

Context Tree based Image Contour Coding using A Geometric Prior

Amin Zheng *Student Member, IEEE*, Gene Cheung *Senior Member, IEEE*, Dinei Florencio *Fellow, IEEE*

Abstract—If object contours in images are coded efficiently as side information, then they can facilitate advanced image / video coding techniques, such as graph Fourier transform coding or motion prediction of arbitrarily shaped pixel blocks. In this paper, we study the problem of lossless and lossy compression of detected contours in images. Specifically, we first convert a detected object contour composed of contiguous between-pixel edges to a sequence of directional symbols drawn from a small alphabet. To encode the symbol sequence using arithmetic coding, we compute an optimal variable-length context tree (VCT) \mathcal{T} via a maximum a posteriori (MAP) formulation to estimate symbols' conditional probabilities. MAP prevents us from overfitting given a small training set \mathcal{X} of past symbol sequences by identifying a VCT \mathcal{T} that achieves a high likelihood $P(\mathcal{X}|\mathcal{T})$ of observing \mathcal{X} given \mathcal{T} , and a large geometric prior $P(\mathcal{T})$ stating that image contours are more often straight than curvy. For the lossy case, we design efficient dynamic programming (DP) algorithms that optimally trade off coding rate of an approximate contour \hat{x} given a VCT \mathcal{T} with two notions of distortion of \hat{x} with respect to the original contour x . To reduce the size of the DP tables, a total suffix tree is derived from a given VCT \mathcal{T} for compact table entry indexing, reducing complexity. Experimental results show that for lossless contour coding, our proposed algorithm outperforms state-of-the-art context-based schemes consistently for both small and large training datasets. For lossy contour coding, our algorithms outperform comparable schemes in the literature in rate-distortion performance.

Index Terms—contour coding, lossless coding, statistical learning, image compression

I. INTRODUCTION

Advances in depth sensing technologies like Microsoft Kinect 2.0 mean that depth images—per pixel distances between physical objects in a 3D scene and the camera—can now be captured easily and inexpensively. Depth imaging has in turn eased the detection of object contours in a captured image, which was traditionally a challenging computer vision problem [1]. If detected object contours in images are compressed efficiently as *side information* (SI), then they can enable advanced image / video coding techniques such as *graph Fourier transform* (GFT) coding [2–5] and motion prediction [6, 7] of arbitrarily shaped pixel blocks. Moreover, coded contours can be transmitted to a central cloud for computation-intensive object detection or activity recognition [8], at a much lower coding cost than compressed depth video. We focus on the problem of coding object contours in this paper.

A. Zheng is with Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, China (e-mail: amzheng@connect.ust.hk).

G. Cheung is with National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, Japan 101-8430 (e-mail: cheung@nii.ac.jp).

D. Florencio is with Microsoft Research, Redmond, WA USA (e-mail: dinei@microsoft.com).

Object contour coding was studied extensively during 1990's when the concept of *object-based video coding* (OBVC) was popular during the MPEG-4 video coding standardization. However, the shape coding techniques developed to represent boundaries of objects [9]—called *video object planes* (VOP)—were not efficient enough, resulting in large SI overhead that rendered OBVC uncompetitive in compression performance. Some contemporary interest in the problem has shifted to lossy contour coding, where curve- and spline-based approximations are used [10–12].

Two recent technological trends have provided new ammunition to revisit the old lossless contour coding problem. First, advances in statistical machine learning have led to sophisticated algorithms that construct suitable *variable-length context trees* (VCT) to estimate conditional probabilities given a training dataset of symbol strings [13–16]. Estimated probability distributions can then be used to compress symbol strings of finite alphabets via arithmetic coding [17]. Second, the availability of fast computing resource, locally or in a nearby cloudlet [18], allows this statistical learning on relevant training datasets to be performed in real-time in restricted cases. While promising, the convergence of these developments has led to a new “*small data*” *statistical learning problem*: due to either statistical non-stationarity and/or tight real-time application requirements, the size of dataset used for statistical training may be limited, and data overfitting becomes a significant concern.

In response, in this paper we propose a *maximum a posteriori* (MAP) formulation to optimize a VCT \mathcal{T} for lossless and lossy contour coding. In particular, given small training data \mathcal{X} , we select a VCT \mathcal{T} that has *both* high likelihood $P(\mathcal{X}|\mathcal{T})$ (agreement between constructed contexts and observations), and high prior probability $P(\mathcal{T})$ —a geometric prior stating that image contours are more often straight than curvy. Like *Bayesian information criterion* (BIC) [19], we design the prior weight parameter α to scale naturally with training data size, so that when the volume of relevant training data becomes larger and thus more reliable, α becomes smaller.

For lossy coding, we design efficient dynamic programming (DP) algorithms that optimally trade off coding rate of an approximate contour \hat{x} given a VCT \mathcal{T} with two different notions of distortion of \hat{x} with respect to the original contour x : *sum squared distance distortion* (SSDD) and *maximum absolute distance distortion* (MADD) [9]. To reduce the size of the DP tables, a new tool called *total suffix tree* (TST) is derived from a VCT \mathcal{T} for compact table entry indexing, reducing computation complexity. Experimental results show that for lossless contour coding, our proposed algorithm outperforms

state-of-the-art context-based schemes [7, 13] consistently for both small and large training datasets. For lossy contour coding, our algorithms outperform comparable schemes in the literature [9, 12] in rate-distortion (RD) performance. Towards the goal of reproducible research [20], we have made the source code for lossless contour coding publicly available¹.

The outline of the paper is as follows. We first discuss related works in Section II. We then discuss our proposal for lossless and lossy contour coding in Section III and IV, respectively. We discuss efficient coding of contour starting points in Section V. Finally, we present experimental results and conclusion in Section VI and VII, respectively.

II. RELATED WORK

A. Lossless Contour Coding

Most works in lossless contour coding [7, 17, 21–27] first convert an image contour into a *chain code* [28]: a sequence of symbols each representing one of four or eight possible *absolute* directions on the pixel grid. Alternatively, a *differential chain code* (DCC) [21] that specifies *relative* directions instead can be used. DCC symbols are entropy-coded using either Huffman [22] or arithmetic coding [17] given symbol probabilities. The challenge is to estimate conditional probabilities for DCC symbols given a set of training data; this is the core problem we address in Section III.

[7, 29] propose a linear geometric model to estimate conditional probabilities of the next DCC symbol. In summary, given a window of previous edges, a line-of-best-fit that minimizes the sum of distances to the edges' endpoints is first constructed. Then the probability of a candidate direction for the next symbol is assumed inversely proportional to the angle difference between the direction and the fitted line. We show in Section VI that this scheme is inferior in estimating symbol probabilities compared to context models, because there are only a few possible angle differences for a small number of previous edges, limiting the expressiveness of the model.

An alternative approach is *context modeling*: given a window of l previous symbols (context) \mathbf{x}_{i-l}^{i-1} , compute the conditional probability $P(x_i | \mathbf{x}_{i-l}^{i-1})$ of the next symbol x_i by counting the number of occurrences of \mathbf{x}_{i-l}^{i-1} followed by x_i in the training data. In [17, 23–25, 30], Markov models of fixed order up to eight are used for lossless coding. However, in applications where the training data is small, there may be not enough occurrences of \mathbf{x}_{i-l}^{i-1} to reliably estimate the conditional probabilities.

VCT [13, 14] provides a more flexible approach for Markov context modeling by allowing the context to have variable length. There are many ways to construct the VCT: Lempel-Ziv-78 (LZ78) [31], prediction by partial matching (PPM) [15], and probabilistic suffix trees (PST) [16]. LZ78 constructs a dictionary from scratch using the input data directly as training data. The probability estimation quality varies depending on the order of first appearing input symbols. PPM considers all contexts restricted by a maximum length with non-zero occurrences in the training data when building VCT.

PPM is efficient for lossless sequential data compression if sufficient training data is available [26, 27], but may suffer from overfitting if training data is limited.

PST first constructs an initial VCT similar to PPM, and then the initial VCT is pruned using five user-selected thresholds [16]. PST algorithm is widely used for protein modeling problems [32, 33]. Through pruning, PST algorithm can avoid overfitting in some cases. However, choosing properly the five application-specific thresholds used for pruning is difficult in general. In contrast, we propose a geometric shape prior—requiring only one parameter—to avoid overfitting specifically for contour coding.

B. Lossy Contour Coding

There are two main approaches to lossy contour coding: chain-code-based and vertex-based. In the first approach, the contours are first converted to chain codes as done in the lossless case. Then the chain codes are approximated according to different criteria before entropy coding. In [34], an approximated chain code must be composed of several predefined sub-chain codes, and the indices of the sub-chain codes are coded. In [35], a line processing technique generates straighter contour segments than the input, which are then efficiently compressed. In [7], a chain code is simplified by removing “irregularities”, which are predefined non-smooth edge patterns. All these approaches approximate the chain codes without considering explicitly distortion due to approximation, and one cannot specify a desired compression ratio. In contrast, we approximate a chain code via RD optimization algorithms, so that different compression ratios can be achieved.

Vertex-based approaches select representative points called vertices along the contour for coding. [36, 37] use a top-down / bottom-up framework to select and code vertices independently. [9, 38, 39] propose an operational rate-distortion (ORD) optimal algorithm to jointly select and code vertices. Many improvements on ORD have since been proposed, including [10–12, 40, 41]. Compared to the chain-code-based approach, the vertex-based approach requires fewer points for coding, but each point (vertex) will consume more bits. We follow the chain-code-based approach for lossy coding in Section IV.

III. LOSSLESS CONTOUR CODING

We first propose an algorithm to efficiently code object contours *losslessly* in a target image. A small set of training images are used for statistical learning; we assume that the target image and the training images are statistically correlated, such as consecutive frames in a video sequence. We assume also that, as a pre-processing step, object contours in an image have first been either outlined manually, or detected automatically using an existing method such as gradient-based edge detection [7]. Each contour is defined by a starting point and a sequence of connected edges. See Fig. 1 for two example contours of VOPs in a frame of MPEG4 test sequence *news*². Our objective here is to losslessly encode the sequence of connected edges in a given contour; coding of the starting points of contours is discussed later in Section V.

¹<http://research.nii.ac.jp/~cheung/software.html>

²ftp://ftp.tnt.uni-hannover.de/pub/MPEG/mpeg4_masks/

TABLE I
NOTATIONS FOR LOSSLESS & LOSSY CONTOUR CODING

Notation	Description
\mathcal{D}	alphabet of relative directions, $\mathcal{D} = \{l, s, r\}$
\mathcal{A}	alphabet of absolute directions, $\mathcal{A} = \{N, E, S, W\}$
\mathcal{X}	training set of DCC strings
$\mathbf{x}(m)$	m -th DCC string in training set \mathcal{X}
M	number of DCC strings in \mathcal{X}
l_m	number of symbols in string $\mathbf{x}(m)$
L	total number of symbols in \mathcal{X}
$N(\mathbf{u})$	number of occurrences of sub-string \mathbf{u} in \mathcal{X}
$P(x \mathbf{u})$	conditional probability of symbol x given context \mathbf{u}
\mathbf{x}, x_i	DCC string and its i -th symbol, $x_i \in \mathcal{D}$
N	length of the DCC string
$\hat{\mathbf{x}}$	approximated DCC string
\hat{N}	length of the approximated DCC string
\mathbf{x}_i^j	A sub-string of length $j - i + 1$ from x_i to x_j , $\mathbf{x}_i^j = [x_j, x_{j-1}, \dots, x_i]$
\mathbf{w}	context
$\mathcal{T}, \mathcal{T}^*$	context tree, an optimal context tree
\mathcal{F}	context forest composed of all possible context trees
D	maximum depth of the context tree
\mathcal{T}^0	initial context tree
K	Number of nodes on \mathcal{T}^0
\mathcal{T}_s	total suffix tree
α	prior weight parameter, $\alpha = a \ln L$
D_{\max}	maximum distortion of the approximated DCC string

We first overview our lossless contour coding algorithm, based on the general concept of *context tree* model [13, 14]. The algorithm is composed of the following steps:

- 1) Convert each contour in a set of training images and a target image into a *differential chain code* (DCC) [21]—a string of symbols each chosen from a size-three alphabet.
- 2) Construct a suitable context tree given the DCC strings in the training images by solving a *maximum a posteriori* (MAP) problem.
- 3) For each symbol in an input DCC string (corresponding to a contour in the target image), identify the conditional probability distribution using the constructed context tree, which is used for arithmetic coding of the symbol.

We discuss these steps in order. Notations for the technical discussions in the sequel are shown in Table I.

A. Differential Chain Code

We first convert each contour into a DCC string; DCC is one member of the family of chain codes proposed by Freeman [21]. A contour is composed of a sequence of “between-pixel” edges that divide pixels in the local neighborhood into two sides, as illustrated in Fig. 2(a). A length- $(N + 1)$ contour can be compactly described as a symbol string denoted by $\mathbf{x}^o = [x_0, \dots, x_N]$. For the first edge x_0 , we assume equal probability for each of four *absolute* directions, north, east, south and west, with respect to the starting point. For each subsequent DCC symbol x_i , $i \geq 1$, only three *relative* directions are possible on a 2D grid with respect to the previous symbol x_{i-1} : left, straight and right, as shown in Fig. 2(b). We denote them by l , s and r , respectively, which constitute a size-three alphabet $\mathcal{D} = \{l, s, r\}$. The problem



Fig. 1. An example of object contours of VOP in an MPEG4 video frame: (a) input image. (b) image with two object contours. The contours are the edges between the green and the red pixels. The starting points of the contours are indicated by the white arrows.

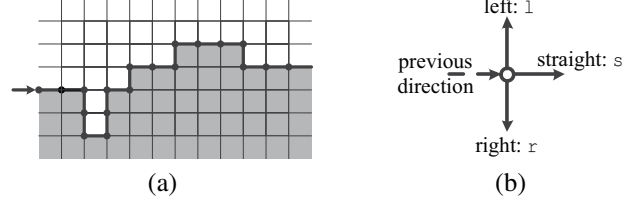


Fig. 2. (a) An example of a contour represented by a four-connected chain codes: east-s-r-s-l-l-s-r-l-r-s-l-r-s-s-r-l-s-s. (b) directional code.

is thus to code a DCC string \mathbf{x} (without the first edge), where $x_i \in \mathcal{D}$ for $i \geq 1$.

B. Definition of Context Tree

We first define notations. Denote by $\mathbf{x}(m)$, $1 \leq m \leq M$, the m -th DCC string in the training set $\mathcal{X} = \{\mathbf{x}(1), \dots, \mathbf{x}(M)\}$, where M denotes the total number of DCC strings in \mathcal{X} . The length of $\mathbf{x}(m)$ is denoted by l_m , and the total number of symbols in \mathcal{X} is denoted by $L = \sum_{m=1}^M l_m$.

Denote by $\mathbf{x}_i^j = [x_j, x_{j-1}, \dots, x_i]$, $i < j$ and $i, j \in \mathbb{Z}^+$, a *sub-string* of length $j - i + 1$ from the i -th symbol x_i to the j -th symbol x_j in reverse order. Further, denote by \mathbf{uv} the concatenation of sub-strings \mathbf{u} and \mathbf{v} .

We now define $N(\mathbf{u})$ as the number of occurrences of sub-string \mathbf{u} in the training set \mathcal{X} . $N(\mathbf{u})$ can be computed as:

$$N(\mathbf{u}) = \sum_{m=1}^M \sum_{i=1}^{l_m - |\mathbf{u}| + 1} \mathbf{1}(\mathbf{x}(m)_i^{i+|\mathbf{u}|-1} = \mathbf{u}) \quad (1)$$

where $\mathbf{1}(\mathbf{c})$ is an indicator function that evaluates to 1 if the specified binary clause \mathbf{c} is true and 0 otherwise.

Denote by $P(x|\mathbf{u})$ the conditional probability of symbol x occurring given its previous sub-string is \mathbf{u} , where $x \in \mathcal{D}$. Given training data \mathcal{X} , $P(x|\mathbf{u})$ can be estimated using $N(\mathbf{u})$ as done in [32],

$$P(x|\mathbf{u}) = \frac{N(\mathbf{xu})}{N(\mathbf{u})} \quad (2)$$

Given \mathcal{X} , we learn a context model to assign a conditional probability to any symbol given its previous symbols in a DCC string. Specifically, to calculate the conditional probability $P(x_i|\mathbf{x}_1^{i-1})$ for the symbol x_i given all its previous symbols \mathbf{x}_1^{i-1} , the model determines a *context* \mathbf{w} to calculate $P(x_i|\mathbf{w})$,

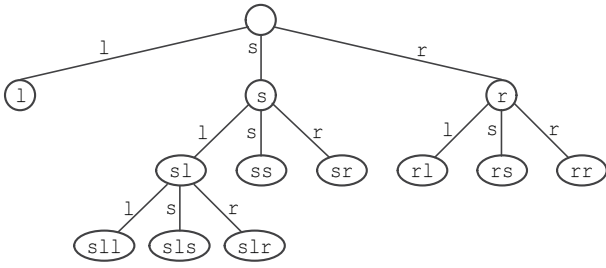


Fig. 3. An example of context tree. Each node is a sub-string and the root node is an empty sub-string. The contexts are all the end nodes on \mathcal{T} : $\mathcal{T} = \{l, sll, sls, slr, ss, sr, rl, rs, rr\}$.

where \mathbf{w} is a *prefix* of the sub-string \mathbf{x}_1^{i-1} , i.e., $\mathbf{w} = \mathbf{x}_{1-l}^{i-1}$ for some context length l :

$$P(x_i | \mathbf{x}_1^{i-1}) = P(x_i | \mathbf{w}) \quad (3)$$

$P(x_i | \mathbf{w})$ is calculated using (2) given \mathcal{X} . The context model determines a unique context \mathbf{w} of finite length for every possible past \mathbf{x}_1^{i-1} . The set of all mappings from \mathbf{x}_1^{i-1} to \mathbf{w} can be represented compactly as a context tree.

Denote by \mathcal{T} the context tree, where \mathcal{T} is a *full ternary tree*: each node has either zero children (an *end node*) or all three children (an *intermediate node*). The root node has an empty sub-string, and each child node has a sub-string $\mathbf{u}x$ that is a concatenation of: i) its parent's sub-string \mathbf{u} if any, and ii) the symbol x (one of l, s and r) representing the link connecting the parent node and itself in \mathcal{T} . An example is shown in Fig. 3. The sub-strings of the end nodes are the contexts of the tree \mathcal{T} . For each \mathbf{x}_1^{i-1} , a context \mathbf{w} is obtained by traversing \mathcal{T} from the root node until an end node, matching symbols x_{i-1}, x_{i-2}, \dots into the past.

All possible context trees constitute a *context forest* denoted by \mathcal{F} . The modeling problem is to find the best tree \mathcal{T} from the forest \mathcal{F} given \mathcal{X} .

C. Problem Definition for Optimal Context Tree

Given limited observations \mathcal{X} , we aim to find a suitable tree \mathcal{T} that best describes \mathcal{X} without overfitting. We first write the posterior probability of \mathcal{T} given \mathcal{X} via Bayes' rule:

$$P(\mathcal{T} | \mathcal{X}) = \frac{P(\mathcal{X} | \mathcal{T})P(\mathcal{T})}{P(\mathcal{X})} \quad (4)$$

where $P(\mathcal{X} | \mathcal{T})$ is the likelihood of observing \mathcal{X} given context tree \mathcal{T} , and $P(\mathcal{T})$ is the prior which describes *a priori* knowledge about the context tree. We next define the likelihood and prior terms, and then use the MAP estimator to formulate the context tree optimization problem.

1) *Likelihood Term*: The likelihood is defined as the joint conditional probability of all the observed symbols in \mathcal{X} given their past and \mathcal{T} ,

$$P(\mathcal{X} | \mathcal{T}) = \prod_{m=1}^M \prod_{i=1}^{l_m} P(x(m)_i | \mathbf{x}(m)_1^{i-1}, \mathcal{T}) \quad (5)$$

Given tree \mathcal{T} , for each symbol $x(m)_i$ a prefix (context) \mathbf{w} of past symbols $\mathbf{x}(m)_1^{i-1}$ is identified to compute the conditional

probability. Hence (5) can be rewritten as follows using (3), similarly done in [42]:

$$P(\mathcal{X} | \mathcal{T}) = \prod_{\mathbf{w} \in \mathcal{T}} \prod_{x \in \mathcal{D}} P(x | \mathbf{w})^{N(x\mathbf{w})} \quad (6)$$

2) *Prior Term*: Overfitting occurs if the complexity of a model is too large for the given observed data size. In the case of context tree, it means that the number of occurrences $N(\mathbf{u})$ of a particular context \mathbf{u} is too small to have probabilities $P(x | \mathbf{u})$ reliably estimated using $\frac{N(x\mathbf{u})}{N(\mathbf{u})}$. To avoid overfitting, one can design a prior to control the size of \mathcal{T} —the complexity of the model—depending on the volume of training data. This general idea is used for example in [42], where *Bayesian information criterion* (BIC) [19] is employed to constrain the order of a fixed-length Markov model for given data size.

In the case of contour coding, we propose a *geometric prior*, defined as the sum of *straightness*, $s(\mathbf{w})$, of all contexts \mathbf{w} in tree \mathcal{T} , based on the assumption that contours in natural images are more likely straight than curvy. We calculate $s(\mathbf{w})$ as follows. We first map a context \mathbf{w} to a *shape segment* on a 2D grid with $|\mathbf{w}| + 2$ points from the most recent symbol $w_{|\mathbf{w}|}$ to the symbol w_1 furthest in the past *plus* an initial edge. Without loss of generality, we assume that the absolute direction of the initial edge is East. As an example, Fig. 4(b) shows a context $\mathbf{w} = lrl$ with five points, where the dotted arrow is the initial edge. Denote by \mathbf{p}_k , $1 \leq k \leq |\mathbf{w}| + 2$, the 2D coordinate of the k -th point. $s(\mathbf{w})$ is defined as the *maximum distance* $\text{dist}()$ from any \mathbf{p}_k to a straight line $f(\mathbf{p}_1, \mathbf{p}_{|\mathbf{w}|+2})$ connecting \mathbf{p}_1 and $\mathbf{p}_{|\mathbf{w}|+2}$,

$$s(\mathbf{w}) = \max_k \text{dist}(\mathbf{p}_k, f(\mathbf{p}_1, \mathbf{p}_{|\mathbf{w}|+2})) \quad (7)$$

Some examples of $s(\mathbf{w})$ are shown in Fig. 4.

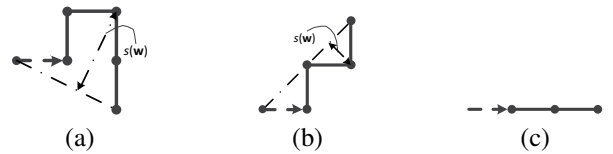


Fig. 4. Three examples of $s(\mathbf{w})$. (a) $\mathbf{w} = srll$ and $s(\mathbf{w}) = 4\sqrt{5}/5$. (b) $\mathbf{w} = lrl$ and $s(\mathbf{w}) = \sqrt{2}/2$. (c) $\mathbf{w} = ss$ and $s(\mathbf{w}) = 0$.

We can now define prior $P(\mathcal{T})$ based on the sum of $s(\mathbf{w})$ of all contexts \mathbf{w} in \mathcal{T} as follows:

$$P(\mathcal{T}) = \exp\left(-\alpha \sum_{\mathbf{w} \in \mathcal{T}} s(\mathbf{w})\right) \quad (8)$$

where α is an important parameter to be discussed soon. We see that in general a tree \mathcal{T} with fewer contexts \mathbf{w} has a smaller sum in (8), meaning that our prior tends to reduce the number of parameters. Further, in general fewer contexts also means a shallower context tree. Thus, our prior promotes shorter contexts, which is also reasonable.

3) *MAP Estimation*: We can now write the optimal context tree problem via a MAP formulation:

$$\mathcal{T}^* = \arg \max_{\mathcal{T} \in \mathcal{F}} P(\mathcal{X} | \mathcal{T})P(\mathcal{T}) \quad (9)$$

(9) can be rewritten as:

$$\mathcal{T}^* = \arg \max_{\mathcal{T} \in \mathcal{F}} \left\{ \prod_{\mathbf{w} \in \mathcal{T}} \prod_{x \in \mathcal{D}} P(x|\mathbf{w})^{N(x\mathbf{w})} \cdot \exp \left(-\alpha \sum_{\mathbf{w} \in \mathcal{T}} s(\mathbf{w}) \right) \right\} \quad (10)$$

For ease of computation, we minimize the negative log of (10) instead and divide by L :

$$F(\mathcal{T}) = -\frac{1}{L} \sum_{\mathbf{w} \in \mathcal{T}} \sum_{x \in \mathcal{D}} N(x\mathbf{w}) \cdot \ln P(x|\mathbf{w}) + \frac{\alpha}{L} \sum_{\mathbf{w} \in \mathcal{T}} s(\mathbf{w}) \quad (11)$$

The first term in (11) can be interpreted as the average information of all symbols in observed \mathcal{X} . The second term is the average straightness of all contexts. α weighs the relative importance of the prior against the likelihood.

4) *Selection of Weighting Parameter*: Similar in principle to BIC [19], we define $\alpha = a \ln L$, where a is a chosen parameter. In doing so, when the size of the training data L becomes larger, the weight factor for the prior $a \ln L/L$ becomes smaller. This agrees with the intuition that when sufficient training data is available, the prior term becomes less important.

D. Optimization of Context Tree

The optimization of context tree consists of two main steps. We first construct an initial context tree denoted by \mathcal{T}^0 by collecting statistics from \mathcal{X} . We then prune \mathcal{T}^0 to get the optimal context tree \mathcal{T}^* by minimizing the objective (11).

1) *Construction of Initial Context Tree*: Since \mathcal{T}^* is a sub-tree of \mathcal{T}^0 , each node (intermediate node or end node) in \mathcal{T}^0 can be a potential context (end node) in \mathcal{T}^* . Given a maximum tree depth D , [32, 43] construct \mathcal{T}^0 by collecting statistics for all nodes, i.e., $(3^{D+1} - 1)/2$, which means that the required memory is exponential with respect to D . To reduce the memory requirement, we enforce an upper-bound K on the number of nodes in \mathcal{T}^0 given D . Specifically, we first choose $D = \lceil \ln L / \ln 3 \rceil$ as done in [14], which ensures a large enough D to capture natural statistics of the training data of length L . We then choose $K = 3D^3$ in our experiments, which is much smaller than $(3^{D+1} - 1)/2$.

Having chosen D and K , we parse the training data \mathcal{X} to collect statistics for K potential contexts. Specifically, we traverse \mathcal{X} once to first collect statistics for the first $2K$ different sub-strings \mathbf{u} we encounter, where $|\mathbf{u}| \leq D$. Each sub-string \mathbf{u} has three counters which store the number of occurrences of sub-strings $l\mathbf{u}$, $s\mathbf{u}$ and $r\mathbf{u}$, i.e., $N(l\mathbf{u})$, $N(s\mathbf{u})$ and $N(r\mathbf{u})$. Then we keep only the K sub-strings with the largest numbers of occurrences to construct \mathcal{T}^0 , as described in Algorithm 1.

The obtained tree by Algorithm 1 may not be a full tree, because some intermediate nodes may have only one or two children. Thus, we add children to these intermediate nodes to ensure that each intermediate node has three children. We assume that the statistics of a newly added child $u\mathbf{v}$ is the same as its parent \mathbf{u} ; i.e., $P(x|u\mathbf{v}) = P(x|\mathbf{u})$. The occurrences of the added children are set to ensure that the total number of

Algorithm 1 Choose K potential contexts

- 1: Initialize \mathcal{T}^0 to an empty tree with only root node
 - 2: **for** each symbol x_i , $i \geq D + 2$ in \mathcal{X} , match x_{i-k}^{i-1} with nodes on \mathcal{T}^0 from $k = 1$ to $k = D$ in order **do**
 - 3: **if** there exist a node $\mathbf{u} = x_{i-k}^{i-1}$ on \mathcal{T}^0 **then**
 - 4: increase the counter $N(x_i\mathbf{u})$ by 1
 - 5: **else if** number of nodes on \mathcal{T}^0 is less than $2K$ **then**
 - 6: add node $\mathbf{u} = x_{i-k}^{i-1}$ to \mathcal{T}^0
 - 7: **end if**
 - 8: **end for**
 - 9: Sort nodes on \mathcal{T}^0 by $N(\mathbf{u})$ in descending order and choose the first K nodes from the sorted nodes.
-

occurrences of all three children is equal to the number of occurrence of their parent. Specifically, $N(\mathbf{u}v) = N(\mathbf{u}) - \sum_{z \in \mathcal{D}, z \neq v} N(\mathbf{u}z)$ if only one child $\mathbf{u}v$ is added, and $N(\mathbf{u}v) = N(\mathbf{u}s) = \frac{1}{2}(N(\mathbf{u}) - \sum_{z \in \mathcal{D}, z \neq v, z \neq s} N(\mathbf{u}z))$ if two children $\mathbf{u}v$ and $\mathbf{u}s$ are added.

After the adding procedure, we arrive at an initial context tree \mathcal{T}^0 with maximum depth D . The memory requirement of the initialization is $O(3 \cdot 2K) = O(K)$, and the time complexity is $O(K \cdot L)$.

2) *Pruning of Initial Context Tree*: The obtained initial tree \mathcal{T}^0 is then pruned to minimize the objective (11), resulting in an optimal tree \mathcal{T}^* . Since both the likelihood and the prior in (11) are summations of all the contexts (end nodes) in \mathcal{T} , we rewrite the objective as the sum of *end node cost* denoted by $f(\mathbf{w})$:

$$F(\mathcal{T}) = \sum_{\mathbf{w} \in \mathcal{T}} f(\mathbf{w}) \quad (12)$$

where

$$f(\mathbf{w}) = -\frac{1}{L} \sum_{x \in \mathcal{D}} N(x\mathbf{w}) \cdot \ln P(x|\mathbf{w}) + a \cdot \frac{\ln L}{L} \cdot s(\mathbf{w}) \quad (13)$$

$f(\mathbf{w})$ is the cost of end node \mathbf{w} on \mathcal{T} .

We minimize (12) recursively by dividing the original problem into sub-problems. The sub-problem is to minimize the end node cost $F(\mathcal{T}_{\mathbf{u}}^0)$ of a sub-tree $\mathcal{T}_{\mathbf{u}}^0$ rooted at node \mathbf{u} . Specifically, we define a recursive function $J(\mathcal{T}_{\mathbf{u}}^0)$ that minimizes $F(\mathcal{T}_{\mathbf{u}}^0)$ as follows:

$$J(\mathcal{T}_{\mathbf{u}}^0) = \min\{J(\mathcal{T}_{\mathbf{ul}}^0) + J(\mathcal{T}_{\mathbf{us}}^0) + J(\mathcal{T}_{\mathbf{ur}}^0), f(\mathbf{u})\} \quad (14)$$

In words, (14) states that we can either treat node \mathbf{u} as an end node and compute its cost $f(\mathbf{u})$ (and as a result eliminating all nodes in the sub-tree below), or treat it as an intermediate node and recurse. The complexity of (14)—the total number of recursions—with initial tree \mathcal{T}^0 as argument is proportional to the number of nodes, hence $O(K)$.

3) *Analysis of the Optimization Algorithm*: To better understand the relationship between the likelihood and the depth of the context tree, for an intermediate node \mathbf{u} with three end-node children on tree \mathcal{T} , we examine the change in likelihood if the three end-node children are pruned. Following (11), the change in likelihood is calculated as follows:

$$\begin{aligned}
& -\frac{1}{L} \sum_{v \in \mathcal{D}} \sum_{x \in \mathcal{D}} N(x|uv) \ln P(x|uv) + \frac{1}{L} \sum_{x \in \mathcal{D}} N(x|u) \ln P(x|u) \\
& = -\frac{1}{L} \sum_{v \in \mathcal{D}} N(uv) \sum_{x \in \mathcal{D}} P(x|uv) \ln \frac{P(x|uv)}{P(x|u)} \\
& = -\frac{1}{L} \sum_{v \in \mathcal{D}} N(uv) D_{KL}(P(\cdot|uv) || P(\cdot|u))
\end{aligned} \tag{15}$$

where $D_{KL}(P(\cdot|uv) || P(\cdot|u))$ is the *Kullback-Leibler divergence* (KLD) [44] of $P(\cdot|u)$ from $P(\cdot|uv)$, which is non-negative. As discussed previously, the average negative log of the likelihood can be regarded as the average information. Hence this difference is the average information gain when the children of node u are pruned. In general, we can conclude that the log likelihood term becomes smaller when the tree grows deeper. On the other hand, the log prior becomes larger when the tree grows deeper. Our objective is thus to find a properly sized context tree that balances these two terms.

E. Adaptive Arithmetic Coding

For each symbol x_i in the DCC strings of the target image, we first find the matched context \mathbf{w} of x_i , i.e., $\mathbf{w} = \mathbf{x}_{i-|\mathbf{w}|}^{i-1}$, and get the corresponding conditional probability distribution $P(x_i|\mathbf{w})$ from the resulting optimal context tree \mathcal{T}^* . Then, the probability distribution is inputted into an adaptive arithmetic coder [17] to encode x_i . The length of the DCC string is also losslessly coded using fixed length binary coding.

IV. LOSSY CONTOUR CODING

We now discuss lossy contour coding. Specifically, we approximate each DCC string in the test image by minimizing an RD cost. The rate of encoding an approximated DCC string is computed using a context tree \mathcal{T}^* , constructed from training data as described previously. Two distortion metrics, *sum squared distance distortion* (SSDD) and *maximum absolute distance distortion* (MADD), are introduced for different applications. We first discuss the two distortion metrics. Then we discuss the algorithms of approximating the DCC strings using one of the two metrics in order.

A. Distortion Definition

When approximating a contour, the chosen distortion metric should be application-specific. If the intended application is image / video compression, where coded contours are used as side information to facilitate transform coding [4] or motion prediction of arbitrarily shaped blocks [7], then a metric measuring the aggregate distortion between the original and approximated contours would be appropriate. SSDD would be a suitable metric in this case.

Denote by \mathbf{x} and $\hat{\mathbf{x}}$ the original and approximated DCC strings respectively, and by N and \hat{N} the lengths of \mathbf{x} and $\hat{\mathbf{x}}$. To describe a DCC string geometrically on a 2D grid, we first map \mathbf{x} to a *contour segment*, composed of contiguous vertical or horizontal edges. Fig. 2(a) shows a contour segment and

the corresponding DCC string. Specifically, given a default starting point $(0, 0)$ on the 2D grid, we determine the i -th edge relative to the $(i-1)$ -th edge using the i -th symbol x_i . Denote by $\mathbf{p}_{\mathbf{x}}(i)$ the 2D coordinate of the endpoint of the i -th edge and $a_{\mathbf{x}}(i) \in \mathcal{A} = \{\text{N, E, S, W}\}$ the absolute direction of the i -th edge. The i -th edge is uniquely determined by the coordinate $\mathbf{p}_{\mathbf{x}}(i)$ and the absolute direction $a_{\mathbf{x}}(i)$ alone.

Denote by $d(\mathbf{p}_{\hat{\mathbf{x}}}(j), \mathbf{x})$ the distortion of the j -th approximated symbol \hat{x}_j with respect to the original DCC string \mathbf{x} . $d(\mathbf{p}_{\hat{\mathbf{x}}}(j), \mathbf{x})$ is calculated as the *minimum absolute distance* between coordinate $\mathbf{p}_{\hat{\mathbf{x}}}(j)$ of the j -th edge and the segment derived from \mathbf{x} on the 2D grid:

$$d(\mathbf{p}_{\hat{\mathbf{x}}}(j), \mathbf{x}) = \min_{1 \leq i \leq N} |\mathbf{p}_{\hat{\mathbf{x}}}(j) - \mathbf{p}_{\mathbf{x}}(i)| \tag{16}$$

SSDD $D_S(\hat{\mathbf{x}}, \mathbf{x})$ is then calculated as the sum of squared distortions of all approximated symbols:

$$D_S(\hat{\mathbf{x}}, \mathbf{x}) = \sum_{j=1}^{\hat{N}} d^2(\mathbf{p}_{\hat{\mathbf{x}}}(j), \mathbf{x}) \tag{17}$$

Another distortion metric is MADD, which measures the *maximum* distortion between the original and approximated contours. MADD is suitable for applications where perceptual quality is evaluated [11, 12, 45]. Consider for example a long contour with all edges shifted to the left by one pixel. The contour shift should incur a small perceptual penalty rather than the sum of all individual edge shifts, and so MADD is more reasonable than SSDD in this case. We calculate MADD $D_M(\hat{\mathbf{x}}, \mathbf{x})$ as the maximum distortion of all the approximated symbols:

$$D_M(\hat{\mathbf{x}}, \mathbf{x}) = \max_{1 \leq j \leq \hat{N}} d(\mathbf{p}_{\hat{\mathbf{x}}}(j), \mathbf{x}) \tag{18}$$

B. SSDD based Contour Coding

To approximate contour \mathbf{x} using SSDD (17) as the distortion metric, we first write the RD cost as follows:

$$\min_{\hat{\mathbf{x}}} D_S(\hat{\mathbf{x}}, \mathbf{x}) + \lambda R(\hat{\mathbf{x}}) \tag{19}$$

where $R(\hat{\mathbf{x}})$ denotes the overhead to encode DCC string $\hat{\mathbf{x}}$ and λ is the Lagrange multiplier. $R(\hat{\mathbf{x}})$ is approximated as the total information of the symbols in $\hat{\mathbf{x}}$,

$$R(\hat{\mathbf{x}}) = - \sum_{j=1}^{\hat{N}} \log_2 P(\hat{x}_j | \hat{\mathbf{x}}_1^{j-1}) \tag{20}$$

Given a context tree \mathcal{T}^* , a context \mathbf{w} is selected for each \hat{x}_j to calculate $P(\hat{x}_j | \hat{\mathbf{x}}_1^{j-1})$, where \mathbf{w} is a prefix of $\hat{\mathbf{x}}_1^{j-1}$. Specifically, $P(\hat{x}_j | \hat{\mathbf{x}}_1^{j-1}) = P(\hat{x}_j | \mathbf{w})$, calculated as $\frac{N(\hat{x}_j \mathbf{w})}{N(\mathbf{w})}$.

Using the definitions of $D_S(\hat{\mathbf{x}}, \mathbf{x})$ and $R(\hat{\mathbf{x}})$, the objective is written as:

$$F_S(\hat{\mathbf{x}}, \mathbf{x}) = \sum_{j=1}^{\hat{N}} d^2(\mathbf{p}_{\hat{\mathbf{x}}}(j), \mathbf{x}) - \lambda \sum_{j=1}^{\hat{N}} \log_2 P(\hat{x}_j | \hat{\mathbf{x}}_1^{j-1}) \tag{21}$$

For simplicity, we assume that the 2D coordinates of the first and last edges of the approximated $\hat{\mathbf{x}}$ are the same as those of the original \mathbf{x} , i.e., $\mathbf{p}_{\hat{\mathbf{x}}}(1) = \mathbf{p}_{\mathbf{x}}(1)$ and $\mathbf{p}_{\hat{\mathbf{x}}}(\hat{N}) = \mathbf{p}_{\mathbf{x}}(N)$.

1) *Dynamic Programming Algorithm*: We now describe an efficient DP algorithm to find the optimal approximated contour $\hat{\mathbf{x}}$, minimizing (21). We first rewrite (21) as:

$$F_S(\hat{\mathbf{x}}, \mathbf{x}) = \sum_{j=1}^{\hat{N}} f(\mathbf{p}_{\hat{\mathbf{x}}}(j), \hat{\mathbf{x}}_1^j) \quad (22)$$

where

$$f(\mathbf{p}_{\hat{\mathbf{x}}}(j), \hat{\mathbf{x}}_1^j) = d^2(\mathbf{p}_{\hat{\mathbf{x}}}(j), \mathbf{x}) - \lambda \log_2 P(\hat{x}_j | \hat{\mathbf{x}}_1^{j-1}) \quad (23)$$

$f(\mathbf{p}_{\hat{\mathbf{x}}}(j), \hat{\mathbf{x}}_1^j)$ is the RD cost of symbol \hat{x}_j . Since \mathbf{x} is fixed, we omit it in the parameters of $f(\cdot)$ to simplify notations.

Denote by $C_j(\hat{\mathbf{x}}_{j-D}^{j-1}, \mathbf{p}_{\hat{\mathbf{x}}}(j-1), a_{\hat{\mathbf{x}}}(j-1))$ the minimum aggregate RD cost from \hat{x}_j to last edge $\hat{x}_{\hat{N}}$ given that the D previous symbols (called *history* in the sequel) are $\hat{\mathbf{x}}_{j-D}^{j-1}$, and the previous edge has coordinate $\mathbf{p}_{\hat{\mathbf{x}}}(j-1)$ and absolute direction $a_{\hat{\mathbf{x}}}(j-1)$. D is the maximum depth of \mathcal{T}^* . The code rate of \hat{x}_j depends on no more than its D previous symbols in history $\hat{\mathbf{x}}_{j-D}^{j-1}$.

We can calculate $C_j(\hat{\mathbf{x}}_{j-D}^{j-1}, \mathbf{p}_{\hat{\mathbf{x}}}(j-1), a_{\hat{\mathbf{x}}}(j-1))$ recursively as follows:

$$C_j(\hat{\mathbf{x}}_{j-D}^{j-1}, \mathbf{p}_{\hat{\mathbf{x}}}(j-1), a_{\hat{\mathbf{x}}}(j-1)) = \min_{\hat{x}_j \in \mathcal{D}} \begin{cases} f(\mathbf{p}_{\hat{\mathbf{x}}}(j), \hat{\mathbf{x}}_{j-D}^j), & \mathbf{p}_{\hat{\mathbf{x}}}(j) = \mathbf{p}_{\mathbf{x}}(N) \\ f(\mathbf{p}_{\hat{\mathbf{x}}}(j), \hat{\mathbf{x}}_{j-D}^j) + C_{j+1}(\hat{\mathbf{x}}_{j+1-D}^j, \mathbf{p}_{\hat{\mathbf{x}}}(j), a_{\hat{\mathbf{x}}}(j)), & \text{o.w.} \end{cases} \quad (24)$$

where $\mathbf{p}_{\hat{\mathbf{x}}}(j)$ and $a_{\hat{\mathbf{x}}}(j)$ are the coordinate and absolute direction of the j -th edge derived from chosen symbol \hat{x}_j . In words, (24) chooses the next symbol \hat{x}_j to minimize the sum of a local cost $f(\mathbf{p}_{\hat{\mathbf{x}}}(j), \hat{\mathbf{x}}_{j-D}^j)$ and a recursive cost $C_{j+1}(\cdot)$ for the remaining symbols in $\hat{\mathbf{x}}$, given updated history $\hat{\mathbf{x}}_{j+1-D}^j$ and the j -th edge. If the coordinate of the next edge matches the coordinate of the last edge of \mathbf{x} , i.e., $\mathbf{p}_{\hat{\mathbf{x}}}(j) = \mathbf{p}_{\mathbf{x}}(N)$, we terminate the recursion.

The absolute direction $a_{\hat{\mathbf{x}}}(j)$ and the coordinate $\mathbf{p}_{\hat{\mathbf{x}}}(j)$ of the next edge j are derived from the chosen next symbol \hat{x}_j . $a_{\hat{\mathbf{x}}}(j)$ is the resulting absolute direction after the previous absolute direction $a_{\hat{\mathbf{x}}}(j-1)$ proceeds in the relative direction specified by \hat{x}_j . Coordinate $\mathbf{p}_{\hat{\mathbf{x}}}(j)$ is then derived from $a_{\hat{\mathbf{x}}}(j)$ relative to the previous coordinate $\mathbf{p}_{\hat{\mathbf{x}}}(j-1)$. For example, given that the previous absolute direction is North, after turning left, i.e., $\hat{x}_j = 1$, the resulting absolute direction is West. Then $\mathbf{p}_{\hat{\mathbf{x}}}(j)$ is computed as $\mathbf{p}_{\hat{\mathbf{x}}}(j-1)$ going west by one pixel. With the updated $\mathbf{p}_{\hat{\mathbf{x}}}(j)$ and $\hat{\mathbf{x}}_{j-D}^j$, the local cost $f(\mathbf{p}_{\hat{\mathbf{x}}}(j), \hat{\mathbf{x}}_{j-D}^j)$ is computed using (23).

In practice, we restrict the approximated DCC string $\hat{\mathbf{x}}$ to be no longer than the original DCC string \mathbf{x} , i.e., $\hat{N} \leq N$, to induce a lower rate. So if $j > N$, we stop the recursion and return infinity to signal an invalid solution.

2) *Complexity Analysis*: The complexity of the DP algorithm is upper-bounded by the size of the DP table times the complexity of computing each table entry. Denote by Q the total number of possible coordinates³ $\mathbf{p}_{\hat{\mathbf{x}}}(j-1)$. Examining the

³As a computation / quality tradeoff, we can restrict the set of potential coordinates, for example, to be a set of neighborhood grid points within some fixed distance D_{\max} from the original contour segment \mathbf{x} ; e.g., the set of points in the grey area in Fig.5 within $D_{\max} = 2$ from original \mathbf{x} .

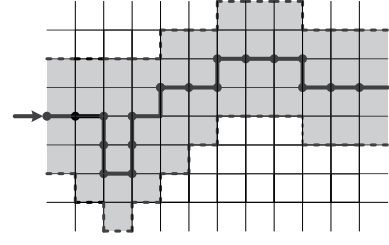


Fig. 5. An example of possible locations (region in gray) of approximated DCC string defined by D_{\max} . In this example, $D_{\max} = 2$.

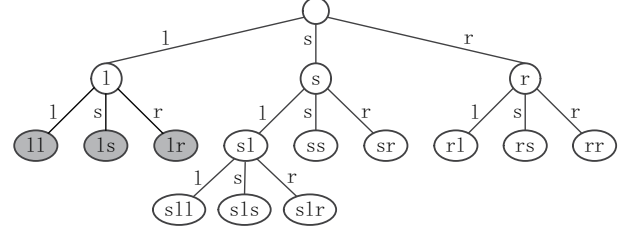


Fig. 6. An example of *total suffix tree* (TST) derived from the context tree in Fig. 3. End nodes in gray are added nodes based on the context tree. All the end nodes construct a TST: $\mathcal{T}_s^* = \{ll, ls, lr, sl, ss, sr, rl, rs, rr\}$.

subscript and three arguments of the recursive function $C_j(\cdot)$, we see that the DP table size is $N \times 3^D \times Q \times 4$, or $O(N3^D Q)$. The complexity of compute one table entry using (24) is $O(1)$. Hence the complexity of the algorithm is $O(N3^D Q)$.

3) *Total Suffix Tree (TST)*: When the training data is large, D is also large, resulting in a very large DP table size due to the exponential term 3^D . In (24) when calculating local cost $f(\mathbf{p}_{\hat{\mathbf{x}}}(j), \hat{\mathbf{x}}_{j-D}^j)$, actually the context required to compute rate is $\mathbf{w} = \hat{\mathbf{x}}_{j-|\mathbf{w}|}^{j-1}$, where the context length $|\mathbf{w}|$ is typically smaller than D because the context tree \mathcal{T}^* of maximum depth D is variable-length. Thus, if we can, at appropriate recursive calls, reduce the history from $\hat{\mathbf{x}}_{j+1-D}^j$ of length D to $\hat{\mathbf{x}}_{j+1-k}^j$ of length k , $k < D$, for recursive call to $C_{j+1}(\cdot)$ in (24), then we can reduce the DP table size and in turn the computation complexity of the DP algorithm.

The challenge is how to “remember” just enough previous symbols $\hat{x}_j, \hat{x}_{j-1}, \dots$ during recursion so that the right context \mathbf{w} can still be correctly identified to compute rate at a later recursive call. The solution to this problem can be described simply. Let \mathbf{w} be a context (end node) in context tree \mathcal{T}^* . Context \mathbf{w} must be created at some previous recursive call $C_j(\cdot)$ by concatenating a chosen j -th symbol $\hat{x}_j = w_{|\mathbf{w}|}$ with suffix $\mathbf{w}_1^{|\mathbf{w}|-1}$ of context \mathbf{w} . It implies that the recursion in (24) must remember suffix $\mathbf{w}_1^{|\mathbf{w}|-1}$ for this creation of \mathbf{w} to take place at a later recursion. To create suffix $\mathbf{w}_1^{|\mathbf{w}|-1}$ at a later recursive call, one must remember its suffix $\mathbf{w}_1^{|\mathbf{w}|-2}$ at an earlier call. We can thus generalize this observation and state that *a necessary and sufficient condition to preserve all contexts \mathbf{w} in context tree \mathcal{T}^* is to remember all suffixes of \mathbf{w} during the recursion.*

All suffixes of contexts in \mathcal{T}^* can themselves be represented as a tree, which we call a *total suffix tree* (TST), denoted as

\mathcal{T}_s^* . By definition, \mathcal{T}^* is a sub-tree of \mathcal{T}_s^* . Further, TST \mathcal{T}_s^* is also a full tree given \mathcal{T}^* is a full tree; \mathcal{T}_s^* is essentially a union of all sub-trees of \mathcal{T}^* , and a sub-tree of a full tree is also a full tree. \mathcal{T}^* has $O(K)$ contexts, each of maximum length D . Each context can induce $O(D)$ additional end nodes in TST \mathcal{T}_s^* . Hence TST \mathcal{T}_s^* has $O(KD)$ end-nodes.

Fig. 6 illustrates one example of TST derived from the context tree shown in Fig. 3. TST \mathcal{T}_s^* can be used for compact DP table entry indexing during recursion (24) as follows. When an updated history $\hat{\mathbf{x}}_{j+1-D}^j$ is created from a selection of symbol \hat{x}_j , we first truncate $\hat{\mathbf{x}}_{j+1-D}^j$ to $\hat{\mathbf{x}}_{j+1-k}^j$, where $\hat{\mathbf{x}}_{j+1-k}^j$ is the longest matching string in \mathcal{T}_s^* from root node down. Because TST \mathcal{T}_s^* is a full tree, the longest matching string always corresponds to an end node. The shortened history $\hat{\mathbf{x}}_{j+1-k}^j$ is then used as the new argument for the recursive call. Practically, it means that only DP table entries of arguments $\hat{\mathbf{x}}_{j+1-k}^j$ that are end nodes of TST \mathcal{T}_s^* will be indexed, thus reducing complexity from original $O(N3^DQ)$ to $O(NKDQ)$, which is now polynomial in D .

C. MADD based Contour Coding

When the distortion metric is MADD, instead of an unconstrained Lagrangian formulation, we formulate instead a distortion-constrained problem as follows:

$$\begin{aligned} \min_{\hat{\mathbf{x}}} \quad & R(\hat{\mathbf{x}}) \\ \text{s.t.} \quad & D_M(\hat{\mathbf{x}}, \mathbf{x}) \leq D_{\max} \end{aligned} \quad (25)$$

where D_{\max} is the maximum distortion permitted. Example when $D_{\max} = 2$ is shown in Fig. 5. D_{\max} can be varied to induce different RD tradeoff.

Given the definition of rate and distortion, (25) can be rewritten as:

$$\begin{aligned} \min_{\hat{\mathbf{x}}} \quad & -\sum_{j=1}^{\hat{N}} \log_2 P(\hat{x}_j | \hat{\mathbf{x}}_1^{j-1}) \\ \text{s.t.} \quad & \max_{1 \leq j \leq \hat{N}} d(\mathbf{p}_{\hat{\mathbf{x}}}(j), \mathbf{x}) \leq D_{\max} \end{aligned} \quad (26)$$

Similar to the SSDD case, this minimization problem can also be solved by using DP by simplifying the problem to:

$$\begin{aligned} \min_{\hat{\mathbf{x}}} \quad & \sum_{j=1}^{\hat{N}} r(\hat{\mathbf{x}}_1^j) \\ \text{s.t.} \quad & r(\hat{\mathbf{x}}_1^j) = -\log_2 P(\hat{x}_j | \hat{\mathbf{x}}_1^{j-1}) \\ & d(\mathbf{p}_{\hat{\mathbf{x}}}(j), \mathbf{x}) \leq D_{\max} \end{aligned} \quad (27)$$

where $r(\hat{\mathbf{x}}_1^j)$ denotes the coding cost of \hat{x}_j . The problem becomes finding an approximated DCC string $\hat{\mathbf{x}}$ in the region \mathcal{R} restricted by D_{\max} in order to minimize the total rate.

Denote by $C'_j(\hat{\mathbf{x}}_{j-D}^{j-1}, \mathbf{p}_{\hat{\mathbf{x}}}(j-1), a_{\hat{\mathbf{x}}}(j-1))$ the minimum total rate from \hat{x}_j to $\hat{x}_{\hat{N}}$ given the D previous symbols $\hat{\mathbf{x}}_{j-D}^{j-1}$, and the previous edge has coordinate $\mathbf{p}_{\hat{\mathbf{x}}}(j-1)$ and absolute direction $a_{\hat{\mathbf{x}}}(j-1)$. We can compute $C'_j(\hat{\mathbf{x}}_{j-D}^{j-1}, \mathbf{p}_{\hat{\mathbf{x}}}(j-1), a_{\hat{\mathbf{x}}}(j-1))$ recursively as follows:

$$\begin{aligned} C'_j(\hat{\mathbf{x}}_{j-D}^{j-1}, \mathbf{p}_{\hat{\mathbf{x}}}(j-1), a_{\hat{\mathbf{x}}}(j-1)) = \\ \min_{\hat{x}_j \in \mathcal{D}} \begin{cases} r(\hat{\mathbf{x}}_{j-D}^j), & \mathbf{p}_{\hat{\mathbf{x}}}(j) = \mathbf{p}_{\mathbf{x}}(N) \\ r(\hat{\mathbf{x}}_{j-D}^j) \\ + C'_{j+1}(\hat{\mathbf{x}}_{j+1-D}^j, \mathbf{p}_{\hat{\mathbf{x}}}(j), a_{\hat{\mathbf{x}}}(j)), & \text{o.w.} \end{cases} \end{aligned} \quad (28)$$

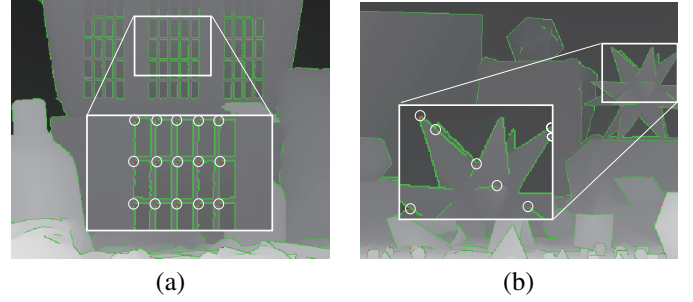


Fig. 7. Examples of starting points. Green pixels are the pixels along one side of the detected edges and red pixels (inside white circles) are the starting points. (a) Laundry. (b) Moebius.

where we restrict \hat{x}_j to induce only edges that are within the feasible region \mathcal{R} delimited by D_{\max} . The recursion is same as (24) except the local cost function. Hence the complexity of the DP algorithm here is also same as in the SSDD case.

V. STARTING POINT CODING

We propose a mixed-Golomb (M-Golomb) algorithm to encode the 2D coordinates of starting points of all contours in the target image. In general, the starting points are not uniformly distributed on the image. They tend to cluster around objects or figures that appear in the scene, as illustrated in Fig. 7. This means that the differences in coordinates of neighboring starting points tend to be small, and Golomb coding [46] is suitable to encode coordinate differences that are more likely small than large.

A 2D coordinate of a starting point has two components (horizontal and vertical); we use Golomb coding to code only the differences in coordinates in one component. Specifically, we first sort the starting points according to this chosen component in ascending order. The coordinate differences in neighboring points are then coded using Golomb coding. The coordinates in the other component are coded using fixed length binary coding.

Golomb coding uses a tunable parameter to divide an input value into two parts: the remainder and the quotient. The quotient is sent in unary coding, followed by the remainder in truncated binary coding. For simplicity, we choose the parameter as 2^k , where $0 \leq k \leq \lceil \log_2 W \rceil$ and W is the maximum input value. We examine all possible values of k to find the smallest coding cost of all the starting points in the target image and send this selected parameter as side information.

VI. EXPERIMENTAL RESULTS

We evaluate the performance of our lossless and lossy contour coding algorithms in three scenarios. For scenario 1, we first present results for lossless contour coding of VOPs in MPEG4 sequences, then results for lossless contour coding of objects in depth images and starting points coding in order. For scenario 2, we present visual comparisons of lossy contour coding of objects in depth images, then show the RD performance of lossy depth image coding. For scenario

3, we present visual comparisons of different silhouette approximating methods, then show the RD performance of lossy multiview silhouette coding for 3D reconstruction.

A. Lossless Contour Coding

To evaluate the performance of our proposed context tree based lossless contour coding (CT-LL), we used VOPs of four MPEG4 sequences⁴, Children, Stefan, Robot and Cyc, as test data. The spatial resolution of all four sequences is 352×240 . 10 frames were tested for each sequence. Note that the contours of the VOPs were already outlined as input to our coding algorithm. We also coded object contours in four Middlebury depth images⁵: Moebius (456×368), Cones (448×368), Teddy (448×368) and Laundry (440×368). The contours were detected using a gradient-based edge detection scheme in [7].

To code VOPs in a given video frame of a MPEG-4 sequence, previous two frames of the same sequence were used to train the context tree. To code object contours of a Middlebury depth image, we used other Middlebury images to train the context tree. We tested two training image sets (Train1 and Train2), where each contains four different randomly selected images. To test the performance of our proposed CT-LL with different training sets, we used also a combined training image set (Train1+Train2), which contains images in both Train1 and Train2. We set the parameter $\alpha = 0.25$ in all the experiments.

We compared CT-LL against four lossless compression schemes: i) the Lempel-Ziv-Welch (LZW) algorithm in [47], an improved version of LZ78; ii) the probability suffix tree (PST) algorithm in [16], iii) the prediction by partial matching (PPM) algorithm in [48], and iv) the arithmetic edge coding (AEC) scheme in [7]. The training datasets for PST and PPM were the same as those for CT-LL. Our proposed starting points coding scheme (M-Golomb) was compared against fixed length binary coding (Binary). All the contour compression methods used our proposed M-Golomb for coding the starting points in the experiments.

TABLE II
RESULTS OF LOSSLESS CONTOUR CODING OF VOPs IN BITS PER SYMBOL

Bits/Symbol	LZW	AEC	PST	PPM	CT-LL
Children	1.685	1.256	1.176	1.257	1.170
Stefan	1.494	1.117	0.914	0.956	0.894
Robot	1.808	1.414	1.341	1.318	1.278
Cyc	1.525	1.182	0.879	0.823	0.828
Average	1.628	1.242	1.078	1.089	1.043

1) *Results for Lossless Contour Coding of VOPs in MPEG4 Sequences*: Table II shows the compression results in average bits per symbol for VOP lossless coding using different methods. Compared to the other methods, our proposed CT-LL achieves noticeable bitrate reduction on average. Specifically, on average we reduce the bitrate by 35.97% compared to

LZW, 16.08% compared to AEC, 3.26% compared to PST, and 4.23% compared to PPM.

In general, the context based methods (PST, PPM and CT-LL) outperform LZW and AEC significantly. Compared to PPM, the gain of our CT-LL varies depending on different sequences. In particular, for Children and Stefan with large motion, CT-LL outperforms PPM by 6.92% and 6.49%. While for Cyc with very small motion, CT-LL is worse than PPM marginally by 0.60%. Since there is no prior to prune the contexts in PPM, it fails to avoid overfitting for sequences with large motion. Compared to PST, which can avoid overfitting by setting five application-specific thresholds, CT-LL has steady gain because of the our proposed geometric prior for contour coding.

2) *Results for Lossless Contour Coding of Objects in Middlebury Depth Images*: Table III shows the compression results for lossless contour coding of objects in depth images. Our CT-LL outperforms the other four methods for all three training sets. Specifically, for Train1, the average bit reductions are 30.96%, 10.35%, 7.89% and 2.45% over LZW, AEC, PST and PPM, respectively; for Train2, the average bit reductions are 30.93%, 10.30%, 7.76% and 3.31%; for Train1+Train2, the average bit reductions are 31.74%, 11.36%, 6.62% and 2.49%.

Compared to the results of lossless contour coding of VOPs, the performance of CT-LL of lossless contour coding of objects in depth images decreased a bit. The difference mainly stems from the dissimilarity in statistics between the test and training images in the latter case. Specifically, for coding VOPs in MPEG4 video sequences, the training images are the previous frames, which are quite similar to the current frame. While for the test depth images, the training images are randomly selected images, which were obtained using the same setup but from different 3D scenes. Nonetheless, as shown in Table III, we can still achieve bitrate reduction compared to other methods using different training sets.

Comparing the results of Train1+Train2 to that of Train1 and Train2, we can see that all three context based methods (PST, PPM and CT-LL) benefited from more training data, resulting in lower average bitrates. With twice the training data, PST and PPM resulted in a larger improvement compared to CT-LL. This demonstrates that the performance of our proposed CT-LL is stable for large and small data size. In other words, unlike other context based methods like PST and PPM, our CT-LL maintains good coding performance even for small data size by avoiding overfitting using our proposed geometric prior.

3) *Results for Starting Points Coding*: The performance of our proposed starting points coding scheme (M-Golomb) compared to the fixed binary coding (Binary) is shown in Table IV. We tested both the VOPs in MPEG4 sequences and the object contours in depth images. Both the number of contours and the bit overhead in coding contours of VOPs are averaged to one frame to better compare to that of the depth images. For the VOPs which contain only a few contours in one frame, i.e., Children and Stefan, the proposed M-Golomb has little advantage over Binary. However, we can save on average 29.56% bits of starting points and 6.94%

⁴http://ftp.tnt.uni-hannover.de/pub/MPEG/mpeg4_masks/

⁵<http://vision.middlebury.edu/stereo/data/>

TABLE III
RESULTS OF LOSSLESS CODING CONTOURS OF OBJECTS IN DEPTH IMAGES IN BITS PER SYMBOL

Bits/Symbol	LZW	AEC	Train1			Train2			Train1+Train2		
			PST	PPM	CT-LL	PST	PPM	CT-LL	PST	PPM	CT-LL
Moebius	1.912	1.409	1.378	1.272	1.233	1.373	1.286	1.240	1.326	1.257	1.220
Cones	1.727	1.424	1.245	1.181	1.164	1.282	1.232	1.199	1.241	1.176	1.154
Teddy	2.014	1.519	1.555	1.493	1.450	1.550	1.493	1.434	1.510	1.479	1.439
Laundry	1.642	1.265	1.290	1.216	1.189	1.257	1.199	1.165	1.255	1.195	1.166
Average	1.824	1.404	1.367	1.291	1.259	1.366	1.303	1.260	1.333	1.277	1.245

TABLE IV
RESULTS OF CODING STARTING POINTS

	Input	Average Num of Contours	Bits of Starting Points			Total Bits		
			Binary	M-Golomb	Δ Bits	Binary	M-Golomb	Δ Bits
Mask Sequences	Children	3.4	57.8	58.7	1.56%	1684.7	1685.6	0.05%
	Stefan	1	17	21	23.53%	517.1	521.1	0.77%
	Robot	21.3	362.1	273.8	-24.39%	2474.1	2385.8	-3.57%
	Cyc	21.6	367.2	295.5	-19.53%	2959.5	2887.8	-2.42%
	Average	11.8	201.0	162.3	-19.29%	1908.9	1870.1	-2.03%
Depth Images	Moebius	144	2592	1785	-31.13%	8825	8018	-9.14%
	Cones	82	1476	1081	-26.76%	7399	7004	-5.34%
	Teddy	100	1800	1289	-28.39%	7894	7467	-5.41%
	Laundry	125	2250	1563	-30.53%	9254	8567	-7.42%
	Average	112.8	2029.5	1429.5	-29.56%	8343.0	7764.0	-6.94%

bits of total bits for the depth images which contain lots of contours in one image.

B. Depth Image Coding

We implemented our proposed SSDD based lossy contour coding (CT-SSDD) scheme for coding edges as side information in MR-GFT [4], one of the state-of-the-art depth image compression methods. In [4], the detected edges of the whole image are losslessly coded by employing AEC [7] and transmitted to the decoder for directional intra prediction. We replace AEC with CT-SSDD to compress four Middlebury depth images which are same as that in section VI-A. Train2 are used as the training images.

Different from the losslessly edge coding in MR-GFT, we compressed contours lossily using our proposed CT-SSDD. Changes in the contours can lower the edge coding cost, but can also lead to larger intra prediction residual. We thus augmented the depth image to match the approximated edges obtained using CT-SSDD, in order to reduce the prediction residual. Specifically, the pixel values between the original edge and the approximated edge were replaced with the pixel values on the other side of the approximated edge, as shown on Fig. 8.

Since the contours are coded as side information for compressing depth images, the distortion term in CT-SSDD should be related to the distortion of the depth signal. Distortion of our described depth image coding scheme comes from two sources: i) distortion due to quantization of the transform coefficients, and ii) distortion due to depth pixel augmentation to suit approximated contours, as described previously. Because the contour coding and the residual coding in MR-GFT are performed in separate steps, it is impractical to consider both distortions simultaneously.

As a straightforward attempt, the distortion term $D_S(\hat{x}, x)$ in (17) is modified and defined as the augmentation distortion,

which is calculated as the sum of squared distortion of augmented depth pixels instead of the approximated symbols. Further, we choose $\lambda = 0.85 \times (2^{\frac{QP-12}{3}})$, which is the same λ as used in the mode selection part of MR-GFT, where QP is the quantization parameter. Thus this selection of λ provides an appropriate weight between the edge coding and the residual coding.

We compare the depth image coding performance of our proposed MR-GFT+CT-SSDD against MR-GFT+CT-LL, MR-GFT+PPM, the original MR-GFT in [4] and HEVC Test Model HM-16.0⁶ (HEVC), where MR-GFT+CT-LL and MR-GFT+PPM are obtained by replacing the edge coding scheme (AEC) in MR-GFT with the proposed CT-LL and PPM.

1) Edge Approximation and Depth Image Augmentation:

Fig. 8 shows two examples of edge approximation and depth image augmentation. We see that the changed edges are mostly the irregular edges (e.g., the white region in Fig. 8). This is because these irregular edges along with their contexts have more scattered probability distributions in the trained context tree \mathcal{T}^* , which will consume larger amount of bits after arithmetic coding. As shown in Fig. 8, after approximating the edges, the depth pixels are augmented to match the new edges.

Note that edges with smaller gradient (e.g., mesh grid of basket in Fig. 8(b) and flower in Fig. 8(d)) are more likely to be approximated than edges with larger gradient (e.g., the boundaries of basket in Fig. 8(b) and Teddy bear in Fig. 8(d)). This is because approximation of the edges with larger gradient will result in larger augmentation distortion.

2) Results in RD Performance: The RD performance of the proposed MR-GFT+CT-SSDD and MR-GFT+CT-LL against MR-GFT+PPM, MR-GFT and HEVC is presented in Fig. 9. The proposed MR-GFT+CT-SSDD achieves promising bit rate reduction over a wide range of PSNR, and the proposed

⁶https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.0/

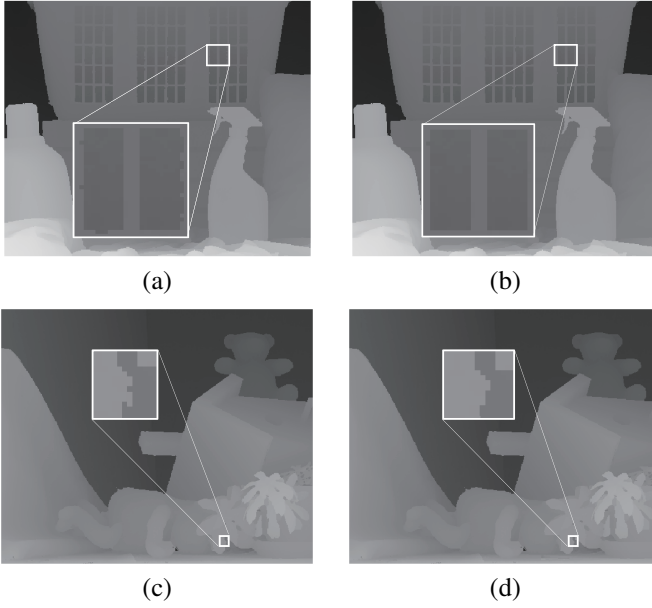


Fig. 8. Examples of edge approximation and depth image augmentation. (a) Original depth image of Laundry. (b) Augmented depth image of Laundry after edge approximation with $QP = 37$. (c) Original depth image of Teddy. (d) Augmented depth image of Teddy after edge approximation with $QP = 37$.

TABLE V
RESULTS OF BIT ALLOCATION FOR CODING TEDDY

QP	MR-GFT			MR-GFT CT-LL			MR-GFT CT-SSDD		
	Edge	Resi	Total	Edge	Resi	Total	Edge	Resi	Total
22	6515	12132	18647	6178	12132	18310	6160	12132	18292
27	6515	7857	14372	6178	7857	14035	6101	7854	13955
32	6515	6225	12740	6178	6225	12403	5827	6021	11848
37	6515	5344	11859	6178	5344	11522	5548	5322	10870

MR-GFT+CT-LL also outperforms other three schemes. On average, we achieve a bit rate reduction of 19.34% over HEVC, 10.29% over MG-GFT, 5.02% over MR-GFT+PPM and 3.46% over MR-GFT+CT-LL.

Compared to MR-GFT, the gain of the proposed two schemes comes from the more efficient edge coding schemes. Table V gives the detailed results of bits of coding edges among the total bits with different QP of Teddy. When $QP = 37$, we save on average 14.84% bits and 5.17% bits for coding edges with the proposed CT-SSDD and CT-LL compared to AEC adopted in MR-GFT. Note that the bits of coding residuals (difference between total bits and bits of coding edges) of MR-GFT+CT-LL and MR-GFT are the same, while they are a bit larger than that of MR-GFT+CT-SSDD. Using MR-GFT+CT-SSDD, the depth image is augmented with fewer irregular edges before coding, resulting in more efficient directional intra prediction.

The bit rate reduction of CT-SSDD over CT-LL becomes larger when the QP increases. This is also reflected in the RD curve in Fig. 9. As previously discussed, the strength of the approximation is controlled by λ . When QP is small, λ is also small which makes the edges less likely be changed. In other words, in high bit rate situation, it is unnecessary to approximate the edges, while more edges can be changed in

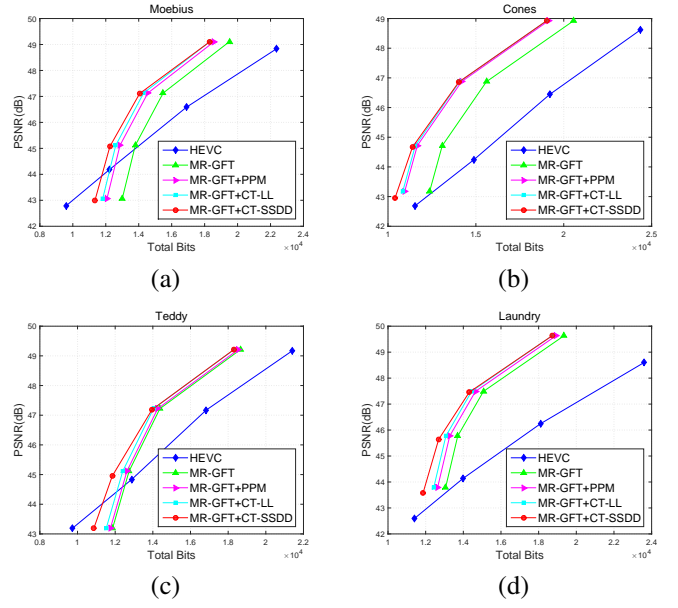


Fig. 9. RD performance comparison among different compression schemes for depth images.

the low bit rate situation.

C. Multiview Silhouettes Coding for 3D Reconstruction

We test our proposed MADD based lossy contour coding CT-MADD on the multiview silhouette sequences from Microsoft Research. The sequences are obtained using the equipment set up in [49]. The spatial resolution of the silhouettes is 384×512 . Each sequence contains 10 frames, and eight views are taken for each frame. Fig. 10(a) shows an example of the silhouettes of eight views for a single frame. The silhouettes were extracted from the images captured with an eight camera rig, as illustrated in Fig. 10(b). With the extracted silhouettes, the 3D model can be reconstructed as shown in Fig. 10(c). We coded four multiview silhouette sequences using CT-MADD and reconstructed the 3D models with the coded silhouettes. The RD performance was evaluated, where distortion is the volume error between the 3D model reconstructed with the original silhouettes and that with the coded silhouettes.

The 3D model was reconstructed based on a volume intersection scheme [50]. We projected each silhouette onto an initialized 3D volume and the final 3D model is the intersected volume of the eight projected silhouettes.

We compare the RD performance of CT-MADD against two lossy contour compression schemes: the operational rate-distortion optimal polygon-based shape coding (ORD) in [9] and the improved version based on an adaptive edge coding scheme (EA-ORD) in [12]. ORD and EA-ORD also adopt the MADD measure same as the proposed CT-MADD. For all three methods, D_{\max} was set from 1 to 5 to test different distortion.

1) *Comparison of Contour Approximation:* Fig. 11 shows one view of silhouette approximation using different methods with $D_{\max} = 1$ and $D_{\max} = 3$. With the same D_{\max} , CT-MADD consumes the fewest bits to code the silhouette. As for subjective quality, the silhouettes approximated by CT-MADD are most similar visually to the original silhouettes,

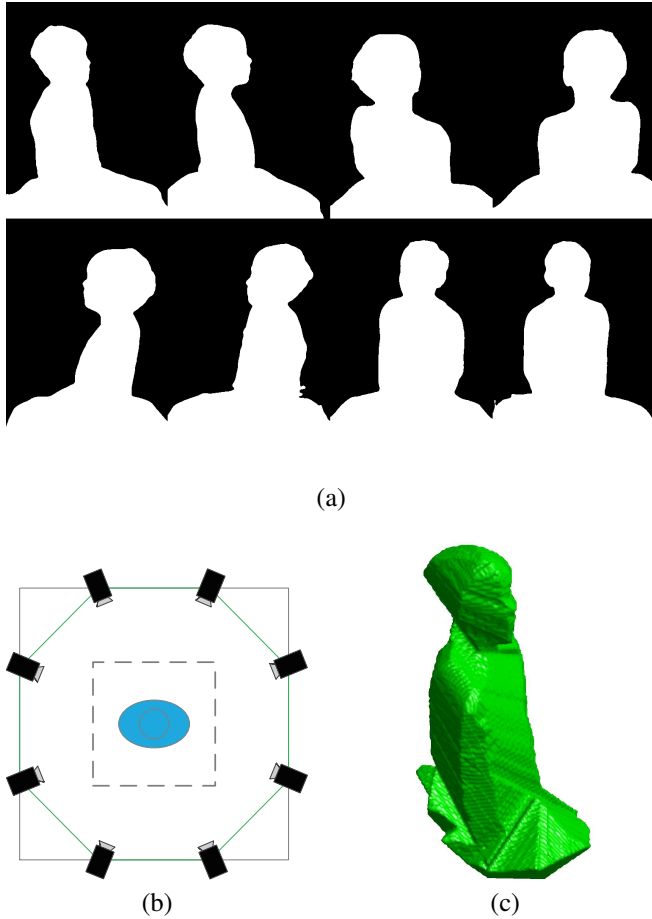


Fig. 10. (a) Silhouettes of eight views for a single frame. (b) Top views of the capture rig layout showing 8 cameras and human subject within the cube. (c) One view of a model reconstructed from the silhouettes in (a).

while the results by ORD and EA-ORD contain lots of staircase shapes.

In both ORD and EA-ORD, only some contour pixels are selected for coding. The approximated silhouette was constructed by connecting these selected contour pixels in order, making the result unnatural with too many staircase lines.

2) *Evaluation of RD Performance:* The RD performance is shown in Fig. 12. We observe that CT-MADD outperforms ORD and EA-ORD significantly both in bit rate and distortion. Specifically, at the same D_{\max} , CT-MADD has an average bitrate reduction of 50% and 38% and an average of distortion reduction of 31% and 43% compared to ORD and EA-ORD.

In ORD and EA-ORD, the differences between two selected contour pixels (vertices) are coded using unary code. Larger differences will consume more bits. Although CT-MADD codes more number of contour pixels, the accurately estimated conditional probabilities enable our contour pixels to be coded much more efficiently than ORD and EA-ORD.

VII. CONCLUSION

We investigate the problem of lossless and lossy coding of image contours, focusing on the case when the sizes of training datasets \mathcal{X} are limited. To avoid overfitting, we propose a *maximum a posteriori* (MAP) formulation to compute an

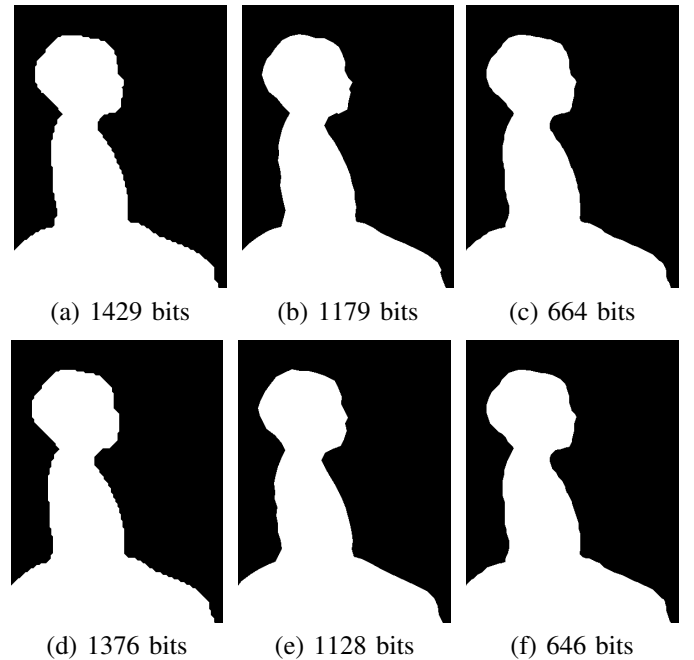


Fig. 11. Examples of silhouette approximation of different methods. The original silhouette is the second view in Fig. 10(a). Left column: ORD. Middle column: EA-ORD. Right column: CT-MADD. (a) ~ (c) $D_{\max} = 1$. (d) ~ (f) $D_{\max} = 3$.

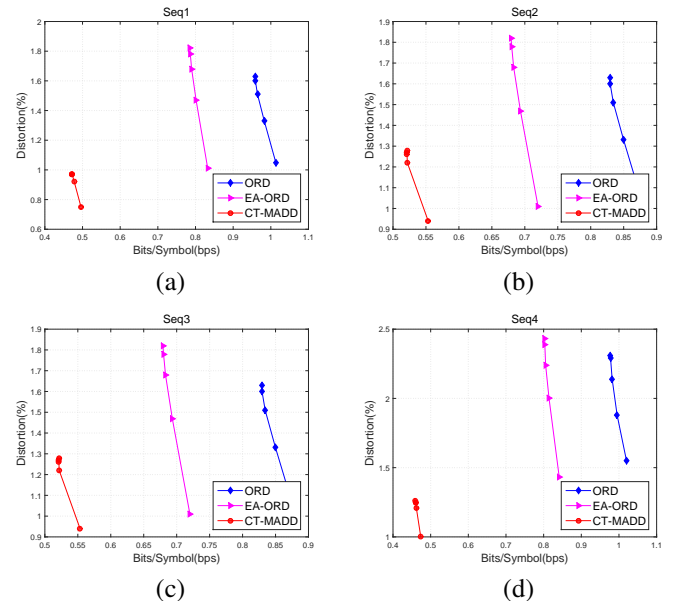


Fig. 12. RD performance comparison among different compression schemes for multiview silhouettes coding.

optimal variable-length context tree (VCT) \mathcal{T} —one that has both large likelihood $P(\mathcal{X}|\mathcal{T})$ that can explain observed data \mathcal{X} , and large prior $P(\mathcal{T})$ stating that contours are more likely straight than curvy. For the lossy case, we develop dynamic programming (DP) algorithms that approximate a detected contour by trading off coding rate with two different notions of distortion. The complexity of the DP algorithms can be reduced via compact table entry indexing using a total suffix tree (TST) derived from VCT \mathcal{T} . Experimental results show that our proposed lossless and lossy algorithms outperform

state-of-the-art coding schemes consistently for both small and large training datasets.

REFERENCES

- [1] C. Grigorescu, N. Petkov and M. A. Westenberg, "Contour detection based on nonclassical receptive field inhibition," *IEEE Trans. Image Process.*, vol. 12, no. 7, pp. 729–739, 2003.
- [2] G. Shen, W.-S. Kim, S.K. Narang, A. Ortega, J. Lee, and H. Wey, "Edge-adaptive transforms for efficient depth map coding," in *IEEE Picture Coding Symposium*, Nagoya, Japan, December 2010.
- [3] W. Hu, G. Cheung, X. Li, and O. Au, "Depth map compression using multi-resolution graph-based transform for depth-image-based rendering," in *IEEE International Conference on Image Processing*, Orlando, FL, September 2012.
- [4] W. Hu, G. Cheung, A. Ortega, and O. Au, "Multi-resolution graph Fourier transform for compression of piecewise smooth images," in *IEEE Transactions on Image Processing*, January 2015, vol. 24, no.1, pp. 419–433.
- [5] W. Hu, G. Cheung, and A. Ortega, "Intra-prediction and generalized graph Fourier transform for image coding," in *IEEE Signal Processing Letters*, November 2015, vol. 22, no.11, pp. 1913–1917.
- [6] I. Daribo, D. Florencio, and G. Cheung, "Arbitrarily shaped sub-block motion prediction in texture map compression using depth information," in *2012 Picture Coding Symposium*, Krakow, Poland, May 2012.
- [7] I. Daribo, D. Florencio and G. Cheung, "Arbitrarily shaped motion prediction for depth video compression using arithmetic edge coding," *IEEE Trans. Image Process.*, vol. 23, no. 11, pp. 4696–4708, Nov. 2014.
- [8] D. Weinland, R. Ronfard and E. Boyer, "A survey of vision-based methods for action representation, segmentation and recognition," *Computer Vision and Image Understanding*, vol. 115, no. 2, pp. 224–241, 2011.
- [9] A. K. Katsaggelos, L. P. Kondi, F. W. Meier, W. Fabian, J. O. Ostermann and G. M. Schuster, "MPEG-4 and rate-distortion-based shape-coding techniques," *Proceedings of the IEEE*, vol. 86, no. 6, pp. 1126–1154, 1998.
- [10] F. Sohel, G. C. Karmakar, L. S. Dooley and M. Bennamoun, "Sliding-window designs for vertex-based shape coding," *IEEE Trans. Multimedia*, vol. 14, no. 3, pp. 683–692, 2012.
- [11] Z. Lai, J. Zhu, Z. Ren, W. Liu, and B. Yan, "Arbitrary directional edge encoding schemes for the operational rate-distortion optimal shape coding framework," in *Proc. Conf. Data Compression*. IEEE, 2010, pp. 20–29.
- [12] J. H. Zhu, Z. Y. Lai, W. Y. Liu and J. B. Luo, "Adaptive edge encoding schemes for the rate-distortion optimal polygon-based shape coding," in *Proc. Conf. Data Compression*, 2014, pp. 103–112.
- [13] R. El-Yaniv R. Begleiter and G. Yona, "On prediction using variable order markov models," *Journal of Artificial Intelligence Research*, pp. 385–421, 2004.
- [14] J. Rissanen, "A universal data compression system," *IEEE Trans. Information Theory*, vol. 29, no. 5, pp. 656–664, 1983.
- [15] J. Cleary and I. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Trans. Commun.*, vol. 32, no. 4, pp. 396–402, 1984.
- [16] D. Ron, Y. Singer and N. Tishby, "The power of amnesia: Learning probabilistic automata with variable memory length," *Machine learning*, vol. 25, no. 2-3, pp. 117–149, 1996.
- [17] C. C. Lu and J. G. Dunham, "Highly efficient coding schemes for contour lines based on chain code representations," *IEEE Trans. Commun.*, vol. 39, no. 10, pp. 1511–1514, 1991.
- [18] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, Anthony D and R. Katz, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
- [19] R. W. Katz, "On some criteria for estimating the order of a markov chain," *Technometrics*, vol. 23, no. 3, pp. 243–249, 1981.
- [20] P. Vandewalle, J. Kovačević and M. Vetterli, "Reproducible research in signal processing," *IEEE Signal Processing Mag.*, vol. 26, no. 3, pp. 37–47, 2009.
- [21] H. Freeman, "Application of the generalized chain coding scheme to map data processing," 1978, pp. 220–226.
- [22] Y. Liu and B. Žalik, "An efficient chain code with huffman coding," *Pattern Recognition*, vol. 38, no. 4, pp. 553–557, 2005.
- [23] Y. H. Chan and W. C. Siu, "Highly efficient coding schemes for contour line drawings," in *Proc. IEEE Int. Conf. Image Process.* IEEE, 1995, pp. 424–427.
- [24] R. Estes and R. Algazi, "Efficient error free chain coding of binary documents," in *Proc. Conf. Data Compression*, 1995, p. 122.
- [25] M. J. Turner and N. E. Wiseman, "Efficient lossless image contour coding," in *Computer Graphics Forum*, 1996, vol. 15, no.2, pp. 107–117.
- [26] O. Egger, F. Bossen and T. Ebrahimi, "Region based coding scheme with scalability features," in *Proc. Conf. VIII European Signal Processing*. IEEE, 1996, vol. 2, LTS-CONF-1996-062, pp. 747–750.
- [27] C. Jordan, S. Bhattacharjee, F. Bossen, F. Jordan, and T. Ebrahimi, "Shape representation and coding of visual objects in multimedia applications an overview," *Annals of Telecommunications*, vol. 53, no. 5, pp. 164–178, 1998.
- [28] H. Freeman, "On the encoding of arbitrary geometric configurations," *IRE Trans. Electronic Computers*, , no. 2, pp. 260–268, Jun 1961.
- [29] I. Daribo, G. Cheung and D. Florencio, "Arithmetic edge coding for arbitrarily shaped sub-block motion prediction in depth video coding," in *IEEE International Conference on Image Processing*, Orlando, FL, September 2012.
- [30] T. Kaneko and M. Okudaira, "Encoding of arbitrary curves based on the chain code representation," *IEEE Trans. Commun.*, vol. 33, no. 7, pp. 697–707, 1985.
- [31] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.
- [32] P. Bühlmann and A. J. Wyner, "Variable length markov chains," *The Annals of Statistics*, vol. 27, no. 2, pp. 480–513, 1999.
- [33] G. Bejerano and G. Yona, "Variations on probabilistic suffix trees: statistical modeling and prediction of protein families," *Bioinformatics*, vol. 17, no. 1, pp. 23–43, 2001.
- [34] M. C. Yeh, Y. L. Huang and J. S. Wang, "Scalable ideal-segmented chain coding," in *Proc. IEEE Int. Conf. Image Process.*, 2002, vol. 1, pp. 194–197.
- [35] S. Zahir, K. Dhou and B. Prince George, "A new chain coding based method for binary image compression and reconstruction," 2007, pp. 1321–1324.
- [36] J. Chung, J. Lee, J. Moon and J. Kim, "A new vertex-based binary shape coder for high coding efficiency," *Signal processing: image communication*, vol. 15, no. 7, pp. 665–684, 2000.
- [37] C. Kuo, C. Hsieh and Y. Huang, "A new adaptive vertex-based binary shape coding technique," *Image and vision computing*, vol. 25, no. 6, pp. 863–872, 2007.
- [38] G. M. Schuster and A. K. Katsaggelos, "An optimal segmentation encoding scheme in the rate distortion sense," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 1996, vol. 2, pp. 640–643.
- [39] G. M. Schuster, G. Melnikov and A. K. Katsaggelos, "Operationally optimal vertex-based shape coding," *IEEE Signal Processing Mag.*, vol. 15, no. 6, pp. 91–108, 1998.
- [40] F. A. Sohel, L. S. Dooley and G. C. Karmakar, "Variable width admissible control point band for vertex based operational-rate-distortion optimal shape coding algorithms," in *Proc. IEEE Int. Conf. Image Process.* IEEE, 2006, pp. 2461–2464.
- [41] F. A. Sohel, L. S. Dooley and G. C. Karmakar, "New dynamic enhancements to the vertex-based rate-distortion optimal shape coding framework," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 10, pp. 1408–1413, 2007.
- [42] S. Alcaraz-Corona and R. Rodríguez-Dagnino, "Bi-level image compression estimating the markov order of dependencies," *IEEE Journal of Selected Topics in Signal Processing*, vol. 4, no. 3, pp. 605–611, 2010.
- [43] A. Akimov, A. Kolesnikov and P. Fränti, "Lossless compression of map contours by context tree modeling of chain codes," *Pattern Recognition*, vol. 40, no. 3, pp. 944–952, 2007.
- [44] S. Kullback, "Letter to the editor: the kullback-leibler distance," 1987.
- [45] D. Neuhoff and K. G. Castor, "A rate and distortion analysis of chain codes for line drawings," *IEEE Trans. Information Theory*, vol. 31, no. 1, pp. 53–68, 1985.
- [46] S. W. Golomb, "Run-length encodings," *IEEE Trans. Information Theory*, 1966.
- [47] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 6, no. 17, pp. 8–19, 1984.
- [48] A. Moffat, "Implementing the ppm data compression scheme," *IEEE Trans. Communications*, vol. 38, no. 11, pp. 1917–1921, 1990.
- [49] C. Loop, C. Zhang and Z. Y. Zhang, "Real-time high-resolution sparse voxelization with application to image-based modeling," in *Proc. ACM Conf. High-Performance Graphics*, 2013, pp. 73–79.
- [50] W. N. Martin and J. K. Aggarwal, "Volumetric descriptions of objects from multiple views," *IEEE Trans. Pattern Analysis and Machine Intelligence*, , no. 2, pp. 150–158, 1983.