 3D," in *2012 IEEE International Conference on Multimedia and Expo Workshops*, 2012, pp. 617–622.

[6] I. Schiopu and I. Tabus, "MDL segmentation and lossless compression of depth images," in *Fourth Workshop on Information Theoretic Methods in Science and Engineering (WITMSE)*, Helsinki, Finland, August 2011.

[7] ——, "Depth image lossless compression using mixtures of local predictors inside variability constrained regions," in *Proc. of the 5th International Symposium on Communications, Control and Signal Processing (ISCCSP 2012)*, Rome, Italy, May 2012.

[8] J. Heo and Y.-S. Ho, "Improved context-based adaptive binary arithmetic coding over H.264/AVC for lossless depth map coding," *IEEE Signal Processing Letters*, vol. 17, no. 10, pp. 835–838, 2010.

[9] ——, "Improved CABAC design in H.264/AVC for lossless depth map coding," in *Proc. IEEE Int Multimedia and Expo (ICME) Conf*, 2011, pp. 1–4.

[10] S. Mehrotra, Z. Zhang, Q. Cai, C. Zhang, and P. A. Chou, "Low-complexity, near-lossless coding of depth maps from kinect-like depth cameras," in *Proc. IEEE 13th Int Multimedia Signal Processing (MMSP) Workshop*, 2011, pp. 1–6.

[11] L. Cappellari, C. Cruz-Reyes, G. Calvagno, and J. Kari, "Lossy to lossless spatially scalable depth map coding with cellular automata," in *Proc. DCC '09. Data Compression Conf*, 2009, pp. 332–341.

[12] S. Yea and A. Vetro, "Multi-layered coding of depth for virtual view synthesis," in *Proc. Picture Coding Symp. PCS 2009*, 2009, pp. 1–4.

[13] V. Ratnakar, "RAPP: Rapp: lossless image compression with runs of adaptive pixel patterns," in *Thirty-Second Asilomar Conference on Signals, Systems & Computers*, 1998.

[14] "JBIG, progressive bi-level image compression," ISO/IEC Int. Standard 11544, 1993.

[15] P. Howard, F. Kossentini, B. Martins, S. Forchhammer, and W. Rucklidge, "The emerging JBIG2 standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 7, pp. 838–848, Nov 1998.

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

[17] A. Akimov, A. Kolesnikov, and P. Fränti, "Lossless compression of map contours by context tree modeling of chain codes," *Pattern Recognition*, vol. 40, pp. 944–952, 2007.

[18] H. Nicolas, S. Pateux, and D. Le Guen, "Minimum description length criterion and segmentation map coding for region-based video compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 2, pp. 184–198, 2001.

[19] Q. Luo and T. Khoshgoftaar, "Unsupervised multiscale color image segmentation based on MDL principle," *IEEE Transactions on Image Processing*, vol. 15, no. 9, pp. 2755–2761, September 2006.

[20] G. Schuster and A. Katsaggelos, "A video compression scheme with optimal bit allocation among segmentation, motion, and residual error," *IEEE Transactions on Image Processing*, vol. 6, no. 11, pp. 1487–1502, 1997.

[21] C. L. B. Jordan, F. Bossen, and T. Ebrahimi, "Scalable shape representation for content based visual data compression," in *IEEE International Conference on Image Processing*, vol. 1, Santa Barbara, CA, 1997, pp. 512–515.

[22] I. Daribo, G. Cheung, and D. Florencio, "Arithmetic edge coding for arbitrarily shaped sub-block motion prediction in depth video compression," in *IEEE International Conference on Image Processing*, Orlando, USA, October 2012.

[23] I. Tabus and S. Sarbu, "Optimal structure of memory models for lossless compression of binary image contours," in *IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP 2011)*, Prague, Czech Republic, May 2011, pp. 809–812.

[24] S. Forchhammer and J. Salinas, "Progressive coding of palette images and digital maps," in *Data Compression Conference*, 2002.

[25] S. R. Tate, "Lossless compression of region edge maps," Deparment of Computer Science, Duke University, Durham, NC, Tech. Rep., 1992.

[26] Y. Morvan, D. Farin, and P. de With, "Depth-image compression based on an R-D optimized quadtree decomposition for the transmission of multiview images," in *Proc. IEEE Int. Conf. Image Processing ICIP*, vol. 5, October 2007, pp. V–105 – V–108.

[27] S. Milani and G. Calvagno, "A depth image coder based on progressive silhouettes," *IEEE Signal Processing Letters*, vol. 17, no. 8, pp. 711–714, August 2010.

[28] M.-K. Kang and Y.-S. Ho, "Depth video coding using adaptive geometry based intra prediction for 3-D video systems," *IEEE Transactions on Multimedia*, vol. 14, no. 1, pp. 121–128, 2012.

[29] F. Jäger, "Contour-based segmentation and coding for depth map compression," in *Proc. of IEEE Visual Communications and Image Processing VCIP '11*, Nov 2011.

[30] 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.

[31] J. Rissanen, "A universal data compression system," *IEEE Transactions on Information Theory*, vol. 29, September 1983.

[32] "Evaluation of cost functions for stereo matching," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN, June 2007.

[33] M. Weinberger, J. Rissanen, and R. Arps, "Applications of universal context modeling to lossless compression of gray-scale images," *IEEE Transactions on Image Processing*, vol. 5, April 1996.

[34] A. Akimov, A. Kolesnikov, and P. Fränti, "Lossless compression of color map images by context tree modeling," *IEEE Transactions on Image Processing*, vol. 16, pp. 114–120, January 2007.

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

[36] 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, pp. 1309–1324, August 2000.

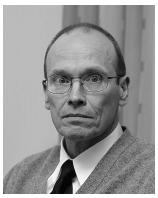
[37] D. Speck, "Local activity level classification model for continuous-tone coding," ISO/IEC JTC1/SC29/WGI, Tech. Rep., 1995.

[38] P. Zanuttigh and G. Cortelazzo, "Compression of depth information for 3D rendering," in *3DTV Conference: The True Vision-Capture, Transmission and Display of 3D Video*, Potsdam, Germany, May 2009.

Ioan Tabus received the M.S. degree in electrical engineering in 1982 from the "Politehnica" University of Bucharest, Romania, and the Ph.D. degree (with honors) from Tampere University of Technology (TUT), Finland, in 1995. He was holding teaching positions between 1984 to 1995 with the Department of Control and Computers, "Politehnica" University of Bucharest. From 1996 he was a Senior Researcher and since January 2000, he has been a Professor with the Department of Signal Processing at TUT. His research interests include audio, image and data compression, image processing, and genomic signal processing. He is coauthor of two books and more than 210 publications in the fields of signal compression, image processing, bioinformatics, and system identification. He is a Senior Member of IEEE and was an Associate Editor for IEEE Transactions on Signal Processing between 2002-2005. He currently is the Editor-in-Chief of EURASIP Journal on Bioinformatics and Systems Biology. Dr. Tabus is co-recipient of 1991 "Traian Vuia" Award of the Romanian Academy.

Ionut Schiopu was born in Caracal, Romania, in 1986. He received the B.Sc. degree in Automatic Control and Applied Informatics in 2009 and the M.Sc. degree in System Engineering in 2011, both from the "Politehnica" University of Bucharest, Bucharest, Romania. Since 2011, he works as a Researcher at the Department of Signal Processing, Tampere University of Technology, Tampere, Finland, where he is currently pursuing the Ph.D. degree. His research interests include depth image compression and depth video coding.

He was a Teaching Assistant with the Department of Automatic Control and Systems Engineering, Faculty of Automatic Control and Computers, "Politehnica" University of Bucharest, between 2009 and 2011.



Jaakko Astola received the Ph.D. degree in mathematics from Turku University, Finland, in 1978. From 1976 to 1977 he was with the Research Institute for Mathematical Sciences of Kyoto University, Kyoto, Japan. Between 1979 and 1987 he was with the Department of Information Technology, Lappeenranta University of Technology, Lappeenranta, Finland. In 1984 he worked as a visiting scientist in Eindhoven University of Technology, The Netherlands. From 1987 to 1992 he was Associate Professor in Applied Mathematics at Tampere University, Tampere, Finland. From 1993 he has been Professor of Signal Processing at Tampere University of Technology. His research interests include signal processing, coding theory, switching theory, spectral techniques, and statistics. He is a Fellow of the IEEE, SPIE and EURASIP.

Publication 4

Copyright ©2013 IEEE. Reprinted, with permission, from

[P4] I. Schiopu and I. Tabus. Lossy depth image compression using greedy rate-distortion slope optimization. *IEEE Signal Processing Letters (IEEE SPL)*, 20(11):1066–1069, November 2013.

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Tampere University of Technology's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

Lossy Depth Image Compression using Greedy Rate-Distortion Slope Optimization

Ionut Schiopu*, *Student Member, IEEE,* Ioan Tabus, *Senior Member, IEEE*

***Abstract*—We introduce a method to create lossy versions of one image, either by successively merging the constant regions of the original image, or by iteratively splitting the regions from a created lossy image using horizontal or vertical line segments. Merging and split decisions are greedily taken, according to the best slope towards next point in the rate-distortion curve. For each created lossy image, the region contours and the optimal depth values can be entropy coded in three ways: with a new algorithm, or with two existing lossless coding algorithms. The obtained results compare favorably with the existing lossy methods.**

EDICS Category: IMD-CODE Image/video coding and transmission

I. Introduction

THE next generation technologies in the entertainment field are the 3DTv and the free-view point video (FFV), in which an important role has the compression of depth map images and sequences. Consequently, depth image compression is an active field, with many potential applications.

Lossy depth image compression was proposed in the past using several approaches. A first approach is to obtain a decomposition of the depth image using a binary tree triangular decomposition [1], or a quad-tree decomposition of the depth image with a wedgelet and platelet based approach that fills regions [2]. A second approach is to obtain a segmentation of the images: in [3] a segmentation of the color image is used for segmenting the depth image; in [4] the method starts from an over-segmented depth image and merges regions according to the average depth value for each region and the number of objects assumed to exist in the depth map; in [5] the depth image is segmented into objects, and the contour and the depth of the objects are compressed using various methods; in [6], [7] the image is compressed by two image pyramid structures, one for arc breakpoints and other for sub-band samples.

The letter presents an algorithm that generate sequences of lossy depth images for the full range of rates and proposes suitable entropy coders. The algorithm is not scalable, however suitable modifications can make it scalable by decreasing the performance. We show how to iteratively generate sequences of depth images using a greedy best slope criterion by merging regions, Subsection II-A, or splitting regions, Subsection II-B. Suitable entropy coding procedures are presented in Subsection II-C. Obtained results for compressing commonly used depth images are presented in Section III.

Manuscript received August 30, 2013.

Copyright (c) 2012 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

I. Schiopu and I. Tabus are with the Department of Signal Processing, Tampere University of Technology, Tampere, Finland, corresponding author: Ionut Schiopu, e-mail: ionut.schiopu@tut.fi (see http://www.cs.tut.fi/~schiopu/).

II. Description of the greedy slope optimization (GSO) method

In the GSO method we construct two sequences of lossy images, each image being composed of connected regions, and each region having the same reconstructed depth value.

The first sequence starts with the original image, partitioned into its constant connected components, and the next lossy images are generated by merging at each step the two regions, for which the slope of the rate-distortion (RD) curve is optimal. The greedy merging process results in a good trade-off rate-distortion for the medium and high rates, until the number of regions is in the order of tens. The most important image contours of the initial depth image are contained by the last lossy images in the sequence, any of them can be selected to define the template for the second phase. This first phase sequence ends with a lossy image with only two regions.

The second sequence of lossy images is obtained starting from the chosen template, and advances by splitting the regions, using horizontal or vertical line segments. For each obtained lossy image we encode the contours of the regions and the depth value of each region using entropy coding. In general the contours of the regions are encoded using chain-codes, except the straight line segments which are encoded more efficiently according to their position inside a region.

Each depth image, Z, is a matrix with n_r rows and n_c columns. An integer $Z(x,y) \in \{0,1,\ldots,2^{B-1}\}$ is stored for each pixel (x,y), using B bits, representing the distance between the camera lens and a point in the scene. In this letter we illustrate the method over images containing integers stored using $B = 8$ b.

A. GSOm: GSO with region merging

We denote Z the current lossy reconstruction of the original depth image Z_0, and explain how the next image in the sequence, Z', is constructed.

The image Z is partitioned as $\cup_{i=1}^{n_\Omega}\Omega_i$, where each of the n_Ω regions is a set of pixels connected in 4-connectivity. Initially, the partition of Z_0 includes all maximal constant regions in the image. A maximal region Ω_i has the same depth value d_i for all its pixels, and every pair of neighboring regions has distinct depth values. The partition into regions is efficiently encoded using the 3OT chain-code representation of the crack-edges separating the neighboring pixels belonging to different

regions [8]. For each pair of neighboring regions (Ω_i, Ω_j) their common contour segment is denoted $\Gamma_{i,j}$, formed of 3OT chain-codes. The partition is encoded using the sequence of chain-codes $\Gamma(Z)$, which is the union of all $\Gamma_{i,j}$.

The next image in the sequence, Z', is obtained from Z, by merging the pair $(\Omega_{i^*}, \Omega_{j^*})$. The new partition is specified by the contours $\Gamma(Z')$, having one segment less, $\Gamma(Z') = \Gamma(Z) \setminus \Gamma_{i^*,j^*}$. For brevity the pairs of region indices are denoted $p = (i, j)$. When merging two regions we aim at getting the best trade-off between the estimated saving of bits ΔR, due to not encoding Γ_p, and the increase in distortion ΔD, due to having a poorer reconstruction in the common region $\Omega'_\ell = \Omega_i \cup \Omega_j$. The unique depth value d'_ℓ is optimally set as rounded mean value of the depth inside the region Ω'_ℓ. In the rest of the letter the reconstructed depth for Ω_i or Ω'_i is d_i or d'_i respectively.

In order to select the optimal merging we evaluate ΔR by using a model for the contour codelength, $C(\Gamma_p) = C_1 \cdot L(\Gamma)$, where C_1 is a constant representing the cost of encoding a chain-code link and $L(\Gamma_p)$ is the number of links in the contour Γ_p. In this letter we used $C_1 = 1.5$ b and for the cost of encoding a depth value we have used the estimate $C_2 = 8$ b. The actual entropy coding will produce slightly different values, but they are not known at this stage. We estimate the rate variation, ΔR, when we eliminate the contour Γ_p:

$$\Delta R_p = C_1 \cdot L(\Gamma_p) + C_2. \tag{1}$$

The change in distortion $\Delta D = MSE(Z') - MSE(Z)$ depends only on the regions involved in merging: $\Omega_i \cup \Omega_j \to \Omega'_\ell$, due to the iterative nature of constructing the images Z and Z'. In order to evaluate efficiently ΔD, we introduce for any region Ω_k with m_k pixels the following variables: the sum of original depth values $\phi_k = \sum_{(x,y)\in\Omega_k} Z_0(x, y)$, the sum of squared depth, $\varphi_k = \sum_{(x,y)\in\Omega_k} Z_0(x, y)^2$, and the sum of squared reconstruction errors $\rho(\Omega_k) = \sum_{(x,y)\in\Omega_k} (Z_0(x, y) - d_k)^2 = \varphi_k - 2\phi_k d_k + m_k d_k^2$, resulting in the mean square error over the region $MSE_k = \frac{1}{n_r n_c}\rho(\Omega_k)$ and $MSE(Z) = \sum_{k=1}^{n_\Omega} MSE_k$. The merging of the pair of regions (Ω_i, Ω_j) has the following consequences: the contour Γ_p is removed; a new region Ω'_ℓ is created, having the variables $m'_\ell = m_i + m_j$, $\phi'_\ell = \phi_i + \phi_j$, $d'_\ell = \lfloor \frac{\phi'_\ell}{m'_\ell} \rceil$; a change in distortion is introduced:

$$\begin{aligned}\Delta D_p &= \frac{\rho(\Omega'_\ell)}{n_r n_c} - \frac{\rho(\Omega_i) + \rho(\Omega_j)}{n_r n_c} \\ &= \frac{m'_\ell d'^2_\ell - m_i d_i^2 - m_j d_j^2 + 2(d_i\phi_i + d_j\phi_j - d'_\ell\phi_{\ell'})}{n_r n_c}.\end{aligned} \tag{2}$$

The pair of regions to merge, $(\Omega_{i^*}, \Omega_{j^*})$, is chosen from all available pairs of neighbor regions, by greedily minimizing the slope between the points Z and Z' in the RD plot (Fig. 1(a)), i.e. by choosing the pair $p^* = (i^*, j^*)$, for which

$$\lambda_p = \tan\alpha_p = \frac{\Delta D_p}{\Delta R_p} \tag{3}$$

is minimum, resulting in the smallest slope $\lambda^* = \tan\alpha^*$.

Although the number of available pairs of neighboring regions n_Γ in the image Z_k (the same as the number of contour segments at the current step) is large for the first images, when moving from Z_k to Z_{k+1} the value of λ_p will not change, except for the pairs formed by the new region Ω'_ℓ with its neighbors. To avoid making updates after each merging step, the values of λ_p for all possible region pairs are sorted increasingly in the vector $\boldsymbol{\lambda} = [\lambda_{p_1}, \lambda_{p_2}, \ldots, \lambda_{p_{n_\Gamma}}]$, corresponding to the list of pairs of regions truncated to the first n_M pairs, $[(i_1, j_1), (i_2, j_2), \ldots, (i_{n_M}, j_{n_M})]$. The pairs from the list are marked for merging, starting from the first and continuing sequentially, except for pairs (i_k, j_k) having either i_k or j_k an element in an earlier pair $(i_{k-\tau}, j_{k-\tau})$. Additionally, when the size of a region Ω_{i_k} is smaller than three pixels, the index j_k is allowed to appear in following marked pairs. In this letter we used $n_M = 100$, if $n_\Omega/\sigma > 100$; $n_M = \lfloor n_\Omega/\sigma \rfloor$, if $1 \le n_\Omega/\sigma \le 100$; and $n_M = 1$, if $n_\Omega/\sigma < 1$; where $\sigma = 10$.

For obtaining operating points in the RD curve near the lossless rate, we apply a different method, not by merging regions, but by modifying slightly the contours as follows: if at least three neighbors of a current pixel with depth d_i, have the same depth value, d_j, and if $|d_i - d_j| \le 2$, then we set the depth value of the current pixel as d_j. The process is done column-wise sequentially and intermediate images are stored along the process when MSE reached an empirically selected value, so that $\Delta PSNR \in [3, 7]$.

B. GSOs: GSO with region splitting

For the second phase (GSOs), we start from one of the images obtained in GSOm, where the number of regions is small (here we used templates with two and nine regions) and call this image a template, $Z_{templ} = Y_0$ (see Fig. 1(a)). From each such template, which corresponds to very low bitrates in the RD curve obtained for GSOm, we create a sequence of images, $Y_0, Y_1, \ldots$, where the regions of the image Y_k are further split with horizontal or vertical lines to obtain the regions of the image Y_{k+1}. To recreate the contour at the decoder we first encode the contours of Y_0, the same way as for GSOm, and additionally we encode: the decisions to split or not a region; the orientation (vertical or horizontal); and location of the line segment for each decided split.

The image Y_{k+1} is obtained from the image Y_k, by fixing a target slope, λ_{k+1}, and splitting all the regions iteratively, starting from the regions of Y_k until no more splits are allowed at given slope λ_{k+1}, as explained below. The decisions to split form a binary tree, traversed in depth-first order. The greedy decisions are choosing between splitting with a vertical line segment at a position J or a horizontal line segment at position I from all possible positions determined by the boundaries of the region Ω_i ($I_{min} \le I < I_{max}$, $J_{min} \le J < J_{max}$). The resulting regions are denoted $\Omega^I_{i_1}$ and $\Omega^I_{i_2}$, for horizontal splits, while $\Omega^J_{i_1}$ and $\Omega^J_{i_2}$, for vertical splits.

The criterion to be maximized is the slope, in the RD plot, from the set of slopes for all possible splits:

$$\Psi = \left\{ \frac{\Delta D_h(I)}{\Delta R_h(I)} \right\}_{I_{min}\le I<I_{max}} \bigcup \left\{ \frac{\Delta D_v(J)}{\Delta R_v(J)} \right\}_{J_{min}\le J<J_{max}},$$

where the changes in distortion are evaluated as $\Delta D_h(I) = \rho(\Omega_i) - \rho(\Omega^I_{i_1}) - \rho(\Omega^I_{i_2})$, $\Delta D_v(J) = \rho(\Omega_i) - \rho(\Omega^J_{i_1}) - \rho(\Omega^J_{i_2})$, and the additional rate is estimated as $\Delta R_h(I) = C_3 + \log_2(I_{max} - I_{min})$, $\Delta R_v(J) = C_3 + \log_2(J_{max} - J_{min})$ for horizontal, respectively vertical split. The constant C_3 includes

the cost of an additional depth value needed after the split and the cost for encoding the decision. In this letter we use $C_3 = 8$ b, although the true encoding process uses adaptive Markov modeling and obtains better rates, but the choice of C_3 was found to not influence the results too much. If the maximum slope λ^* from the set Ψ is higher than the target slope, $\lambda^* > \lambda_{k+1}$, we split the region with a horizontal or vertical line segment, by encoding the decision and the row or column index selected according to the maximum slope λ^*. The decisions at each region are represented with the variable ξ as follows: $\xi = 1$ for no-split decision ($\lambda^* \leq \lambda_{k+1}$); $\xi = 2$ for vertical split; and $\xi = 3$ for horizontal split. The sequence of variables ξ collected along the split process at the encoder is transmitted to the decoder who can reproduce the splitting process. The variables ξ are encoded using order two adaptive Markov arithmetic coding.

If $\xi = 1$, we compress d_ℓ using the up and left neighboring region values (see Subsection II-C). If $\xi = 2$, we compress the row index I^* using $\log_2(I_{max} - I_{min})$ bits, split the current region into two regions having same column indices and the following row indices: $[I_{min}, I^*]$ for $\Omega^I_{i_1}$; $[I^* + 1, I_{max}]$ for $\Omega^I_{i_2}$. The case $\xi = 3$ for compressing and performing the vertical splits is similar to the previous case. The process continues by applying the algorithm first for Ω_{i_1} and all its descend regions, until no more splits occur in this branch, and only then Ω_{i_2} is processed similarly.

For minimizing the distortion, the reconstruction value in each region is assigned as follows: the depth value $d_\ell - 1$ is set for the first line and the first column of the current region if the neighboring region value d_i is $d_i < d_\ell$; similarly it is set to $d_\ell + 1$ if $d_i > d_\ell$. For a better compression the true contour is smoothed as in Subsection II-A, this time using the constraint $|d_i - d_j| < \gamma$, where γ increases until the value 100.

In Fig. 1(a) we present a RD plot that illustrates the principle of the method, using real points corresponding to the generated sequences of lossy images for *Breakdancers* (including also JPEG2000 results), and the angle α^* that minimized the slope λ^* for generating the next image Z'. Fig. 1(b) shows an example of segmentation for a lossy image from GSOs, using the template with nine regions, for the initial image *Breakdancers*, compressed at 0.039 bpp and having PSNR = 45.215 dB. In Fig. 1(b) can be seen the two types of contours: straight lines (vertical and horizontal) and non-straight true object boundaries. In our experiments we combine the RD curves S_1 (obtained when starting from the template having nine regions) and S_2 (having a starting template with two regions). The transition from S_1, which is better in the range of rates $[0.025\ , 0.1]$ bpp, to S_2 is performed empirically around the point having the slope $\lambda = 200$.

C. Entropy coding

For compressing the contour for the sequence of images $Z_1, Z_2, \ldots$, we used a similar algorithm as in [8], to which we added improvements regarding the searching of the next position of the contour. The encoder transmits a matrix containing anchor points and junction points for 3OT chain-codes, and the all 3OT chain-codes. Here we modified the order in which the 3OT segments are concatenated, obtaining a better compression by using the following approach: start generating a new 3OT chain code segment by choosing the next link with the priority list: right, up, left, down. We refer to [8] for the detailed algorithm of encoding the region contours.

For encoding the depth values from the sequence of images $Z_1, Z_2, \ldots$, we use the method presented in [9], where the depth value of a current region, d_i, is encoded using its position in the list of possible depth values, ζ, generated using the known values $\mathbf{d}$ of the neighboring regions. The contour of Z_k is obtained using 3OT chain-codes which guarantees that $d_i \notin \mathbf{d}$. The initial algorithm excludes $\mathbf{d}$ from ζ, therefore d_i is encoded using $\zeta' = \zeta \cap \mathbf{d}$. This entropy coding method for contours & depth values is dubbed Chain-Code–Value (CCV).

For encoding the depth values from the sequence of images $Y_0, Y_1, \ldots$, a modified version of the algorithm is used: d_i is compressed using the up and left neighboring region values, and because the horizontal and vertical lines do not guarantee $d_i \notin \mathbf{d}$, d_i is encoded using ζ (not ζ'). We denoted this modified algorithm Chain-Code-Line–Value (CCLV).

III. Experimental results

In this letter we present comparative results for six commonly used depth images: *Breakdancers*, frame 0 of cam0 from *breakdancers* dataset [10]; *Ballet*, frame 96 of cam0 from *ballet* dataset [10]; *Art, Aloe, Baby1* and *Bowling1*, full-size resolution, left view (disp1.png) from *Middlebury* dataset [11].

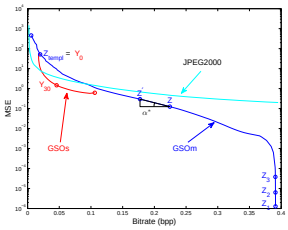
The algorithms were implemented in C. For the arithmetic coding routines we used the implementation from Witten *et al.* For GSOm a new lossy image is saved when $\Delta PSNR > 0.5$ dB between two consecutive lossy images or if $n_\Omega < 10$; while for GSOs, 50 values (selected empirically) are used for the stop criterion λ_k so that $\Delta PSNR > 0.1$ dB. Regarding the runtime, the current not optimized version of the method generates the last image for GSOm in 1.2 s for *Breakdancers* (1.9 s for *Aloe*) and compress a Y_k image in less then 0.65 s.

In Fig. 1(c)-(i)[1] the GSOm sequence is compressed using three entropy coders: CCV; a recent algorithm for lossless compression of depth image, Crack-Edge-Region–Value (CERV) [9]; a palette image coder, the piecewise-constant image model (PWC) [12]; the GSOs sequence is compressed using CCLV. The results are compared also with: the Platelet algorithm [2]; an algorithm which uses color information [3], denoted here "Color & Depth"; the Breakpoint_Geom algorithm [6]; the "Proposed_80" algorithm from [7], denoted here "P80"; H.264 from [6] and [7]; our previous results [8]; the JPEG2000 standard. The Fig. 1(d) shows the results for the *Breakdancers* image for low bitrates, below 0.12 bpp. The figure shows that GSOs + CCLV and entropy coding the GSOm sequence obtain the best results comparing with other algorithms listed above. The CCV algorithm is not the best lossless compressor but under 65 dB obtains the best, or similar, results comparing with the CERV algorithm.

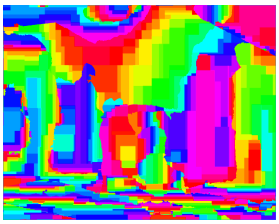
IV. Conclusions

The GSO algorithm presented in this letter uses the greedy slope optimization for generating sequences of lossy images

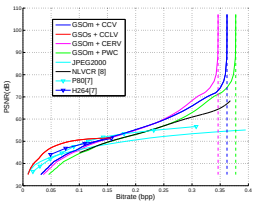
[1]For more results and some lossy images see www.cs.tut.fi/~schiopu/GSO



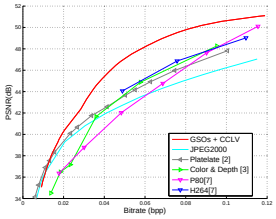
(a) Rate-Distortion points



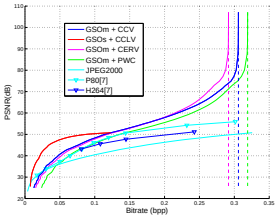
(b) A GSOs segmentation



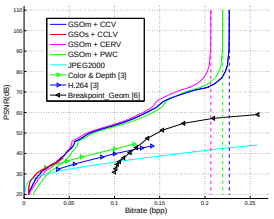
(c) *Breakdancers*, BD-PSNR= 2.89 dB, ref. met. [7]



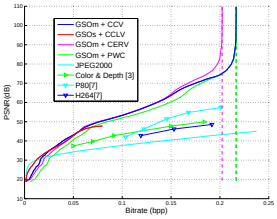
(d) *Breakdancers*, BD-PSNR= 1.72 dB, ref. met. [2]



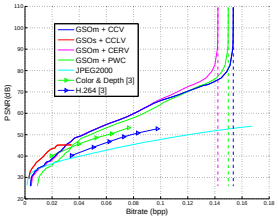
(e) *Ballet*, BD-PSNR= 7.08 dB, ref. met. [7]



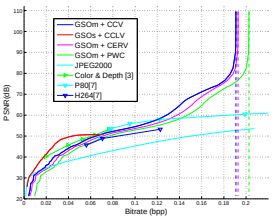
(f) *Art*, BD-PSNR= 6.49 dB, ref. met. [3]



(g) *Aloe*, BD-PSNR= 10.63 dB, ref. met. [3]



(h) *Baby*, BD-PSNR= 4.64 dB, ref. met. [3]



(i) *Bowling*, BD-PSNR= 3.48 dB, ref. met. [3]

Fig. 1. (a) The rate-distortion points corresponding to the sequence of lossy images for *Breakdancers*: GSOm, with blue; GSOs, with red; JPEG2000 with cyan. From the generic image Z to the next image Z' the merging pair is chosen so that the angle α is minimized, obtaining α^* and corresponding slope λ^*. (b) Segmentation for a lossy image from GSOs, using Z_{templ} with nine regions, for the initial image *Breakdancers*, at Bitrate $= 0.039$ bpp, PSNR $= 45.215$ dB. (c)-(i) PSNR vs Bitrate plots for the two sequences, GSOm and GOSs, compressed using the entropy coders: CCV, CCLV, CERV, and PWC, and results previously reported in literature, for a set of 6 images. For each plot the listed BD-PSNR [13] values are showing the average improvement in PSNR of GSOm + CCV (GSOs + CCLV) with respect to the best previously reported method, which is named Reference Method.

by merging (GOSm) or splitting (GOSs) regions. The results showed that the compression of the sequences using suitable entropy coder (CCV/CCLV, CERV, PWC) obtains better results over the full range of bitrates, when compared with previously reported results in literature.

REFERENCES

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

[2] 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*, vol. 5, San Antonio, TX, USA, Sep. 2007, pp. V–105 – V–108.

[3] 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.

[4] S. Milani and G. Calvagno, "A depth image coder based on progressive silhouettes," *IEEE Signal Process. Lett.*, vol. 17, no. 8, pp. 711–714, Aug. 2010.

[5] B. Zhu, G. Jiang, Y. Zhang, Z. Peng, and M. Yu, "View synthesis oriented depth map coding algorithm," in *Proc. APCIP*, vol. 2, Jul. 2009, pp. 104–107.

[6] R. Mathew, P. Zanuttigh, and D. Taubman, "Highly scalable coding of depth maps with arc breakpoints," in *Proc. Data Compress. Conf*, Apr. 2012, pp. 42 – 51.

[7] R. Mathew, D. Taubman, and P. Zanuttigh, "Scalable coding of depth maps with r-d optimized embedding," *IEEE Trans. Image Process.*, vol. 22, no. 5, pp. 1982–1995, May 2013.

[8] I. Schiopu and I. Tabus, "Lossy and near-lossless compression of depth images using segmentation into constrained regions," in *Proc. 20th EUSIPCO*, Bucharest, Aug. 2012, pp. 1099 – 1103.

[9] I. Tabus, I. Schiopu, and J. Astola, "Context coding of depth map images under the piecewise-constant image model representation," *IEEE Trans. Image Process.*, Accepted June 2013.

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

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

[12] P. Ausbeck Jr., "The piecewise-constant image model," *Proc. IEEE*, vol. 17, no. 8, pp. 1779 – 1789, Nov. 2000.

[13] K. Senzaki, *BD-PSNR/Rate computation tool for five data points*, JCT-VC of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Jul. 2011.

Publication 5

Copyright ©2014 IEEE. Reprinted, with permission, from

[P5] I. Schiopu and I. Tabus. Anchor points coding for depth map compression. In *Proc. IEEE International Conference on Image Processing (IEEE ICIP)*, pages 5626–5630, Paris, France, October 2014.

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Tampere University of Technology's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

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 $\Gamma_k = \begin{bmatrix} P_1 & P_2 & \cdots & P_{n_{\Gamma_k}} \end{bmatrix}^T$ of n_{Γ_k} adjacent vertices.

The 3OT chain-code representation [14] codify each Γ_k by a vector $S_k = \begin{bmatrix} s_1 & s_2 & \cdots & s_{n_{\Gamma_k}-2} \end{bmatrix}^T$, 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 $S = \begin{bmatrix} S_1^T & S_2^T & \cdots & S_{n_\Gamma}^T \end{bmatrix}^T$.

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.

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

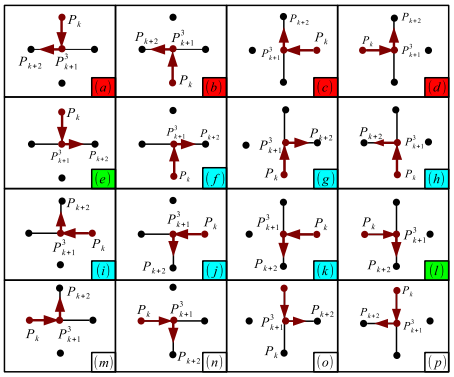


Fig. 2. All possible ways of selecting the next adjacent vertex to visit for a P_k^3 vertex. An arrow shows the next vertex to visit as part of a current Γ_k, a red (black) dot is a visited (unvisited) vertex, and a red (black) line is a visited (unvisited) contour edge.

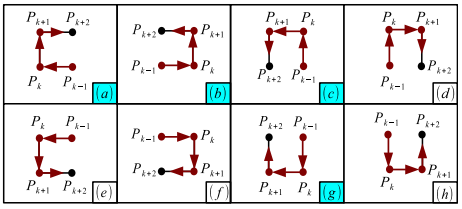


Fig. 3. Cases where a chain-code symbol 2 encodes P_{k+2}. An arrow shows the next adjacent vertex to visit as part of a current Γ_k, a red (black) dot is a visited (unvisited) vertex, and a red (black) line is a visited (unvisited) contour edge.

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.

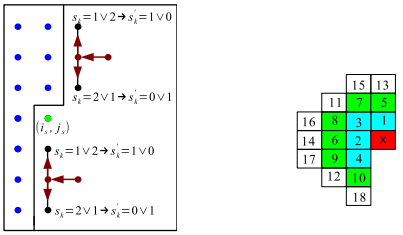
3. DEPTH MAP COMPRESSION

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

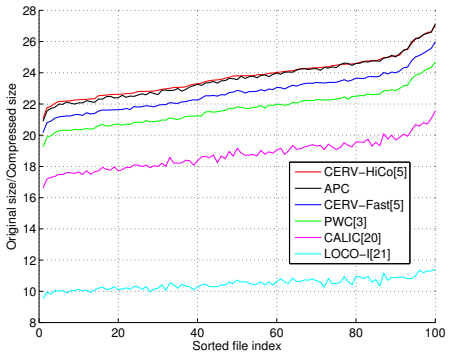
The vector Φ is encoded using the Context Tree algorithm with the tree-depth $d_{\mathcal{T}} = 17$. The tests showed that Φ contains about 10% symbols 1. Because the optimal context tree is unbalanced, i.e. it has only one long branch (branch '0') with a few leaf nodes, the threshold at which the symbol 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

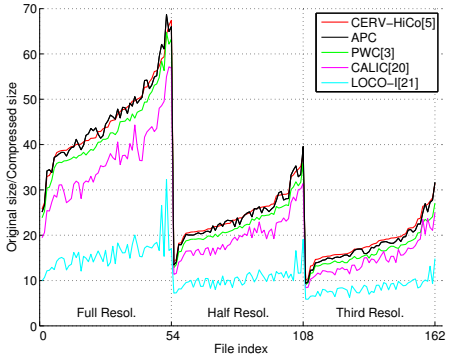
Matrix Υ is encoded the last in column-wise scanning, using a bi-dimensional context of length $d_{\mathcal{T}} = 18$, see Fig. 4(b). After pruning, the optimal context tree has about 7 leaf nodes. The context alphabet has three symbols, while the coding distribution has two symbols (symbol 2 is ignored). 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.

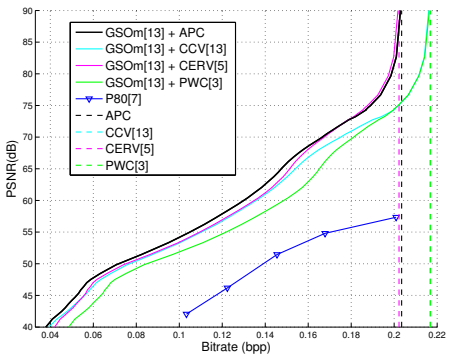


(b) *Breakdancers* sequence.

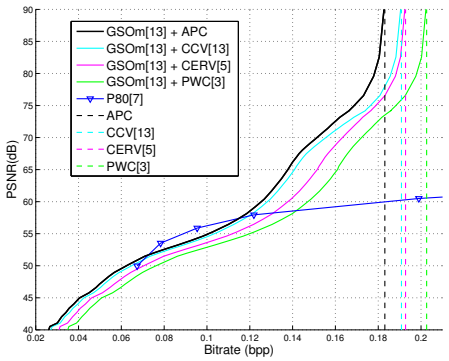


(c) *Middlebury* dataset.

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.

Publication 6

Copyright ©2015 IEEE. Reprinted, with permission, from

[P6] I. Schiopu and I. Tabus. Lossy-to-lossless progressive coding of depth-maps. In *Proc. International Symposium on Signals, Circuits and Systems (ISSCS)*, pages 1–4, Iasi, Romania, July 2015.

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Tampere University of Technology's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

Lossy-to-lossless progressive coding of depth-maps

Ionut Schiopu, Ioan Tabus
Tampere University of Technology, Department of Signal Processing,
P.O.Box 553, FIN-33101, Tampere, Finland

***Abstract*— A progressive coding method is proposed for depth-map images, where the bitstream is encoded so that one can generate many lossy versions of the original, encompassing a wide range, from very low resolution up to lossless reconstruction. The partitions into regions of the lossy versions are assumed to be nested, so that a higher resolution image is obtained by splitting some regions of a lower resolution image. The encoder transmits to the decoder information about which regions to split, the extra contour to be added for obtaining the shapes of the more refined regions, and the extra depth values needed inside each new region. The efficient encoding of the anchor points in the progressive scenario, relative to the contour points already encoded, and the depth information recovery, are the main contributions of this paper. The progressive bitstream produced by the proposed method scales well over the whole range of rates, from low rates to lossless, reaching a performance close to that of the non-progressive methods.**

I. Introduction

The progressivity in depth-map coding is an important functionality, allowing the encoder to broadcast the same bitstream of a depth-map towards many decoders, each having different rate-distortion requirements. Progressive depth-map coding was proposed in [1], where reversible cellular automata are used to achieve spatial scalability, and in [2], where geometry information is conveyed by prioritized breakpoints found by a breakpoint-adaptive transform. The recent algorithms for depth-map compression are based on various methods, e.g. bit plane compression [3] and context coding of contours [4].

In this paper the depth-map images are described using the terminology from [4], [5], [6], the notations being in particular similar to [6], illustrated here in Figure 1 and briefly described below. A depth-map image is an $n_r \times n_c$ matrix Z that contains the depth value $Z(i,j)$ for each pixel position (i,j) in the pixel grid. The image Z is represented using a set of n_Ω connected regions $\{\Omega_1, \Omega_2, \ldots, \Omega_{n_\Omega}\}$, each region Ω_ℓ contains pixel(s) having the (same) depth value, d_ℓ. The contours between neighbor regions can be described in an $(n_r+1)\times(n_c+1)$ *grid of vertices*. Two neighbor pixels are separated by a contour-edge, which unites two vertices in the vertex grid. If the pixels have different depth values, the contour-edge between the two pixels is active, e.g. if $Z(i,j) \neq Z(i,j+1)$ then there is an active *contour edge* between the adjacent vertices $P_k = (i, j+1)$ and $P_\ell = (i+1, j+1)$ from the vertex grid.

The contours between regions can be described by sequences of either active contour edges or adjacent vertices. A sequence of n_{Γ_k} consecutively adjacent vertices forms a *contour segment*, denoted $\Gamma_k = [P_1 \ P_2 \ \cdots \ P_{n_{\Gamma_k}}]^T$. All contours of Z are represented by the set of contour segments $\{\Gamma_k\}_{k=1,2,\ldots,n_\Gamma}$. Each Γ_k is represented efficiently using the 3OT representation [4], by describing the advancing on the contour segment from one vertex to one of its adjacent vertices using the symbols: *'0'* if the previous and next contour-edges are on the same straight line; *'1'* if the direction of movement is the same as previously but the orientation (horizontal/vertical) changes; *'2'* if both direction of movement and orientation are changing. Figure 1.(c) shows a sequence of adjacent vertices and their associated 3OT symbols. Hence, Γ_k is codified by an anchor point, P_1, a direction point, P_2, and the vector $S_k = [s_1 \ s_2 \ \cdots \ s_{n_{\Gamma_k}-2}]^T$ of 3OT symbols.

Here we introduce a method for building a progressive bitstream, where the descriptions of the regions are embedded in such a way that the most important regions are described first, in the lowest rate version, and then information is added for splitting progressively the large regions into finer regions, until getting the full detail representation. This hierarchical segmentation and generation of a sequence of lossy images is obtained by the Greedy rate-distortion Slope Optimization algorithm with region merging (GSOm) [5].

Our goal here is to introduce the progressive functionality over a wide range of rates, without degrading too much the performance comparing to non-progressive methods. Section II introduces the method, Section III the contour compression, and Section IV the depth values compression. We discuss the results in Section V, and draw conclusions in Section VI.

II. Progressive coding of GSO sequences (P-GSO)

We start by briefly describing the GSOm algorithm introduced in [5], that is generating the nested lossy versions $\mathcal{Z}^{np} = \{Z_1, Z_2, \ldots, Z_n\}$ of a depth map image Z_0. Image $Z_{\tau+1}$ is generated from Z_τ by merging several regions of Z_τ, where the next pair of regions to be merged is selected as that which will produce an optimal slope in the rate-distortion plane. The number of regions, which are merged to transform Z_τ into $Z_{\tau+1}$, is set by the user in order to get a certain SNR difference. Each image Z_τ is then encoded using a specialized depth-map entropy coder (e.g. CERV[4], APC[6]) in $\mathcal{L}^{np}(Z_\tau)$ bits, resulting in a non-progressive coding of $\mathcal{Z}^{np}$.

The main contribution of this paper consists of a progressive method, dubbed Progressive coding of GSO sequences (P-GSO), that creates a bitstream which can be truncated to generate the sequence $\mathcal{Z}^{np}$ in the reversed order, $\mathcal{Z}^{p} = \{Z_n, Z_{n-1}, \ldots, Z_0\}$, or any subsequence of it. In non-progressive coding, for each image Z_τ the encoding is performed independently. In the case of progressive coding, Z_τ is created from the prefixes of the bitstream of images from Z_n to $Z_{\tau+1}$, which makes the progressive coding of Z_τ

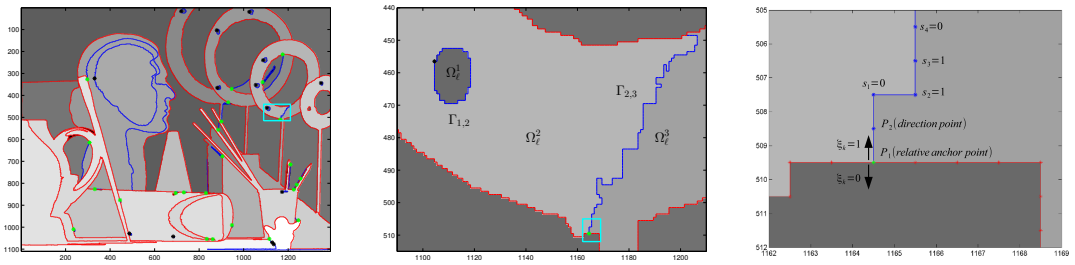


Fig. 1. Graphical representation of entities used at step τ to compress the contour of the image Z_τ. The known contour map of $Z_{\tau+1}$ is represented by contour edges and vertices marked with red. Encoded contour segments are represented by contour edges and vertices marked with blue. Double anchor points are marked with black and relative anchor points are marked with green. From (a) to (c) we zoom in the cyan rectangles. **(a)** Overlay of the contour maps of $Z_{\tau+1}$ and Z_τ over the depth map of the lossy image Z_τ, for the image *Art* (full size, disp1). **(b)** The contour edges from the zoomed rectangle in (a). The contour segments $\Gamma_{1,2}$ and $\Gamma_{2,3}$ are separating three new regions $\Omega_\ell^1, \Omega_\ell^2$, and Ω_ℓ^3 from the initial region Ω_ℓ. The contour segment $\Gamma_{1,2}$ is encoded using a double anchor point and a vector of 166 $3OT$ symbols; the contour segment $\Gamma_{2,3}$ is encoded using a relative anchor point, a direction point and a vector of 122 $3OT$ symbols. **(c)** The contour edges and the vertices in the zoomed rectangle from (b). As an example of a binary switch, ξ_k has two possible values. For $\Gamma_{2,3}$, $\xi_k = 1$ points to the direction point P_2. Four out of the 122 $3OT$ symbols, which are encoding the contour segment $\Gamma_{2,3}$, are shown.

less efficient than the non-progressive one. Hence, the rate-distortion curve obtained by the progressive encoding is below the non-progressive rate-distortion curve, since the progressive codelength $\mathcal{L}^p(Z_\tau)$ is larger than the non-progressive codelength $\mathcal{L}^{np}(Z_\tau)$. Our goal is to design the encoding process between each two consecutive images, $Z_{\tau+1}$ and Z_τ, so that the penalty paid for progressivity, $\mathcal{L}^p(Z_\tau) - \mathcal{L}^{np}(Z_\tau)$, is as small as possible.

P-GSO is generating each Z_τ using a priori information from $Z_{\tau+1}$, and by encoding auxiliary information. On one hand, in GSOm, at each step τ from 1 to n, the set $\Delta\Gamma^{(\tau)} = \{\Gamma_k\}_{k=1,2,\ldots,n_{\Delta\Gamma}^{(\tau)}}$ of $n_{\Delta\Gamma}^{(\tau)}$ contours is removed from Z_τ to obtain $Z_{\tau+1}$ (having less regions), and new depth values are computed, in each of the newly formed regions, as the truncated average of original depth values over the region. On the other hand, here in P-GSO, at each step τ from $n-1$ to 0, we do the opposite: $n_{\Delta\Gamma}^{(\tau)}$ contours, $\Delta\Gamma^{(\tau)}$, are added to the contours of $Z_{\tau+1}$, in order to obtain the contours of Z_τ, and the truncated averages are added to the newly formed regions, so that at the end of step τ we reconstruct Z_τ. From a few regions (in the order of tens) present in Z_n, P-GSO is splitting iteratively the regions to obtain typically several thousand regions, the lossy versions selected in the sequence $\mathcal{Z}^p$ being snapshots at intermediate stages in the split process. At one iteration, P-GSO performs tens of splits, while in the near-lossless part the number is in the order of thousands splits. In the next sections we describe the progressive coding, at step τ, of the contours and of the depth values.

III. Contour compression

The GSOm algorithm eliminates less important contours from the contours of Z_τ, to obtain the contours of $Z_{\tau+1}$. In the end, Z_n contains only the most important contours from Z_0. Hence GSOm is classifying in $\mathcal{Z}^{np}$ the contours of Z_0 in n levels of importance: $\Delta\Gamma^{(1)}, \Delta\Gamma^{(2)}, \ldots, \Delta\Gamma^{(n)}$.

The P-GSO algorithm is doing the reverse, at each iteration is adding to the already encoded contours $\Gamma^{(\tau+1)}$ of the lossy version $Z_{\tau+1}$, the contour segments on the next lower level of importance, $\Delta\Gamma^{(\tau)}$, to obtain the contours of Z_τ, i.e. $\Gamma^{(\tau)} = \Delta\Gamma^{(\tau)} \cup \Gamma^{(\tau+1)}$. Each Γ_k belonging to $\Delta\Gamma^{(\tau)}$ is encoded by an anchor point, a direction point, and a chain-code vector S_k.

A. Anchor points representation

The encoding of anchor points was studied and an efficient non-progressive solution, for searching and traversing the contour segments, is given by APC in [6]. Same anchor points classification is used also here: *double*, *relative* (see Figure 1.(b)), and *edge* (located on the border of the image). The connection to [6] ends here, all subsequent algorithmic steps being different and specific to the goal of *progressive* coding. The contour segments $\Gamma^{(\tau+1)}$ are used at step τ as a priori information in the anchor points search, and hence the implemented search procedures are different than in [6].

The first type of search finds the so called relative anchor points, by traversing the contour segments $\Gamma^{(\tau+1)}$ and marking in a binary vector $\Psi^{(\tau)}$ the vertices that have a different number of adjacent vertices in $\Gamma^{(\tau)}$. The found vertices have minimum two adjacent vertices part of $\Gamma^{(\tau+1)}$, and maximum two adjacent vertices part of $\Gamma^{(\tau)}$, representing the possible direction points. If a found vertex, say P_k, has: **(i)** four adjacent vertices part of $\Gamma^{(\tau+1)}$, no contour can start form P_k in $\Gamma^{(\tau)}$; **(ii)** three adjacent vertices part of $\Gamma^{(\tau+1)}$, the direction point is the remaining vertex (rare case); **(iii)** two adjacent vertices part of $\Gamma^{(\tau+1)}$, the position of each direction point, relative to the anchor point $P_1 = P_k$, is saved using a binary switch, ξ_k (see Figure 1.(c)), that is appended to a vector $\Xi^{(\tau)}$. At the end of the search, $n_\xi^{(\tau)}$ relative anchor points and $n_\xi^{(\tau)}$ direction points are found in $\Delta\Gamma^{(\tau)}$.

The second type of search finds the edge and double anchor points by a column-wise search, and marks them in a matrix

$\Upsilon^{(\tau)}$ by checking column-wise the contour graph of Z_τ. Here there is no need to encode a direction point because: **(a)** an edge anchor point is a vertex with one adjacent vertex, and it is found at the image border; **(b)** a double anchor point has two adjacent vertices part of $\Gamma^{(\tau)}$, but from it can either start a contour segment which forms a loop inside a larger region, or start two contour segments (reason why it is called double).

The contour segment Γ_k is described starting with the anchor point, continuing with the direction point, and then the vertices described by the vector S_k of 3OT symbols, and it is ending with a vertex that has one of the following proprieties: *(a)* is on the edge of the image; *(b)* has at least three adjacent vertices in the contour of $Z_{\tau+1}$; *(c)* has at least three visited adjacent vertices in the contour of Z_τ. Hence, the decoder is able to detect the contour's end, without knowing its length.

B. Entropy coding the contour segments

In this paper two encoding functions are introduced:

1) Integer Value Encoder, IVE(v, nb_δ, nb_r): The IVE function is introduced to encode an integer $v \geq 0$ by first selecting the range in which it belongs, and encoding it using the found range limits. A switch is used to select one out of four possible ranges, where the first range is $[0,\ 2^{nb_\delta}]$, while the following are shifted by increasing with nb_r the number of bits needed to represent the upper bound of previous range.

2) Golomb-Rice encoding of vectors, GR(Φ): A vector Φ is encoded depending on its number of elements, n_Φ, as follows. If $n_\Phi \leq 50$, then Φ is encoded using the Golomb-Rice (GR) algorithm with $M = 2^{k^*}$. Else an encoded switch specifies to divide or not Φ into two sub-vectors, denoted Φ_L and Φ_H. If the switch is on, for each element in Φ the decision to add it to Φ_L or Φ_H is encoded and the Golomb-Rice algorithm is applied to Φ_L and Φ_H, else the GR algorithm is applied to Φ.

The contour segments from $\Delta\Gamma^{(\tau)}$ are encoded by: **(a)** the vectors $\Psi^{(\tau)}$ (of relative anchor points) and $\mathrm{vec}(\Upsilon^{(\tau)})$ (of edge and double anchor points), each encoded by first obtaining a vector Φ which saves the number of consecutive values 0, and using *GR*(Φ); **(b)** two integer values $v_1 = n_\xi^{(\tau)}$ (number of relative anchor points), and $v_2 = n_{\Delta\Gamma}^{(\tau)} - n_\xi^{(\tau)}$ (number of edge and double anchor points), each encoded by *IVE*$(v_i, 4, 2)$; **(c)** the vector of direction points, $\Xi^{(\tau)}$, written directly in the output file; **(d)** the vector of concatenated 3OT vectors, $S^{(\tau)} = \left[S_1^T\ S_2^T\ \cdots\ S_{n_{\Delta\Gamma}^{(\tau)}}^T\right]^T$, encoded using the Context Tree Algorithm [7] with a maximum tree depth of $d_\mathcal{T} = 17$.

IV. Depth value compression

Here the depth of any region Ω_ℓ^j is called the truncated average value of the true depth values (from Z_0) at all pixels of the regions, and we denote it by d_ℓ^j, carrying the same sub- or super- indexes as the region symbolic name, Ω_ℓ^j.

Let us consider that a region from $Z_{\tau+1}$, say Ω_ℓ, with the truncated average d_ℓ, is split into a set of n_ℓ regions, $\{\Omega_\ell^j\}_{j=1,2,\ldots,n_\ell}$ in Z_τ. All regions in $Z_{\tau+1}$ and Z_τ are defined by the GSOm algorithm as follow: **(a)** each region Ω_ℓ^j is formed of m_ℓ^j pixels, $\Omega_\ell^j = \{(x_i, y_i)\}_{i=1,2,\ldots,m_\ell^j}$, having the depth values $z_i = Z_0(x_i, y_i)$ in the original depth-map image Z_0; **(b)** the real-valued average inside Ω_ℓ^j is $\breve{d}_\ell^j = \frac{1}{m_\ell^j}\sum_{i=1}^{m_\ell^j} z_i$, and the truncated average is $d_\ell^j = \left\lfloor \breve{d}_\ell^j \right\rfloor = \breve{d}_\ell^j - \Delta_\ell^j$, where $\Delta_\ell^j \in [-0.5,\ 0.5]$ is the rounding error. The region Ω_ℓ is defined similarly as having: m_ℓ pixels, the real-value average $\breve{d}_\ell = \frac{1}{m_\ell}\sum_{i=1}^{m_\ell} z_i$ and truncated average $d_\ell = \left\lfloor \breve{d}_\ell \right\rfloor = \breve{d}_\ell - \Delta_\ell$, where $\Delta_\ell \in [-0.5,\ 0.5]$ is the rounding error. Since Ω_ℓ is split into $\{\Omega_\ell^j\}_{j=1,2,\ldots,n_\ell}$, then $m_\ell = \sum_{j=1}^{n_\ell} m_\ell^j$.

The decoder has available from $Z_{\tau+1}$ all information describing the region Ω_ℓ, and from Z_τ (after decoding the contours of Z_τ) all the pixel locations and number of pixels for each region $\{\Omega_\ell^j\}_{j=1,2,\ldots,n_\ell}$. To obtain Z_τ, it only needs to receive the set of truncated averages $\{d_\ell^j\}_{j=1,2,\ldots,n_\ell}$.

In this paper we assume that the values d_ℓ^j are scattered around the truncated average d_ℓ. This assumption is used for the first $n_\ell - 1$ truncated averages, $\{d_\ell^j\}_{j=1,2,\ldots,n_\ell-1}$, that are transmitted to the decoder using the differences

$$b_j = d_\ell - d_\ell^j, j = 1, 2, \ldots, n_\ell - 1. \tag{1}$$

The values b_j, from all the split operations that transform $Z_{\tau+1}$ into Z_τ, are appended in the vector $B = \left[b_1\ b_2\ \cdots\ b_{n_B^{(\tau)}}\right]^T$, where $n_B^{(\tau)}$ is the difference between the number of regions in Z_τ and $Z_{\tau+1}$. We encode each b_j by its sign, using one bit, and by its absolute value, $|b_j|$, using the adaptive zero-order model with Laplace estimator, with ns_B symbols. To set ns_B we first find the maximum absolute difference $b_M = \max_{j=1,2,\ldots,n_B^{(\tau)}} |b_j|$, compute the number of bits needed to represent it, $n_M = \lceil log_2(b_M) \rceil$, and then encode n_M using $\log_2(7)$ bits. Both encoder an decoder can now set $ns_B = 2^{n_M}$. However, sometimes encoding additional bits to be able to set $ns_B = b_M$, is improving the result.

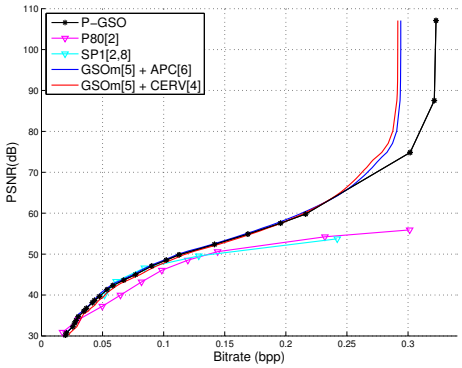
The last truncated average, $d_\ell^{n_\ell}$, is encoded using the second redundancy: the sum of initial depth values of the pixels in Ω_ℓ is equal to the sum of initial depth values of the pixels in $\{\Omega_\ell^j\}_{j=1,2,\ldots,n_\ell}$. This is written using real-value averages as $\left(\sum_{j=1}^{n_\ell} m_\ell^j\right)\breve{d}_\ell = \sum_{j=1}^{n_\ell} m_\ell^j \breve{d}_\ell^j$, or using truncated averages as

$$\left(\sum_{j=1}^{n_\ell} m_\ell^j\right) d_\ell + \left(\sum_{j=1}^{n_\ell} m_\ell^j\right) \Delta_\ell = \sum_{j=1}^{n_\ell} m_\ell^j d_\ell^j + \sum_{j=1}^{n_\ell} m_\ell^j \Delta_\ell^j. \tag{2}$$

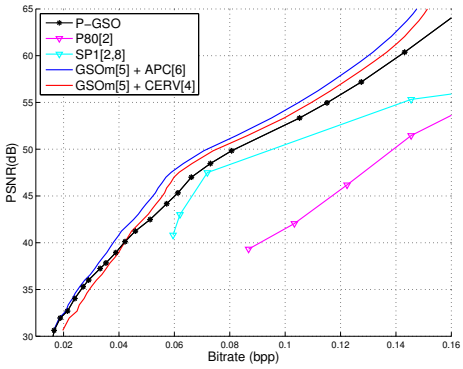
We now rewrite (2) for the selected left term, $d_\ell^{n_\ell}$, as

$$d_\ell^{n_\ell} = \underbrace{d_\ell + \sum_{j=1}^{n_\ell-1} \frac{m_\ell^j}{m_\ell^{n_\ell}}\left(d_\ell - d_\ell^j\right)}_{\hat{d}_\ell^{n_\ell}} + \underbrace{\sum_{j=1}^{n_\ell} \frac{m_\ell^j}{m_\ell^{n_\ell}}\left(\Delta_\ell - \Delta_\ell^j\right)}_{\varepsilon_\ell^{n_\ell}}, \tag{3}$$

where $\hat{d}_\ell^{n_\ell}$ is the estimation of the truncated average $d_\ell^{n_\ell}$ that the decoder can compute since $\{d_\ell^j\}_{j=1,2,\ldots,n_\ell-1}$ and d_ℓ are already available, and $\varepsilon_\ell^{n_\ell}$ is the rounding residual that depends on the coefficients $\left\{\frac{m_\ell^j}{m_\ell^{n_\ell}}\right\}_{j=1,2,\ldots,n_\ell-1}$. We minimize $\varepsilon_\ell^{n_\ell}$ by selecting $\Omega_\ell^{n_\ell}$ as the region with the highest number of pixels



(a)



(b)

Fig. 2. Comparative results for the progressive methods: P80 [2], SP1 [2], [8], and our method, P-GSO, with the non-progressive methods: CERV [4], APC [6], for the images **(a)** *Ballet* (cam0, f96) from lossy-to-lossless; **(b)** *Aloe* (full size, disp1) zoomed in the PSNR range $[30,\ 65]$ dB.

among all Ω_ℓ^j, so that $\frac{m_\ell^j}{m_\ell^{n_\ell}} < 1, \forall j = 1, 2, \ldots, n_\ell - 1$. Finally, we round $\hat{d}_\ell^{n_\ell}$ as $\hat{d}_\ell^{n_\ell} = \left\lfloor \hat{d}_\ell^{n_\ell} \right\rceil - \hat{\Delta}_\ell^{n_\ell}$, where $\hat{\Delta}_\ell^{n_\ell}$ is the rounding error, and encode $d_\ell^{n_\ell}$ using the prediction residual

$$a_\ell = d_\ell^{n_\ell} - \left\lfloor \hat{d}_\ell^{n_\ell} \right\rceil = \varepsilon_\ell^{n_\ell} + \hat{\Delta}_\ell^{n_\ell}. \tag{4}$$

The values a_ℓ are appended to a vector A. At iteration τ we have $A = \left[a_1\ a_2\ \cdots\ a_{n_A^{(\tau)}}\right]^T$, where $n_A^{(\tau)}$ is the number of regions split in $Z_{\tau+1}$. The tests showed that $a_\ell \in [-2,\ 2]$, and hence the vector A is described using a vector of possible symbols, $\Lambda = [-1\ 0\ 1\ -2\ 2]^T = [\lambda_1\ \lambda_2\ \lambda_3\ \lambda_4\ \lambda_5]^T$, by finding for each a_ℓ the index k_ℓ for which $\lambda_{k_\ell} = a_\ell$. Next we compute $k_M = \max_{\ell=1,2,\ldots,n_A^{(\tau)}} k_\ell$, and then encode it using $\log_2(5)$ bits. Finally we encode each k_ℓ using the adaptive zero-order model with Laplace estimator with k_M symbols.

V. Experimental results

We experiment with 180 images from three datasets: *Breakdancers* and *Ballet* [9], and *Middlebury* [10], out of which Figure 2 presents two images (see the additional file[1]).

The new method, P-GSO, selects from $\mathcal{Z}^{np}$ a subsequence $\mathcal{Z}^p$, which contains the lossy images having the distortion the closest to some selected round values (e.g., $30, 40, 50, 60$ dB). In Figure 2 we compare P-GSO with the progressive method "Proposed_80" from [2], called here P80, and the non-progressive method "Segment_Param_1" described in [2], [8] and called here SP1; and two non-progressive methods CERV [4] and APC [6], that are used together with GSOm [5]. For P-GSO, the first image in the sequence is encoded using APC.

One can see that the proposed method scales well over the whole range of bitrates, from low bitrates until the lossless bitrate. The comparison with P80, which is the only progressive method previously reported in the literature, shows the significant improvements obtained with our method (with gains from several dB to more than ten dB in the nearlossless regime). When compared to the non-progressive methods, our progressive method succeeds in having a low penalty for the progressiveness. However due to encoding of much more intermediate parameters in the embedded stream, the progressive scheme experience the inevitable losses with respect to the non-progressive counterpart, with more notable drop in PSNR towards the near-lossless range.

VI. Conclusions

The new method, P-GSO, compares favorably with the other existing progressive coding methods. The experimental result showed that the subsequences of GSO lossy versions are encoded form lossy to lossless reconstruction, reaching a performance close to that of the non-progressive methods.

Acknowledgment

Ionut Schiopu was supported by a Nokia Scholarship grant from the Foundation of Nokia Corporation.

References

[1] L. Cappellari, C. Cruz-Reyes, G. Calvagno, and J. Kari, "Lossy to lossless spatially scalable depth map coding with cellular automata," in *Proc. DCC*, Mar. 2009, pp. 332–341.

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

[3] K. Kim, G. Park, and D. Suh, "Bit-plane-based lossless depth-map coding," *Optical Engineering*, vol. 49, no. 6, pp. 067 403–1–10, 2010.

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

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

[6] ——, "Anchor points coding for depth map compression," in *Proc. IEEE ICIP*, Paris, France, Oct. 2014, pp. 5626 – 5630.

[7] J. Rissanen, "A universal data compression system," *IEEE Trans. Inform. Theory*, vol. 29, pp. 656–664, Sep. 1983.

[8] P. Zanuttigh and G. Cortelazzo, "Compression of depth information for 3D rendering," in *Proc. 3DTV-CON*, May 2009, pp. 1–4.

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

[10] H. Hirschmuller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *Proc. CVPR*, Minneapolis, MN, 2007, pp. 1–8.

[1] Additional results at www.cs.tut.fi/~schiopu/PGSO/PGSOresults.pdf

Publication 7

Copyright ©2015 IEEE. Reprinted, with permission, from

[P7] I. Schiopu and I. Tabus. Parametrizations of planar models for region-merging based lossy depth-map compression. In *Proc. 3DTV Conference: Immersive and interactive 3D Media experience over networks (3DTV-CON)*, pages 1–4, Lisbon, Portugal, July 2015.

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Tampere University of Technology's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

PARAMETRIZATIONS OF PLANAR MODELS FOR REGION-MERGING BASED LOSSY DEPTH-MAP COMPRESSION

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

We present efficient methods for parametrizing planar models, to be used for depth value reconstruction inside selected regions in a depth image. The optimal plane for each region is represented using its quantized heights at three pixel locations. The decoder uses the decoded quantized heights to approximately represent the optimal plane. The three pixel locations are selected so that the distortion due to the approximation of the plane over the region is minimized. The planar reconstructions are used in competition with the piecewise constant reconstruction at the regions obtained through a merging process, where the two regions to be merged are those ensuring the optimal slope in the rate-distortion curve. The lossy depth compression algorithm including the planar modeling obtains a significantly better rate-distortion performance than the algorithm that uses only constant regions, with improvements up to 8 dB.

Index Terms — Planar model, plane parametrization, entropy coding, greedy slope optimization

1. INTRODUCTION

Depth maps compression was intensively studied using various approaches for lossy and lossless reconstruction such as: bit plane compression [1], context coding of contours [2], quad-tree decompression and platelets [3], combining texture and depth compression [4]. Planar models were used for depth-map image compression in [5], where the segmentation process takes place in the virtual 3D point cloud. In [5], the planar models are only a few (less than ten), and they are obtained by using a graph-cut co-segmentation algorithm.

A depth-map image is represented by an $n_r \times n_c$ matrix Z that stores the depth value $Z(x, y)$ for each pixel position (x, y) in the pixel grid. The image Z is partitioned into n_Ω regions $\Omega_1, \Omega_2, \ldots, \Omega_{n_\Omega}$. Each region Ω_k is either a single pixel, or is a *connected* region.

We build here on the footsteps of the GSOm algorithm [6], which generates the sequence of lossy versions $Z_1, Z_2, \ldots, Z_n$ for the original depth-map Z_0. The image $Z_{\tau+1}$ is generated from Z_τ, by merging several regions, where the next pair of regions to be merged is selected as that which will produce the optimal slope in the rate-distortion plane. In [6] the piecewise constant model was used for depth reconstruction so that the changes in rate-distortion, after merging two regions, are easy to compute.

In here we allow a planar model additionally to the piecewise constant model and decide which type of model to be used by the greedy slope criterion. In contrast to the very few planar models used in [5], here the planar models obtain very detailed descriptions, reaching thousands of small planar regions in the near lossless regime. The importance of properly choosing the parametrization of the plane, and of quantizing its parameters with the right amount of bits, is now crucial.

The parametrization of the planar model, the parameter quantization and encoding are discussed in Section 2. The GSOm algorithm [6] is extended to include the planar model option in the greedy decision process for image generation, as presented in Section 2.4. The experimental results are presented in Section 3 and the conclusions are drawn in Section 4.

2. PLANAR MODEL FOR DEPTH ESTIMATION

2.1. Planar model parameterization and estimation

The most used parametric form of the planar model is $z = ax + by + c$. Alternatively, we also consider the equivalent parametrization provided by specifying three points of the plane, and we describe in the sequel the estimation, quantization and entropy coding for the two parametrizations.

Let us consider a region $\Omega_\ell = \{(x_i, y_i)\}_{i=1,2,\ldots,n}$ formed of n pixels, having the true depth values denoted $z_i = Z_0(x_i, y_i)$, $i = 1, 2, \ldots, n$. In order to minimize the MSE over the region, the Least Squares (LS) solution, that is minimizing the sum of squared modeling errors $\sum_{i=1}^{n} \varepsilon_i^2$, is computed, where the errors ε_i are defined by

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ & \vdots & \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}, \tag{1}$$

resulting in the LS parameters, $\underline{\theta}^* = \begin{bmatrix} a^* & b^* & c^* \end{bmatrix}^T$. If the $\underline{\theta}^*$ parameters would be encoded, then the decoder would use $\underline{\theta}^*$ to obtain first the floating point value z_i^*:

$$z_i^* = \begin{bmatrix} x_i & y_i & 1 \end{bmatrix} \underline{\theta}^* = z_i - \varepsilon_i^*, \; i = 1, 2, \ldots, n. \tag{2}$$

where ε_i^* are the optimal modeling errors. Then, in the reconstructed image at the pixel location (x_i, y_i), the value $\hat{z}_i$ is computed by rounding z_i^*:

$$\hat{z}_i = \lfloor z_i^* \rceil = z_i^* - \Delta_i^*, \; i = 1, 2, \ldots, n, \tag{3}$$

where $\Delta_i^* \in [-0.5, \; 0.5]$ are rounding errors. From (2) and (3) it results that z_i would be reconstructed as $\hat{z}_i = z_i - (\varepsilon_i^* + \Delta_i^*)$. The distortion obtained by the optimal model $\underline{\theta}^*$ over Ω_ℓ is:

$$MSE_1 = \frac{1}{n} \sum_{i=1}^{n} (z_i - \hat{z}_i)^2 = \frac{1}{n} \sum_{i=1}^{n} (\varepsilon_i^* + \Delta_i^*)^2. \tag{4}$$

However, the $\underline{\theta}^*$ parameters are not transmitted to the decoder, but instead the heights of the plane at three selected pixels are

encoded, after truncating them to integers. Let us denote the three pixel locations $A = (x_\alpha, y_\alpha), B = (x_\beta, y_\beta)$, and $C = (x_\gamma, y_\gamma)$, and their values in Z_0 as z_α, z_β, z_γ. The selection of the points A, B, C for a given region Ω_ℓ is done in the same way at encoder and decoder, according to the methods described in Section 2.2.

The heights of the optimal plane at these three points are the real numbers z^*_α, z^*_β, z^*_γ (given by (2) for $i = \alpha, \beta, \gamma$), and their rounded values $\hat{z}_\alpha$, $\hat{z}_\beta$, $\hat{z}_\gamma$ are encoded and sent to the decoder. They are represented using (3) for $i = \alpha, \beta, \gamma$ as:

$$\begin{bmatrix} \hat{z}_\alpha \\ \hat{z}_\beta \\ \hat{z}_\gamma \end{bmatrix} = \begin{bmatrix} z^*_\alpha - \Delta^*_\alpha \\ z^*_\beta - \Delta^*_\beta \\ z^*_\gamma - \Delta^*_\gamma \end{bmatrix} = \underbrace{\begin{bmatrix} x_\alpha & y_\alpha & 1 \\ x_\beta & y_\beta & 1 \\ x_\gamma & y_\gamma & 1 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} a^* \\ b^* \\ c^* \end{bmatrix}}_{\underline{\theta}^*} - \underbrace{\begin{bmatrix} \Delta^*_\alpha \\ \Delta^*_\beta \\ \Delta^*_\gamma \end{bmatrix}}_{\underline{\Delta}} \quad (5)$$

where the matrix Q stacks the coordinates of the selected locations A, B, C that are defining a triangle $\triangle ABC$.

The decoder determines the three pixel locations A, B, C for the region Ω_ℓ, as in Section 2.2, and uses $\hat{z}_\alpha$, $\hat{z}_\beta$, $\hat{z}_\gamma$ to compute the reconstruction plane, $\tilde{\underline{\theta}}$, by solving:

$$\underbrace{\begin{bmatrix} x_\alpha & y_\alpha & 1 \\ x_\beta & y_\beta & 1 \\ x_\gamma & y_\gamma & 1 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} \tilde{a} \\ \tilde{b} \\ \tilde{c} \end{bmatrix}}_{\tilde{\underline{\theta}}} = Q\tilde{\underline{\theta}} = \begin{bmatrix} \hat{z}_\alpha \\ \hat{z}_\beta \\ \hat{z}_\gamma \end{bmatrix}. \quad (6)$$

The decoder is setting for each pixel location (x_i, y_i), in the reconstructed image, the reconstructed value, $\breve{z}_i$, computed as the corresponding rounded height, $\tilde{z}_i$, in the $\tilde{\underline{\theta}}$ plane:

$$\breve{z}_i = \lfloor \tilde{z}_i \rceil = \tilde{z}_i - \tilde{\Delta}_i = [x_i \; y_i \; 1]\tilde{\underline{\theta}} - \tilde{\Delta}_i, i = 1, 2, \ldots n, \quad (7)$$

where $\tilde{\Delta}_i \in [-0.5, \; 0.5]$ are rounding errors. Finally, the distortion in region Ω_ℓ, computed using the planar model, is

$$MSE_p = \frac{1}{n} \sum_{i=1}^{n} (z_i - \breve{z}_i)^2, \quad (8)$$

while the distortion computed using the constant model is $MSE_c = \frac{1}{n} \sum_{i=1}^{n} (z_i - \lfloor d_\ell \rceil)^2$, where $d_\ell = \frac{1}{n} \sum_{i=1}^{n} z_i$ [6] .

2.2. Selecting A, B, C for MSE optimization

The selection of the points A, B, C using various methods, as described below, will impact on both the distortion over the region $D_\ell = MSE_p$ and the rate R_ℓ. The distortion is reduced if the approximation of the plane $\underline{\theta}^*$ by the plane $\tilde{\underline{\theta}}$ becomes better. Heuristically, the rounding error made at the points A, B, C in (5) will change very little the ideal plane $\underline{\theta}^*$ if A, B, C will be as far apart one from another as possible. However, that will make the values z^*_α, z^*_β, z^*_γ very large in magnitude, and thus very costly to encode. As a main trade-off, we differentiate here between methods that are constraining A, B, C to be inside the region Ω_ℓ (method M_1, which minimizes an expected MSE criterion, and method M_2, which maximizes the area of the triangle ABC) and methods allowing A, B, C to lie outside the region Ω_ℓ, but only exceeding moderately the region (methods M_3 to M_6 below).

2.2.1. Methods searching for A, B, C inside Ω_ℓ

Selecting the points A, B and C of the triangle $\triangle ABC$ is equivalent to setting the elements of the matrix Q in (6), emphasized for clarity where needed by denoting Q_{ABC}.

($\mathbf{M_1}$) In this method first we consider the MSE excess, $MSE_p - MSE_1$, second we isolate in it one component, denoted MSE_e, and finally define its expected value to be the criterion of interest. We evaluate MSE_p using the steps: **(i)** from (5) and (6) obtain:

$$Q\underline{\theta}^* = Q\tilde{\underline{\theta}} + \underline{\Delta} \Longleftrightarrow \underline{\theta}^* - \tilde{\underline{\theta}} = Q^{-1}\underline{\Delta}; \quad (9)$$

(ii) rewrite $z_i - \breve{z}_i$ using (2), (3), and (7) as:

$$\begin{aligned} z_i - \breve{z}_i &= (z_i - \hat{z}_i) + (\tilde{\Delta}_i - \Delta^*_i) + [x_i \; y_i \; 1](\underline{\theta}^* - \tilde{\underline{\theta}}) \\ &\overset{(9)}{=} (z_i - \hat{z}_i) + (\tilde{\Delta}_i - \Delta^*_i) + [x_i \; y_i \; 1]Q^{-1}\underline{\Delta}; \end{aligned}$$

(iii) rewrite (8) using the square of $z_i - \breve{z}_i$ and neglecting the cross-terms (by assuming negligible cross-correlations between modeling errors and rounding errors):

$$MSE_p \approx MSE_1 + \frac{1}{n} \sum_{i=1}^{n} (\tilde{\Delta}_i - \Delta^*_i)^2 + \underbrace{\underline{\Delta}^T Q^{-T} R Q^{-1} \underline{\Delta}}_{MSE_e},$$

where $R = \frac{1}{n} \sum_{i=1}^{n} [x_i \; y_i \; 1]^T [x_i \; y_i \; 1]$. We now concentrate on the term denoted MSE_e and define its expected value as a minimization criterion. If we assume that the rounding errors making up the vector $\underline{\Delta}$ are behaving like independent variables with distribution $\mathcal{U}(-0.5, 0.5)$ when the LS plane is taking all possible positions, then the expected MSE_e is:

$$\mathrm{E}[MSE_e] = \mathrm{E}\left[\underline{\Delta}^T Q^{-T} R Q^{-1} \underline{\Delta}\right] = \frac{1}{12} \mathrm{tr} Q^{-T} R Q^{-1}. \quad (10)$$

Hence, this method is looking for the locations A, B, C as the solution of the following problem

$$\min_{A,B,C \in \Omega_\ell} \mathrm{tr} Q^{-T}_{ABC} R Q^{-1}_{ABC}. \quad (11)$$

($\mathbf{M_2}$) The second method is based on the heuristic idea that the $\triangle ABC$ must cover as much as possible from Ω_ℓ (so that A, B, C are as far as possible from one another inside the region Ω_ℓ). This is achieved when $\triangle ABC$ has maximum area. This area is given by $|\det Q_{ABC}|$. Hence, this method is looking for the locations A, B, C solving

$$\max_{A,B,C \in \Omega_\ell} Area(\triangle ABC) = \max_{A,B,C \in \Omega_\ell} |\det Q_{ABC}|. \quad (12)$$

In both M_1 and M_2, to avoid going through all triplets of points in the region Ω_ℓ, the first two locations, A and B, are selected first, so that they are placed far apart one from the other: **(a)** first pixel location, $A = (x_\alpha, y_\alpha)$, is selected to be on the first column of the first row in the region mask, $x_\alpha = \min_{i=1,2,\ldots,n}\{x_i\}$, $y_\alpha = \min_{i=1,2,\ldots,n}\{y_i | x_i = x_\alpha\}$; **(b)** second pixel location $B = (x_\beta, y_\beta) = (x_{k_2}, y_{k_2})$ is selected at the maximum distance from (x_α, y_α), having the index $k_2 = \arg\max_{i=1,2,\ldots,n}\{(x_\alpha - x_i)^2 + (y_\alpha - y_i)^2\}$. Then the search in both M_1 and M_2 is done only for $C \in \Omega_\ell$.

2.2.2. Methods setting A, B, C using the bounding box of Ω_ℓ

Let us consider the bounding box for the region Ω_ℓ, formed using the coordinates $x_m = \min_i x_i$, $x_M = \max_i x_i$, $y_m = \min_i y_i$, $y_M = \max_i y_i$, $i = 1, 2, \ldots, n$. Let us denote also $\delta_x = x_M - x_m$, $\delta_y = y_M - y_m$, $\bar{x} = x_m + \frac{\delta_x}{2}$, $\bar{y} = y_m + \frac{\delta_y}{2}$.

($\mathbf{M_3}$) The triangle $\triangle ABC$ is set as the triangle having the maximum area inside the bounding box of Ω_ℓ. Hence, we select $A = (x_M, y_m)$, and if $\delta_x > \delta_y$, then $B = (x_m, y_m)$ and $C = (\bar{x}, y_M)$, else $B = (x_M, y_M)$ and $C = (x_m, \bar{y})$.

($\mathbf{M_4}$) The triangle $\triangle ABC$ is set as a triangle that includes the bounding box of Ω_ℓ. Hence, we select $A = (x_m - \delta_x, y_m)$, $B = (x_M, y_m)$, and $C = (x_M, y_M + \delta_y)$.

2.2.3. Methods with differential parameters

The next two methods, M_5 and M_6, are selecting the same three pixel locations $A = (x_M, y_m)$, $B = (x_m, y_M)$, $C = (x_m, y_m)$, and are encoding differential parameters. In the previous methods three rounded heights $\hat{z}_\alpha$, $\hat{z}_\beta$, and $\hat{z}_\gamma$ were selected as parameters, however here we select as parameters the following values: one rounded height, $\hat{z}_\gamma$, and two rounded height differences: $\hat{z}_\alpha - \hat{z}_\gamma$, $\hat{z}_\beta - \hat{z}_\gamma$. The differences can be written for the selected A, B, C, using (5) at encoder, and (6) at decoder, as:

$$\begin{aligned} \hat{z}_\alpha - \hat{z}_\gamma = \lfloor z^*_\alpha \rceil - \lfloor z^*_\gamma \rceil = \delta_x a^* - (\Delta^*_\alpha - \Delta^*_\gamma) &= \delta_x \tilde{a} \\ \hat{z}_\beta - \hat{z}_\gamma = \underbrace{\lfloor z^*_\beta \rceil - \lfloor z^*_\gamma \rceil = \delta_y b^* - (\Delta^*_\beta - \Delta^*_\gamma)}_{\text{encoder}} &= \underbrace{\delta_y \tilde{b}}_{\text{decoder}}. \end{aligned}$$

The new selected parameters are more sensitive to the rounding errors $\Delta^*_\alpha, \Delta^*_\beta, \Delta^*_\gamma$, and hence we choose to encode the parameters with a higher precision, by rounding the heights using N bits (N being different in M_5 and M_6) of their fractional part. The parameter $\hat{z}_\gamma$ is transmitted using 1-bit precision after the decimal point by encoding $2\hat{z}_\gamma = \lfloor 2z^*_\gamma \rceil$ using *Algorithm D* (see Section 2.3). The sign of each $\hat{z}_\alpha - \hat{z}_\gamma$, $\hat{z}_\beta - \hat{z}_\gamma$ is encoded using one bit.

($\mathbf{M}_5$) The absolute values $|\hat{z}_\alpha - \hat{z}_\gamma|$ and $|\hat{z}_\beta - \hat{z}_\gamma|$ are transmitted using 1-bit precision after the decimal point by encoding $|2(\hat{z}_\alpha - \hat{z}_\gamma)| = ||\lfloor 2z^*_\alpha \rceil - \lfloor 2z^*_\gamma \rceil|$ and $|2(\hat{z}_\beta - \hat{z}_\gamma)| = ||\lfloor 2z^*_\beta \rceil - \lfloor 2z^*_\gamma \rceil||$ using the Golomb-Rice (GR) algorithm.

($\mathbf{M}_6$) The values $|\hat{z}_\alpha - \hat{z}_\gamma|$ and $|\hat{z}_\beta - \hat{z}_\gamma|$ are transmitted by encoding the first $N_6 = 10$ bits of the following sub-unitary values: $||\lfloor 2^{-9} z^*_\alpha \rceil - \lfloor 2^{-9} z^*_\gamma \rceil|$ and $||\lfloor 2^{-9} z^*_\beta \rceil - \lfloor 2^{-9} z^*_\gamma \rceil|$. The decimal representation of the first k^*_{LP} bits is encoded using adaptive zero-order model with Laplace (LP) estimator, while the next $N_6 - k^*_{LP}$ bits are written directly in the output file. The optimal values k^*_{LP} for a^* and b^* are each encoded on $\log_2(10)$ bits.

2.2.4. Baseline method for encoding $\underline{\theta}^*$

($\mathbf{M}_7$) The parameters a^* and b^* are transmitted using $N_7 = 8$ bits precision after the decimal point by encoding $||\lfloor 2^{N_7} a^* \rceil|$ and $||\lfloor 2^{N_7} b^* \rceil|$ using the Golomb-Rice algorithm, and each sign by one bit. The parameter c^* is transmitted using 1-bit precision after the decimal point by encoding $\lfloor 2c^* \rceil$ using *Algorithm D*.

2.3. Encoding the planar model parameters

The parameters for the planar model are encoded differentially with respect to the parameter of the constant model, $\lfloor d_\ell \rceil$, which is very efficiently encoded by the original GSO method [6]. All differences are encoded by the *Algorithm D*. For the methods $M_1 : M_4$ the parameters $\hat{z}_\alpha$, $\hat{z}_\beta$, $\hat{z}_\gamma$ are encoded differentially with respect to $\lfloor d_\ell \rceil$, by encoding $v_j = \lfloor d_\ell \rceil - \hat{z}_i$, where $i = \alpha, \beta, \gamma$ is the parameter index, $\ell = 1, 2, \ldots, n_\Omega$ is the region index, and $j = 1, 2, \ldots n_V$ is the index in a vector V that collects these differences. For the methods M_5 and M_6, the parameter $\hat{z}_\gamma$ is encoded by the difference $v_j = 2\lfloor d_\ell \rceil - \lfloor 2z^*_\gamma \rceil$, while for M_7 the parameter c^* is encoded by the difference $v_j = 2\lfloor d_\ell \rceil - \lfloor 2c^* \rceil$.

[Algorithm D] The algorithm encodes each element v_j, $j = 1, 2, \ldots, n_V$ in a vector V, by its sign, using one bit, and its absolute value, $|v_i|$, using the adaptive zero-order model with Laplace estimator with ns_V symbols. The maximum absolute difference $v_M = \max_{j=1,2,\ldots,n_V} |v_j|$ is first found for V. The number of bits needed to represent v_M is computed as $n_M = \lceil \log_2(v_M) \rceil$, and encoded using the arithmetic coder for 7 symbols on $\log_2(7)$ bits. Finally, the value ns_V is set either $ns_V = 2^{n_M}$, or $ns_V = v_M$ if the codelength estimation shows that the transmission of v_M to the decoder improves the result.

2.4. Extending GSOm to use planar model for selected regions

In this paper we update the GSOm algorithm [6] by introducing model selection between the constant and planar model estimated by M_2, which is in general the best performer out of $M_1 : M_7$. We denote the new algorithm GSOmPF.

In GSOm, a competition was organized between the pair of regions to be merged, and the strategy was to select the pair that will result in the best slope of the RD, after that merge.

Now in GSOmPF the competition is extended, not only which region pair to be merged, but also which method of reconstruction to use for that pair. Now the strategy is to choose among all possible pairs and among the two models, the pair and model which lead to the best slope in RD after merging. From distortion point of view, if the exact parameters are encoded, then the planar model has a better distortion, but since for both models we use quantization of the estimated parameters, the ideal inequality $MSE_p < MSE_c$ might not be always true. From the rate point of view, a planar model has a higher codelength than a constant model. In our simplified model from [6], we estimated the codelength of a constant model parameter as $C_2 = 8$ bits, while now in GSOmPF we estimate the codelength of the planar model parameters as $C_3 = 12$ bits.

2.5. Depth compression using the planar model

The methods described above are implemented by a program written in C and denoted PF, that is using the CLAPACK routines to solve the systems of equation (1) and (6). Since *Algorithm D* uses $\lfloor d_\ell \rceil$ to encode the selected parameters, we used entropy coders for the average in each region and the region contours, while PF is improving the results by encoding the selected parameters as described above, and the model selection decision by adaptive zero-order model with LP estimator.

More implementation details can be found on the paper's webpage[1].

3. EXPERIMENTAL RESULTS

The following datasets are used in this paper: *Middlebury* [10], *Breakdancers* and *Ballet* [11]. Next we discuss the comparative results for 4 images, but results for 180 images can be found on the paper's webpage. The results for the introduced methods are obtained by first generating the lossy depth-map images using the GSOmPF algorithm, then using the entropy coder APC [9], and finally applying each method $M_1 : M_7$.

In Fig. 1.(a) we show the results for running M_7 with 7 different values for N_7. In most of our tests running M_7 with $N_7 = 8$ bits showed the best results. Hence, we selected $N_7 = 8$ for comparison of baseline with the other methods.

In Fig. 1.(b) the methods introduced in Section 2.2 are compared for the depth-map image *Aloe* (full size, disp1). The method M_2 shows best results for the $58-70$ dB range, M_5 for the $50-58$ dB range, while all the methods perform almost the same below 50 dB. Hence, we selected M_2 and M_5 for comparison with other methods on the rest of images.

In Fig. 1.(c)-(f) we show comparative results for four images for the following methods: **(a)** the method "Proposed_80" from [7], called here P80; **(b)** the method "Segment_Param_1" from [7], that was described in [8] and called here SP1; **(c)** our previous entropy coders CERV [2] and APC [9], where the lossy

[1]For additional information see www.cs.tut.fi/~schiopu/PF.

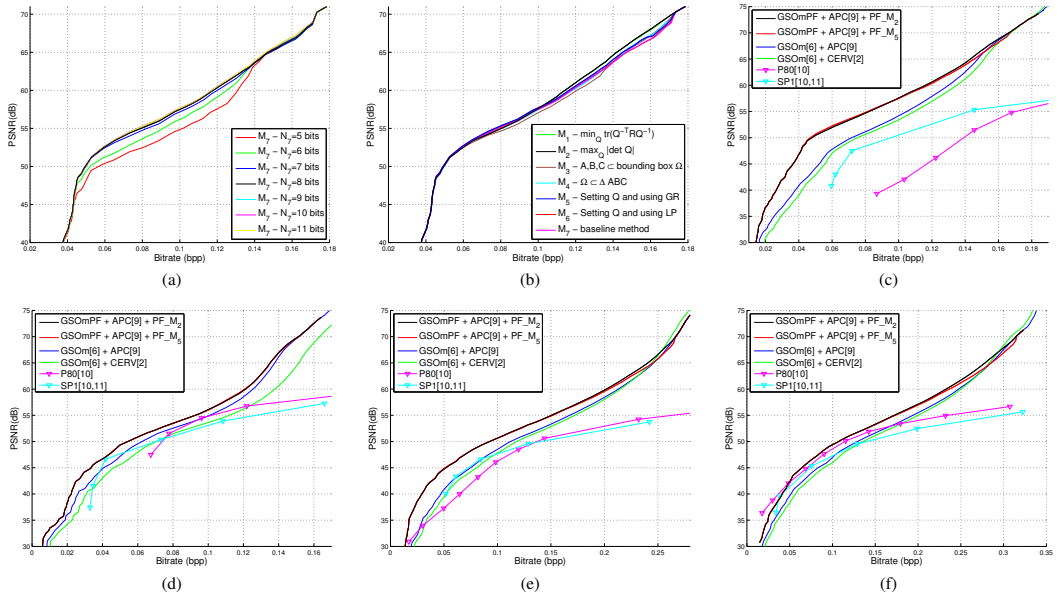


Figure 1. **(a)** Comparison between the results of baseline method M_7 (quantizing the parameters θ^* with $N_7 = 5, 6, \ldots, 11$ bits), for *Art* (full size, disp1); **(b)** Comparison between the seven methods $M_1 : M_7$ described in the paper, for *Art* (full size, disp1), where M_2 and M_5 have almost same performance, better than all others; **(c) - (f)** Comparison between the results of the following methods: P80 [7], SP1 [7, 8], Greedy Slope Optimization with region merging (GSOm) [6] using the entropy coders of CERV [2], APC [9], and the GSOm [6]+ APC [9] including M_2 and M_5 from this paper, for the following images: (c) *Aloe* (full size, disp1); (d) *Bowling1* (full size, disp1); (e) *Ballet* (cam0, frame 96); (f) *Breakdancers* (cam0, frame 0).

images are generated using GSOm [6]; **(d)** the two selected methods for comparison: M_2 and M_5. One can see that a significant improvement between 3 dB and 8 dB is obtained for both M_2 and M_5 algorithms. In Fig. 1.(c) we find a maximum improvement of around 8 dB, but for other images in the datasets a maximum improvement of almost 15 dB was achieved by M_2 and M_5.

4. CONCLUSIONS

We presented a method for value reconstruction inside a region using a reconstruction plane parameterized by its values at three pixel locations. Their quantized parameters are encoded efficiently by *Algorithm D*. Several methods for finding the pixel location were introduced and two of them consistently outperformed the others. There are further possibilities for improvement of the presented results, if one chooses out of the seven methods the best method for a certain lossy image at certain rate target, and involves extra computations only at the encoder. Model selection was introduced in the structure of the greedy slope optimization method GSOm [6], resulting in overall rate-distortion performance significantly better than the original GSOm method.

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

[3] 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*, Oct. 2007, pp. V–105 – V–108.

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

[5] B.O. Ozkalayci and A.A. Alatan, "3D planar representation of stereo depth images for 3DTV applications," *IEEE Trans. Image Process.*, vol. 23, no. 12, pp. 5222–5232, Dec. 2014.

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

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

[8] P. Zanuttigh and G.M. Cortelazzo, "Compression of depth information for 3D rendering," in *3DTV-CON*, May 2009, pp. 1–4.

[9] I. Schiopu and I. Tabus, "Anchor points coding for depth map compression," in *Proc. IEEE ICIP*, Paris, France, Oct. 2014, pp. 5626–5630.

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

[11] 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, 2004, pp. 600–608.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-3667-0
ISSN 1459-2045