JBEAM: Multiscale Curve Coding via Beamlets

Xiaoming Huo and Jihong Chen

Abstract—**A multiscale coder for curves and boundaries is presented. It utilizes a multiscale structure—beamlets—that is designed primarily for linear and curvilinear features. The coder is composed of three main components: 1) a rate-distortion optimized beamlet-based representation, 2) a tree-based coding from a beamlet representation to a symbol stream, and 3) an entropy coder. This coder is named "JBEAM." Taking advantage of its multiscale property, we utilized tree-based coding to make it progressive. The derived coder has a low order of computational complexity. Simulations demonstrate an advantage over the state-of-the-art industrial standard: JBIG 2. A software package, which includes an implementation of JBEAM, is made available. Variations and potential improvements of this method will be discussed. This work may inspire more activities in this line of research, improving curve coding.**

Index Terms—**Beamlet, curve coding, tree-based coding.**

I. INTRODUCTION

COMPRESSING and coding linear features has a long history. These works are usually under the topics of *contour coding* and *shape coding*. Shape coding has many important applications in modern multimedia signal processing. It has been used in content-based video coding. An editorial [10] and several papers included in the same special issue provide a full spectrum of related information. An efficient method to code the boundary (i.e., shape) of an object can subsequently facilitate efficient image or video coding.

On the other hand, some artificial images—such as line drawings, maps, cartoons, and blueprints—are mainly composed of lines and curves. A coding scheme that is particularly suitable for linear or curvilinear features is expected to work better in compressing these images.

Some relevant works will be reviewed. Back in the 1960s, runlength coding first emerged [11]. The basic idea of a runlength coder is to treat a binary image as a nearest neighbor graph—a pixel is connected to its four (or eight) nearest neighbors—and one codes the relative positions of the neighboring pixels. Soon after, a chain coding method was proposed [12]. The key idea of a chain coder is to code, at each step, multiple and aligned pixels (that form a line segment), instead of one nearest neighbor. Many improvements have been introduced for chain coding. For example, in [18], statistical modeling and

arithmetic coding was added as a post-processing method to fully take advantage of the statistical structure in a chain of line segments. Alternatives other than straight line segment have been explored as basic elements. These works include discrete arcs [5], polygonal curves [16], and a few more. Researchers have noticed that it is not necessary to follow exactly a curve given by a discrete image. Methods taking into account the tradeoff between the fidelity and efficiency of a contour coder are studied in [21]. More interestingly, the idea of using multiscale structure (i.e., a pyramid) to code a contour has been studied by several researchers, e.g., [17] and [19]. Despite all the existing work, the authors have not seen any work that applies tree-based coding for curves. Moreover, the usage of Beamlets, or any other structure like beamlets, has not been addressed explicitly in the literature. The intent of this paper is to develop a fully multiscale coder for curves.

The idea of tree-based coding, which was established in wavelet coding and was adopted as an option in JPEG 2000 [24], has been very successful in image coding. However, a similar idea has *not* been adopted to code curves. Due to the fact that beamlets are newly proposed data structure in statistics [8], it is a good time to address this problem.

The proposed coding method, which is named JBEAM, is efficient: by comparing to existing industrial standard software-JBIG 2-our beamlet coder requires fewer bits.

The proposed method has a low order of computational complexity. It requires a bottom-up tree pruning algorithm in computing the optimal representation. There are efficient means to realize the tree-based coding. In the algorithm description, we will prove that our method has computational complexity $O(n^2)$, not counting the beamlet fitting part, for an $n \times n$ binary image.

The proposed method is *partially* progressive. In each separate stage, the proposed method allows partial reconstruction. It achieves the progressivity by stages, instead of globally. More discussion regarding this will be provided below.

The major innovation of this work is the design of a tree-based coder for curves. In addition, to our knowledge, this is the first appearance of the coding of single beamlets.

Besides the novelty of the methodology, the proposed method will achieve progressivity to the degree that no other curve coding method has reached before. Its effectiveness in application will be demonstrated later.

In our coding method, there are three stages. In the first stage, a beamlet representation, based on optimizing a rate-distortion function, is introduced. In the second stage, given the beamlet representation, an approach similar to the zerotree-based coding is utilized to generate a symbol stream. In the third stage, an entropy coding (based on Lempel-Ziv, which is available as "gzip" in most of the UNIX systems) is applied to code a symbol stream

into a bit stream. In general, we found that the last stage gives *insignificant* compression. Hence, it can be left out without much loss in performance.

Our coder combines three main ideas:

1) utilizing a newly developed data structure (i.e., beamlet);
2) a rate-distortion optimized representation;
3) a tree-based coding strategy, considered a sibling of EZW [22] and SPIHT [20], which are for image coders.

Some recent papers provide updated information on the applications of curve coding. One paper gives a good overview on contemporary shape coding methods, which have been used in multimedia signal processing [27]. The same authors proposed a skeleton-based shape coding method. Another recent paper discusses an enhancement on the traditional context-based methods [3], which was benchmarked with CAE and JBIG [14]. Although these works solve a problem that seems identical to ours, our objective is slightly different. We are studying a more general approach, while the others' works emphasize more on the application front.

Our method is a *curve coding* method. It is different from shape coding because the boundary of an object (i.e., a shape) has to be closed, while a curve does not need to be closed. Moreover, curves may have intersections. Examples of images that our method can deal with can be found in Fig. 11.

The rest of the paper is organized as follows. In Section II, beamlets and how to code a single beamlet are described. Section III describes beamlet representations, as well as how to compute a rate-distortion optimized beamlet representation. The coding scheme is presented in Section IV. Simulation results are presented in Section V. Section VI includes discussion, and, in Section VII, concluding remarks are provided.

II. BEAMLETS

The main idea of beamlet representation is to approximate linear objects in two dimensions by multiscale line segments [9]. The dictionary of beamlets has low order of complexity ($O(n^2)$), compared to the set of all possible line segments ($O(n^4)$), and it takes a small number of beamlets to approximate an arbitrary line segment within a given precision.

In this section, we describe beamlets, their digitization, and how to code a single beamlet.

A. Beamlet Dictionary

The beamlet dictionary is a dyadically-organized library of line segments, which are at a range of scales and locations, and have different orientations and lengths. The dictionary facilitates a multiscale approximation to the collection of all line segments. Beamlets can be defined in the following three steps.

1) Recursive Dyadic Partitioning (RDP): Consider a unit square $[0, 1] \times [0; 1]$. We first divide the unit square into 2×2 smaller squares with equal dyadic sidelengths. Each small square is further divided into 2×2 smaller squares, still having equal and dyadic sidelengths. This process is repeated until the finest scale is reached.
2) Vertex Marking: On the boundary (four sides) of each square, starting from the northwest corner, vertices are marked clockwise at equal distance. The interdistance of
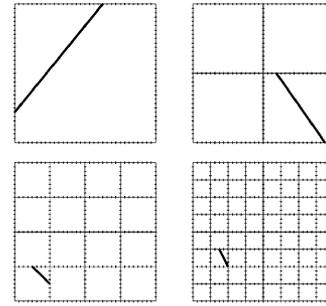

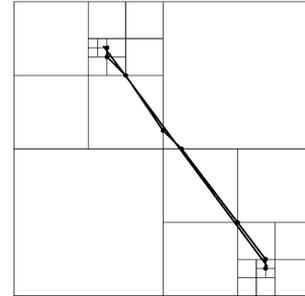Fig. 1.   Dyadic squares marked with vertice and several beamlets.

Fig. 2.   Approximating a (red) line segment by a chain of (green) beamlets.
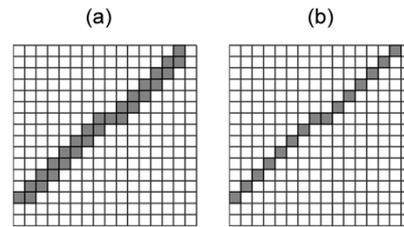

Fig. 3.   Two discrete beamlets. (a) One is generated by allowing connecting to four neighboring pixels. (b) The other eight neighboring pixels.

the vertices is fixed in advance, and it does not vary with the sidelengths of the squares.
3) Connecting: Within each square, any pair of vertices on its boundary determines a line segment. This line segment is called a *beamlet*.

Fig. 1 illustrates the key idea in defining beamlets. The set of all beamlets forms a *beamlet dictionary*. Fig. 2 shows that any line segment can be approximated by a chain of beamlets. A formal description with more details can be found in Donoho's paper [8].

B. Digital Beamlets

Given two end pixels on the boundary of a dyadic square, a *digital beamlet* is determined by digitally interpolating between the two pixels. There are two topological possibilities:

1) multiple pixels are allowable per column;
2) only a single pixel is allowable per column.

A more rigorous (i.e., mathematical) discussion regarding how to numerically realize the above idea is provided in Appendix I. Fig. 3 gives two digital beamlets with the same endpoints, satisfying different topological conditions. We will concentrate on a digital beamlet that follows the second condition, which we will call a *thin* digital beamlet.

TABLE I
NUMBER OF BITS TO CODE BEAMLET IN DYADIC SQUARES WITH A
RANGE OF SIZES. WHEN SCALE s IS LARGE, THE RULE "$2s + 3$" IS
FOLLOWED. HOWEVER, WHEN THE SCALE IS SMALL, e.g., $s = 0, 1$,
THE REQUIRED NUMBER OF BITS CAN BE SMALLER. SEE THE
NUMBERS IN THE PARENTHESES IN THE LAST ROW

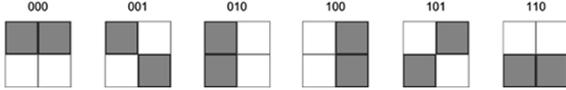| Scale, s	0	1	2	3	4
Size, 2^s	1	2	4	8	16
# beamlets	1	$\binom{4}{2}$ $= 6$	$\binom{12}{2}$ $= 66$	$\binom{28}{2}$ $= 378$	$\binom{60}{2}$ $= 1770$
# bits	(1)	(3)	7	9	11

Fig. 4. Six possible beamlets in a 2×2 image. The coded bits are in the titles.

C. Progressive Coding of A Single Beamlet

We count possible configurations for a single beamlet. This gives us an upper bound for the number of bits that are required. When the size of an image is $2^s \times 2^s$, there are $2^{s+2} - 4$ boundary pixels. Coding a beamlet is equivalent to coding the positions of its two endpixels. It can be done with $(2s + 3) = 2(s + 2) - 1$ bits simply because

$$\log_2 \binom{(2^{s+2} - 4)}{2} \leq 2s + 3.$$

The upper bound given in the above inequality gives a good estimate for the number of bits that are needed. However, when the size of the square (2^s) is small, the number of bits can be significantly reduced. Table I summarizes these quantities. This prompts us to treat the beamlet coding for small squares differently than for large squares, as we do in our coding scheme.

Given a $2^s \times 2^s$ image, we would like to *progressively* code all the possible beamlets, within the bits limits that are given in Table I. Let us first consider some simple cases. For a 1×1 image, there is only one possibility; hence, one bit is sufficient. For a 2×2 image, there are four boundary pixels. Hence, there are six possible beamlets, which are depicted in Fig. 4. They can be coded by 3 bits.

It is more interesting to consider the case when $s \geq 2$. The following proposition is a stronger result.

Proposition 2.1. Consider a length-$(2^K - 1)$ consecutive sequence, $\{1, 2, 3, \ldots, 2^K - 1\}$, in which there is a substring

$$s(i, j) = \{i, i + 1, \ldots, j - 1, j\}$$

where $1 \leq i < j \leq 2^K - 1$. There is a coding scheme for all substrings $s(i, j)$ such that:
1) the coder is progressive;
2) it takes at most $(2K - 1)$ bits;
3) when there are $2k (k < K)$ bits available, the position of the two ends of interval can be recovered with worst case distortion (measured by absolute distance) 2^{K-k};
4) the above distortion is the minimum possible distortion.
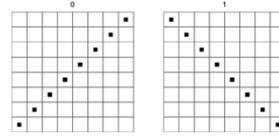
Proof: See Appendix II. ∎

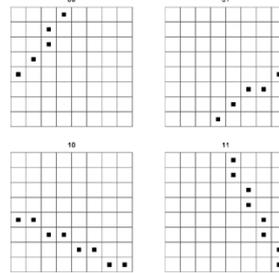Fig. 5. Representable digital beamlets in a 8×8 image when 1 bit is available.

Fig. 6. Representable digital beamlets in a 8×8 image when 2 bits are available. In this figure and Fig. 7, the coded bits are in the titles.
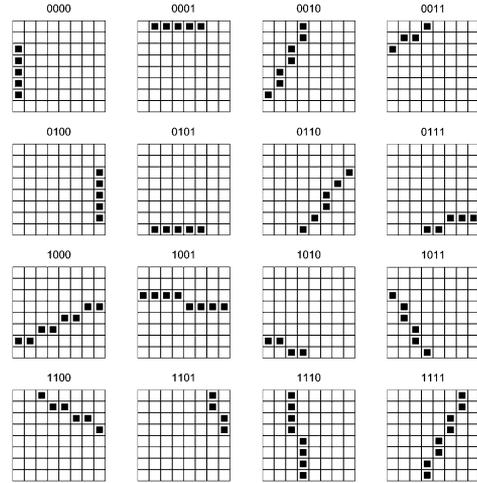

Fig. 7. Representable digital beamlets in a 8×8 image when 4 bits are available.

The proof of the above proposition is constructive. It leads to a progressive coding scheme for all single beamlets in a square. Recall in a $2^s \times 2^s$ square, there are $2^{s+2} - 4$ boundary pixels. Assume these boundary pixels form a sequence, for example, by sampling them clockwise. Then, add three dummy pixels at the end of this sequence. According to the above proposition, there is a progressive coder, whose worst case distortion is minimized for a given number of bits.

We illustrate the progressivity: when more bits are coming in, the position of a beamlet can be approximated better. To illustrate this phenomenon, we present some *representable beamlets* for a 8×8 image, when 1, 2, 4, and 8 bits are available: Figs. 5–8. *Representable beamlets* are picked as follows: They are derived from the coding scheme that is described in detail in Appendix II. Consider there is only one bit available. Suppose the upper right pixel is chosen as the "central pixel" of the coding. The first bit will tell if this pixel is located between the two end pixels of a beamlet, or not. If not (the first bit is "0"), a representative beamlet goes from the central pixel to the last pixel. If so (the first bit is "1"), a representative beamlet goes from the middle of the first half to the middle of the second half.
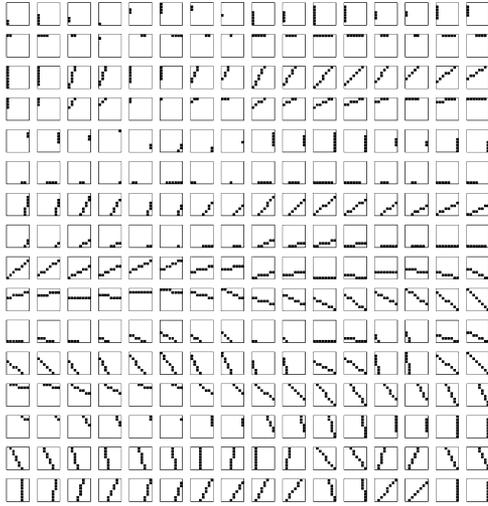
Fig. 8.   Representable digital beamlets in a 8 × 8 image when 4 bits are available.

Fig. 9.   Taking a range of values of λ has insignificant effect on the fitted beamlets.

These two beamlets are plotted in Fig. 5. When more bits are coming in, the representation of the beamlet becomes more accurate to the target line segment. When there are 8 bits, the representative beamlets are illustrated in Fig. 8. Note that 9 bits is sufficient to code all digital beamlets in an 8 × 8 image. Hence, the beamlets in Fig. 8 can approximate an arbitrary beamlet with high accuracy—more specifically, with, at most, one distortion regarding the positions of endpoints.

III. BEAMLET REPRESENTATION, DISTORTION, AND RATE

In this section, we first describe how to fit a single beamlet in a square; then, we describe beamlet representation and a top-down algorithm to compute a beamlet representation.

A. Distortion for A Single Beamlet and One-Beamlet Fitting

A distortion function measures the goodness-of-fit of a beamlet in a given dyadic square; it is a necessary quantity in beamlet fitting. In this section, this quantity is defined.

Let us consider a binary image, which is made by black and white pixels. We assume that only the positions of the black pixels matter—they form the curves. Let y denote the set of black image pixels. Let S denote a dyadic square. Hence, $y \cap S$ denotes the set of image pixels within square S. Let b denote a beamlet, which interchangeably represents a set of pixels that the corresponding digital beamlet traverses. There are two qualitative measures of a beamlet approximation. One is the sum of the Euclidean distances between image pixels and the beamlet. The other is the number of pixels that are noncoincided—pixels that are either in the image, but not in the beamlet; or in the beamlet, but not in the image. We define a distortion function by combining these two measures

$$d_S(y,b) = \chi_1(y \cap S, b) + \lambda \cdot \chi_2(y \cap S, b) \quad (1)$$

where λ is a constant

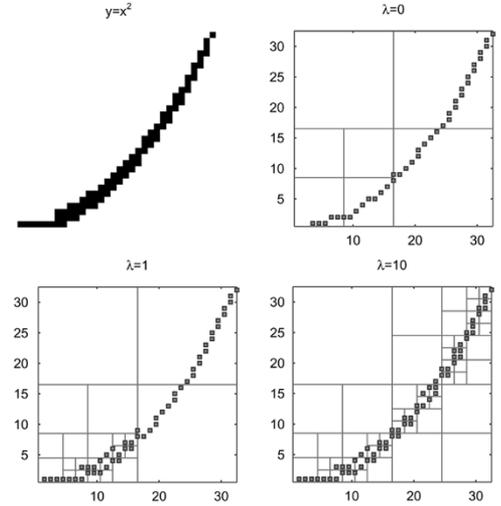$$\chi_1(y \cap S, b) = \sum_{x \in y \cap S} \rho_1(x, b) \quad (2)$$

where $\rho_1(x, b)$ denote the distance between a pixel x and a digital beamlet b, and

$$\chi_2(y \cap S, b) = \sum_{x \in y \cap S} I\{x \notin b\} + \sum_{x \in b} I\{x \notin (y \cap S)\} \quad (3)$$

where $I\{\cdot\}$ is an indicator function. Constant λ is prescribed and positive; it controls the tradeoff between the two types of measure. A discussion on how to choose the value of λ will be in the next section.

In the above definitions, the function $\chi_1(y \cap S, b)$ measures the deviation from y to the digital beamlet b. Within S, when y and b are identical, one has $\chi_1(y \cap S, b) = 0$. The detailed calculation regarding $\chi_1(\cdot, \cdot)$ is given in Appendix III. The second function $\chi_2(y \cap S, b)$ measures the degree of overlapping between the image y and the beamlet b. When the image y and beamlet b coincide in S—occupying the same set of pixels—we have $\chi_2(y \cap S, b) = 0$, and vice versa.

Now, let $\mathcal{B}_S$ denote the set of all possible digital beamlets in square S. The minimum possible distortion of a beamlet fit in square S is given as

$$D(y, S) = \min_{b \in \mathcal{B}_S} d_S(y, b). \quad (4)$$

B. Discussion Regarding the Above Defined Distortion

1) Comparison with the Distortion used in Proposition 2.1: An immediate question concerns how the above defined distortion (second distortion) is associated with the distortion that was used in Proposition 2.1 (first distortion). The key difference is that the first distortion is only effective at measuring the difference between a beamlet and an line segment across; while the second distortion can measure the difference between a beamlet and an *arbitrary* set of image pixels. In fitting a curve image, the complexity of the second distortion is necessary. From now on, in beamlet fitting and beamlet representation, the second distortion is used.

2) Role of λ in (3.1): Note that, in defining the second distortion in (1), a parameter λ is used. Recall that it controls the tradeoff between two types of measure. We found that the choice of its value has some but not significant effect on the final result. To show its effect, an illustration is given in Fig. 9. Note in Fig. 9,

beamlet representation (instead of fitting a single beamlet) is used. We refer to later parts of this paper for more details.

C. Beamlet Representation

We describe the basic idea of the beamlet representation. In the initial step, we build a complete beamlet-decorated quad tree, having depth $\log_2(n)$. Each node is associated with a square, and its four child nodes are associated with its four subsquares. A beamlet representation (BR) is associated with a particular image. Each BR corresponds to an admissible subtree of a complete quad tree. Such a subtree is associated with an incomplete recursive dyadic partitioning (RDP). From this point on, we choose $\mathcal{P}$ to denote an incomplete RDP. Each node is marked with one of three symbols: Q (quadratic split), N (no split), and B (beamlet secorated). Symbol Q can only be associated with an intermediate node, while symbols N and B can only be associated with terminal nodes.

There are two possible approaches to compute a beamlet representation: bottom-up and top-down. Here, we describe a top-down approach, as an example. In our computation for optimal-rate-distortion BR, a bottom-up approach is adopted. A top-down approach works as follows. If there is no content in a sub-square in a RDP, then the node associated with this subsquare is marked by N. If a beamlet can depict the image content in a subsquare, then the associated node is marked by B. We do this marking recursively: if N or B can be assigned, the partition stops; otherwise, the current subsquare is marked with Q, and then divided into smaller subsquares. The process is repeated in the smaller subsquares. In fact, this coincides with the idea of beamlet-decorated recursive dyadic partitioning (BDRDP) that is described in the paper [8]. We utilize such a hierarchical structure to facilitate tree-based coding.

D. Simple Example for BR

As an example, we consider a binary image in Fig. 10. At the coarsest level, no beamlet can fit well; hence, a quadratic split (Q) is assigned. At the next level, two squares contain no dark pixels. They are labeled as empty (N). One square contains a beamlet (B_1). The last remaining square has no appropriate fitting; hence, it is split into four (Q). In the next level, the four smallest squares either contain no dark pixels or are fitted by beamlets. The corresponding partitioning and the vertices of beamlets are given as follows:

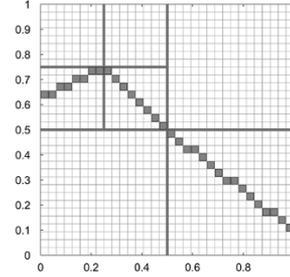| level 0	level 1		level 2			
			N	N	–	–
Q	Q	N	B_2	B_3	–	–
	N	B_1	–	–	–	–
			–	–	–	–

$$(5)$$

Fig. 10. Example of binary image. Note that this image is not a shape. It is made for illustrating the idea of beamlet representation.

where Q, N, and B denote quadratic split, no split, and beamlet, respectively. Symbol "-" in the above table means no need of symbols. The coordinates of beamlets and their corresponding bit representations are shown in (6), shown at the bottom of the page. The bits in the last column are the results of coding single beamlets. Note that from the above two tables, (5) and (6), one can reconstruct the binary image.

We now describe a criterion in finding an optimal beamlet representation. We define the overall distortion of a BR $\mathcal{P}$ as

$$D(y, \mathcal{P}) = \sum_{S \in \mathcal{P}} D(y, S)$$

where y is a binary image and $D(y, S)$ is the square limited distortion function that is defined in (4). Recall that $S \in \mathcal{P}$ indicates that a square S is a *terminal* node in BR $\mathcal{P}$. In fact, we only need to consider the square S that is marked as beamlet decoration (i.e., B). If square S is marked as N, then $D(y, S) = 0$, because the image is empty within S.

Given a BR, one needs to know how many bits are necessary in order to code the BR. Recall that $\mathcal{R}$ stands for an RDP; by allowing it to contain symbols, it can also denote a BR (we allow a little bit of abuse of the notation). Let "$S \in \mathcal{P}$" indicate that a subsquare S is a *terminal* node in RDP $\mathcal{P}$. The overall rate function that is associated with RDP $\mathcal{P}$ can be defined as follows:

$$R(\mathcal{P}) = \text{\# of bits that are required to code } \mathcal{P}$$
$$= [\log_2(3)](\text{\# of symbols in a BR}) + \sum_{S \in \mathcal{P}} R(S)$$

where the "# of symbols in a BR" is the numbers of N, Q, and B in a beamlet representation, the constant $\log_2(3)$ is the average cost to code a symbol, and $R(S)$ is the number of bits for coding a digital beamlet if square S is marked as B.

Note that it takes very little to code a symbol, while it might take quite a lot to code a beamlet. In the forthcoming tree-based coding, we will give symbols higher priority.

On the other hand, in computing the "rate," we ignore the interdependency among the symbols, which could substantially decrease the required amount of bits. The above formula is

	first vertex	second vertex	binary represent
B_1	16	45	10 011 111 011
B_2	5	16	100 100 000
B_3	8	24	101 101 010

$$(6)$$

merely an estimate. The purpose is to facilitate the computation of a rate-distortion-optimized BR.

To find the rate-distortion optimized beamlet representation, we consider the following constrained optimization problem:

$$\min_{\mathcal{P}} R(\mathcal{P}), \qquad \text{subject to } D(y, \mathcal{P}) \leq D^0$$

where constant D^0 is the maximum allowable distortion. The above problem is normally solved via a Lagrangian multiplier method: for $\tau > 0$, consider

$$\min_{\mathcal{P}} R(\mathcal{P}) + \tau D(y, \mathcal{P}). \tag{7}$$

A range of the values of the Lagrangian multiplier, τ, are examined. The frontier of the set, $\{(D(y, \mathcal{P}), R(\mathcal{P})) : \mathcal{P} \text{ is a RDP}\}$, which is called a *feasible set* in many publications, is explored.

E. Bottom-Up Tree Pruning Algorithm

The problem (7) can be solved efficiently by using a *tree pruning* algorithm. Similar algorithms have been designed for other purposes, for example, the best-orthonormal basis (BOB) [7] in finding the optimal wavelet packets.

First, each RDP is associated with an admissible subtree of a complete quad tree. Suppose node (square) S has four children (subsquares) $S_i, i = 1, 2, 3, 4$. For $1 \leq i \leq 4$, let $(D_\tau(y, S_i), R_\tau(S_i))$ denote the pair of distortion and rate, such that for a given value of τ, the objective function in (7) is minimized within square S_i. We consider the options at node S.

1) If all $S_i, i = 1, 2, 3, 4$ are marked with symbol N, then node S is marked with N, and assign

$$D_\tau(S) = 0 \quad \text{and} \quad R_\tau(S) = \log_2(3).$$

2) If node S is not marked with N, there are two possibilities.

a) If node S is marked with a Q, a quad-split is implemented, and we assign

$$\tilde{D}_{\tau,1}(y, S) = \sum_{i=1}^{4} D_\tau(y, S_i) \quad \text{and}$$

$$\tilde{R}_{\tau,1}(S) = \log_2(3) + \sum_{i=1}^{4} R_\tau(S_i).$$

The objective function in (7) is equal to

$$\tilde{R}_{\tau,1}(S) + \tau \cdot \tilde{D}_{\tau,1}(y, S). \tag{8}$$

b) If node S is marked with a B, a digital beamlet is fitted within square S, and we assign

$$\tilde{D}_{\tau,2}(y, S) = D(y, S)$$

and

$$\tilde{R}_{\tau,2}(S) = \log_2(3) + (2j + 3)$$

where function $D(y, S)$ is the minimum distortion that is defined in (4) and j is the scale of square S: square S is a $2^j \times 2^j$ square. The objective function in (7) becomes

$$\tilde{R}_{\tau,2}(S) + \tau \cdot \tilde{D}_{\tau,2}(y, S). \tag{9}$$

We pick the option that gives the minimum value between (8) and (9).

Evidently, the above provides a bottom-up tree "pruning" algorithm: If an N or a B is marked, the subtree rooted at this node is "pruned"; if the node is marked with a Q, then there is no pruning at this node.

Since this tree-pruning algorithm is fairly well-known [6], we choose to describe it briefly. Fix a value for τ. Consider a quad tree. Starting from the bottom of a quad tree, we consider a quadruplet that share a parent. Following the above description, determine which one of the three options shall be chosen. Record the corresponding rate and distortion, and store it at the parent node. Do this for all quadruplets at this level of the quad tree. Then move up to the level immediately above, and repeat the procedure. The entire procedure is terminated when the top node is reached.

The resulting BR is a solution to the problem in (7); this can be proved by induction. Such an approach is often called *dynamic programming*.

By running this algorithm for different values of τ, one can find a set of rate-distortion-optimized beamlet representations, having different pairs of (rate, distortion).

A recent manuscript adopts a similar approach to code piecewise smooth images [23]. Despite the strong similarity in the algorithm, their algorithm serves a different purpose: coding images rather than curves.

F. Computational Complexity

Suppose we have an $n \times n$ image. Suppose n is dyadic: There exists an integer J, such that $n = 2^J$. At each level of the quad tree, there are $1, 4, 4^2, \ldots, 4^{J-1}$, number of quadruplets. Note that the sum of the above numbers is $\leq n^2$. Suppose, in choosing among the three options as described in the previous section, for each quadruplet, it takes $O(1)$ operations. The overall complexity of the algorithm is at most $O(n^2)$. In fact, by studying the three options in the previous section, assuming that the beamlet fitting has been precomputed for all squares whenever it is appropriate, the complexity of choosing one option can be verified as $O(1)$.

The above analysis ignores the computation of fitting beamlets—it only considers the pruning part. In fact, the fitting of beamlet in a square by itself may not be an easy task, as we show in some initial analysis. Intuitively, for a $2^s \times 2^s$ square, there are $\binom{4 \cdot 2^s}{2}$ beamlets; it takes $O(2^s)$ to compute the distortion for each beamlet. Hence, it takes $O(2^{3s})$ to fit a beamlet in a square; it takes $O(n^3)$ for the entire image. However, sophisticated algorithms have been designed to reduce the computational complexity for similar problems, e.g., [4]. The main idea is to utilize or improve the Fourier Slice theorem, which is widely used in scientific fields such as computational tomography. We strongly believe that an algorithm of a low-order computational complexity exists here, even though the exact algorithm is not addressed—which, by itself, can be a very complex problem. Following the same principle, the complexity of fitting

TABLE II
OVERVIEW OF A BEAMLET CODER

Binary Shape Image,	
↑ ↓	Beamlet Transform
Beamlet Representation,	
↑ ↓	BR to symbol stream conversion
Symbol Stream,	
↑ ↓	Coding
Bit Stream.	

a beamlet in a $2^s \times 2^s$ square can be $O(s \cdot 2^{2s})$. Hence, the complexity for beamlet fitting for an entire image is $O(n^2 \log(n))$, compared with $O(n^2)$ without considerations of beamlet fitting.

IV. CODING

Given that a beamlet representation has been computed, a coding scheme is now described.

A. Overview of A Beamlet Coder

There are three steps in a beamlet coder. A binary image that contains linear or curvilinear features is the starting point. From the binary image, a beamlet representation is calculated, as described in the previous section. This first step is called *Beamlet transform*. Based on the BR, a stream made by symbols and bits is derived. We call this stream a *symbol stream*. This second step utilizes the idea of zerotree coding. Finally, an arithmetic/entropy coder is applied in the third step. Table II depicts the entire procedure.

Only the encoding part of the second step will be described. The decoding part is not hard to derive.

B. Tree-Based Compression of Beamlet Representation

Now, from two tables, (5) and (6), which are generated in the previous section, we consider how to generate a stream that is made only by symbols (Q, N, and B) and bits (0 and 1). We describe it in two stages.

In the first stage, we generate a L by $2J + 4$ array, which is made by symbols and bits. Here, L denotes the number of symbols in table (5). All the symbols are listed in the first column, in an order such that the symbols at coarser scales coming before the symbols at finer scales. The binary representation of *beamlets* are listed thereafter. For a beamlet representation given by table (5) and (6), the array looks as follows:

$$\begin{array}{rcl}
l_0 & \to & Q \\
l_1 & \to & Q \\
 & & N \\
 & & N \\
 & & B_1 \quad 1\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1 \\
l_2 & \to & N \\
 & & N \\
 & & B_2 \quad \ \ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0 \\
 & & B_3 \quad \ \ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0.
\end{array} \qquad (10)$$

Note a notation l_j is used to denote the starting point of symbols coming from scale j. Also note that, at the same scale, the order of symbols is not critical, but has to be fixed. We can choose any reasonable ordering scheme, e.g., raster scan [13].

In the second step, a symbol-and-bit stream is generated by the following algorithm.

Algorithm: Convert a Beamlet Representation to Symbol Stream.

For $j_1 = 0 : J$, where $J = \log_2(n)$ and n is the size of the image,

    1) Enumerate all symbols at level j_1, by enumerating first elements of rows between l_{j_1} and $l_{j_1+1} - 1$.

    2) Enumerate bits by the following procedure.

        For $j_2 = 0 : j_1 - 1$,

            enumerate the $[2(j_1 - j_2) + 2]$th and $[2(j_1 - j_2) + 3]$th bits of the binary representation of all beamlets in scale j_2.

        End

        Enumerate the first three bits of the binary representation of all beamlets in scale j_1.

End.

Enumerate all remaining bits, by following the order given in the second "for" loop above.

The above algorithm has appeared in EZW and SPIHT [22], [20]. Applying the above algorithm to the table in (10), the result looks like the following:

$$\begin{aligned}
&Q, Q, N, N, B, B_1(1), B_1(2), B_1(3), N, N, B, B, \\
&B_1(4), B_1(5), B_2(1), B_2(2), B_2(3), B_3(1), \\
&B_3(2), B_3(3), \ldots B_1(2J), B_1(2J + 1), B_2(2J-\\
&2), B_2(2J - 1), B_3(2J - 2), B_3(2J - 1).
\end{aligned}$$

Here, $B_i(k)$ denotes the kth bit of the ith beamlet. Substituting the symbols with the values from (5) and (6), the complete symbol stream is as follows:

$$\begin{aligned}
&Q, Q, N, N, B, 1, 0, 0, N, N, B, B, 1, 1, 1, 0, \\
&0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, \\
&0, 0, 1, 0, EOF.
\end{aligned}$$

Here, "EOF" stands for the end of the file.

By using an arithmetic coder [28] or a Lempel-ziv coder, one can further code the above stream into a bit stream.

It appears that the transformation (i.e., the beamlet representation) and the conversion (BR to symbol stream) are decoupled. This seems to be an undesired feature of our coder. In fact, they are integrated in the sense that the conversion always leaves the fine scale information at the later part of the symbol stream. The BR and the BR-to-symbol-stream conversion are consistent.

V. SIMULATIONS

In this section, simulations are conducted to evaluate the performance of our beamlet coder. In the first set of experiments, Section V-A, we compare JBEAM with JBIG in coding binary images, which are made by curves. It is found that JBEAM can achieve a smaller number of bits—by allowing some distortion, it may take less than 50% of the amount of bits that is required by JBIG. On the other hand, the progressivity and rate-distortion behavior of JBEAM, as well as an illustrative application in a video stream, are provided in the second set of experiments (Section V-B).
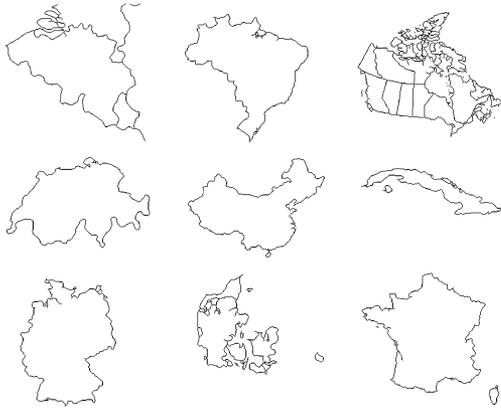
Fig. 11.   Country borders. We use them as our test images.

TABLE III
SIMULATION RESULT FOR LOSSLESS CODING
OF BORDER MAPS OF NINE COUNTRIES

Country	Pixels	JBIG 2 (lossless)	JBEAM (lossless)	JBEAM (lossy)
Belgium	1233	7888	6114	3098
Belize	822	5768	4391	2021
Canada	2911	15224	14555	7366
Switzerland	845	6088	4171	2135
China	813	5792	4609	2041
Cuba	658	4360	3303	1542
Germany	871	5792	4916	2138
Denmark	1350	7992	6780	3735
France	920	6448	5300	2339

A. Line Images

We test the performance of JBEAM on some map images, which are shown in Fig. 11. They are borders of countries; each map is a 256×256 binary image. The original data is provided in the software package [2]. It is compared with JBIG 2 [14].

B. Operational Rate-Distortion Representation

The experimental results are given in Table III. In the table, the first column (Country) lists the names of the nine countries, and the second column (Pixels) lists the numbers of black pixels per image. In the third and fourth columns, the performances of JBIG 2 and JBEAM for lossless coding are compared. The last columns list the number of bits for lossy JBEAM coding.[1] Fig. 12 presents the difference between a reconstruction from a lossy JBEAM and the original binary image. It is observed that by introducing a permissible maximum error of one pixel, we are able to reduce the total number of bits by about 50%.

For fairness, since the JBIG 2 software that we used does not add an entropy coding at the end, the results of JBEAM in Table III are also before entropy coding. By applying entropy coding, we found that the lengths of bit streams typically can be reduced by another 10%.

The illustration in Fig. 12(a) is informative about the property of JBEAM. Fig. 12 shows the closeness between the reconstruction from a lossy JBEAM (the black one) and the original

[1]We choose $\lambda = 0.1$ and $\tau = 5$. They were chosen experimentally such that maximal-1 distortion is achieved.
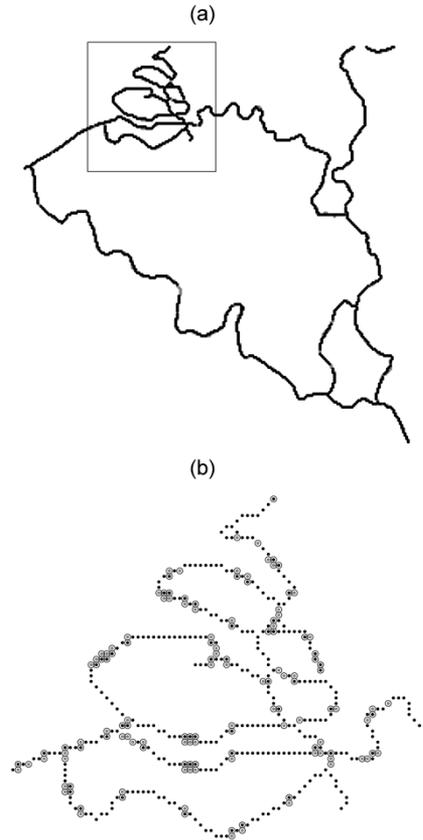

Fig. 12.   (a) Difference between original image and the reconstruction from a lossy JBEAM. (b) Enlargement of the most twisty part, which also is the hardest part to code.

binary image (the gray one). It is hard to see significant differences. In Fig. 12(b), a circle with an inside dot is a pixel that is on the reconstruction of the lossy JBEAM, but is not originally black. An empty circle indicates a pixel that is originally black, but not in the reconstructed image. In many situations, the two types of circles are next to another. This indicates that in the lossy JBEAM, the coder simply moves the image pixel by one, to make the feature more linear! Note that the part in Fig. 12(b) is the most difficult region to code. Hence, it contains the majority of the total distortion.

Based on the above analysis, by allowing a small amount of distortion, one may significantly improve the coding efficiency of JBEAM.

On the other hand, one can easily suggest a malicious situation against the design in JBEAM: imaging two lines, with one immediately on the top of the other. This explains, in an intuitive way, why a lossy JBEAM can be greatly advantageous for an arbitrary image.

One can overcome the above-mentioned difficulty by applying preprocessing. For example, one can intentionally "thin" all the linear and curvilinear features, by calling morphological operators.

C. Coding Object Shapes in Video Sequences

To illustrate other aspects of JBEAM, we apply it to a video sequence. The shapes in this sequence will be coded. Our results are based on two testing sequences.
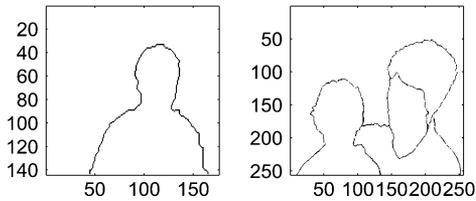
Fig. 13. Shape images. Left: *Figure* sequence. Right: *Mother-daughter* sequences.
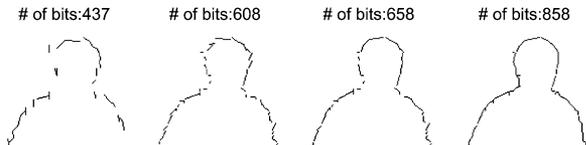

Fig. 14. Progressive reconstruction of the sequence *Figure* (frame 0). The number of transmitted bits are shown in the titles.
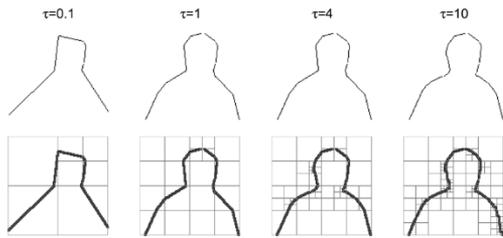

Fig. 15. Beamlet representations of sequence *Figure* (frame 0), with different given τs.

- *Figure* sequence [1], QCIF(144×171 pixels per frame). The shape images are cropped to 128×128.
- *Mother-daughter* sequence, MPEG(2). Two representative shape images are cropped to 256×256.

A sample (which is the shape in one frame) from each of the above sequences is shown in Fig. 13.

1) Progressivity of JBEAM: The progressivity of JBEAM is illustrated in Fig. 14, for the first frame of the video sequence "Figure." In the beamlet representation, the parameters are chosen as $\lambda = 0.3$ and $\tau = 10$. The coded bit stream is truncated in the tails, and an approximate shape is reconstructed, based on the partial stream. In Fig. 14, from left to the right, when more bits are available, the reconstruction becomes more similar to the original shape.

2) Rate-Distortion Tradeoff: In the beamlet representation, the depth of partition mainly depends on the Lagrangian multiplier τ. In Fig. 15, we fix different values of τs, and plot the beamlet representations (lower row) with their corresponding reconstructed images (upper row).

3) Rate-Distortion Curve: In Fig. 16, we plot the rate and distortion distribution for the first 50 frames of the two sequences. Fig. 17 shows the rate-distortion curves of the beamlet coding averaged over these 50 frames. The distortion that is used here is the joint distortion $d_S(\cdot, \cdot)$ that is defined in (1).

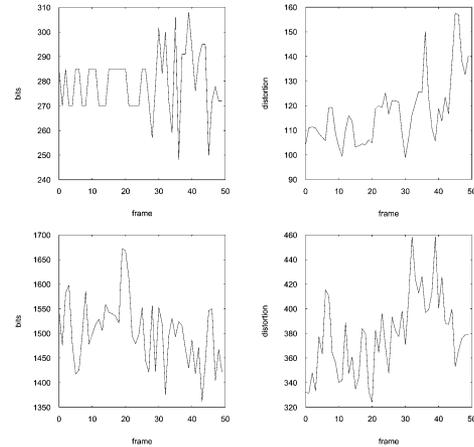It empirically shows that the rate-distortion curve behaves exponentially.

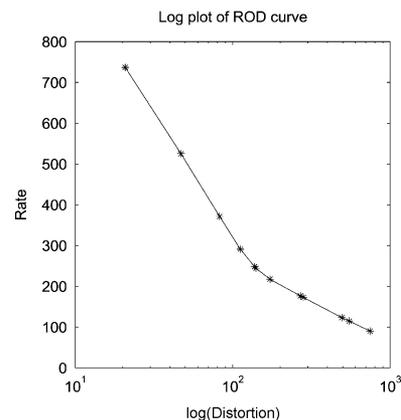Fig. 16. Rate and distortion distribution. Top: "Figure" sequences. Down: "Mother" sequences.

Fig. 17. Rate-distortion curves of two sequences.

VI. DISCUSSION

A. Variation 1: Shape Coding

JBEAM serves a broader purpose: coding curves, which include not only shapes, but also other linear and curvilinear features. In Fig. 14, one can observe that the partial reconstructions in JBEAM are not satisfactory: they are *not* continuous, while shapes should be continuous. With a slight modification in the beamlet representation, one can generate a coder that is specifically designed for shapes. The key idea is to use the sides of the boundary pixels, not the boundary pixels, to index the positions of beamlets. By doing so, one enforces the output of a beamlet representation to be a continuous chain of beamlets, i.e., a continuous curve. Fig. 18 gives an illustration of the above idea. The figure shows that the thickened sides of boundary pixels serve as beamlet indices in BR. The remaining components in such a coder follow a nearly identical approach to JBEAM.

There will be some anticipated difficulties in implementing the approach described above. For example, in fitting the beamlets, the additional condition that the beamlets are continuous will make its computation more complicated. It will be interesting to articulate its details in the future.
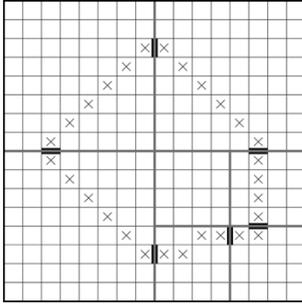
Fig. 18. Illustrate a new indexing scheme of beamlets, in a beamlet representation, which facilitates coding continuous curves.

B. Variation 2: Totally Progressive?

Our algorithm first compute a beamlet representation, based on the rate-distortion curve, and then code the symbol-bit stream. This approach is somewhat divided. Making the coder more coherent is an interesting area of future research.

JBEAM can be easily made progressive in the sense of the "first distortion" (i.e., χ_1), but not strictly the second. As more and more bits coming in, the distortion from the coded beamlet to the "optimal" beamlet decreases in χ_1. In comparison, the "second distortion" (i.e., the value of χ_2) of the partially coded beamlet to the image block may not strictly decrease as more and more bits are added.

C. Comparison With Wavelet Image Coder

The rate-distortion optimization is designed for beamlet representation. It has not been integrated with coding. In the problem of wavelet coding, the problem is more straightforward; the rate-distortion consideration is integrated with the tree-based coding. It is not clear how such an integration can be achieved in beamlet-based coding. The challenge is that in a beamlet coding, when distortion changes, the geometry [which is represented by the structure of the quad tree, as recorded in (5) and (6)] can be changed as well, while in wavelet coding, the quad-tree structure is fixed. This is a problem that we have not resolved well.

D. Comparison Between Lossy JBEAM and Lossy JBIG

Notice that we did not compare a lossy JBEAM with a lossy JBIG2. We found that it is difficult to do so. JBIG2 gives a wide variety of possible approaches [14]. Methods that are handily comparable were described in [15]. However, we have not been able to identify the corresponding software. On the other hand, this paper focuses on showing the possibility of a new architecture in curve coding. We anticipate that further fine-tuning can improve its performance.

E. Distortion

The distortion that was defined in Section III combines two considerations: the total distance to a beamlet, and the degree of overlapping. Discussion on the distortion for shape coding can be found in [15]. Our definition is slightly different. However, they are close to each other.

We think our definition is more intuitive. We can provide illustrations for following two cases: 1) figures having identical χ_1 and different χ_2 and 2) figures having identical χ_2 and different χ_1. They will illustrate the different roles played by the two distortion components. However, no much space is left in the present paper. So we decide to postpone it to a future publication.

F. Connection With Recent Works in Image Compression

The proposed coder is closely related to a recent work in image coding [26], which also calls for a beamlet-like (i.e., wedgelet) data structure. Their work has a different objective: using wedgelets to improve existing wavelet coding. Even though the two methods are very similar, a careful comparison will reveal significant differences between them.

G. Hidden Markov Model

Some beamlets may be more likely to be included in a BR than other beamlets. By integrating a statistical model, such as a Hidden Markov Model that has been used in modeling wavelet coefficients, in the computing of BR, one may derive a more efficient curve coder. Notice that in JBEAM, even though tree-based coding is applied, which hopefully explores the interdependence to some extent, we still believe that a deeper integration can outperform JBEAM. This is a promising research direction.

H. Pointer(s) for Useful Implementation Ideas

In [23], as an image coding project, an algorithm was given to "search for the desired R-D operating slope," which is τ in Section III-E. The same algorithm can be adopted here. The basic idea in their approach is to design a binary search, which is well known in numerical computation. We are skipping the details in this paper.

VII. CONCLUSION

Utilizing a multiscale structure for line segments—beamlets—a tree-like coding method has been designed for generic curves. The designed method is progressive and easy to implement. Numerical experiments have demonstrated an advantage over existing curve coding software—JBIG 2. We have described in detail how to code single line segments (single beamlets) and how to unify them through tree-based coding. The overall approach is novel and has *not* been studied in the literature. We release our software [2] with the hope that more research activities can be inspired.

APPENDIX I
MATHEMATICAL FORMULA FOR A DIGITAL BEAMLET

Without loss of generality, we assume that (x_1, y_1) and (x_2, y_2) are the indices of two end pixels, satisfying the following:

- $0 < x_1 < x_2 \leq n, 0 < y_1 < y_2 \leq n$, where the size of the image is $n \times n$;
- $|x_2 - x_1| \geq |y_2 - y_1|$.

Note that the absolute values in the last inequality is not required.

In the first situation, in which multiple pixels per column are allowed, one basically enumerates all pixels whose interiors intersect the line that is from $(x_1 - 0.5, y_1 - 0.5)$ to $(x_2 - 0.5, y_2 - 0.5)$. In the second situation, in which there is at most a single pixel per column, the set of pixels that are included in a digital beamlet can be derived as

$$\{(j, y(j)) : x_1 \le j \le x_2\}$$

where

$$y(j) = \left\lceil y_1 + \frac{j - x_1}{x_2 - x_1}(y_2 - y_1) \right\rceil.$$

It actually generates the same set of pixels as a well-adopted line drawing method: Bresemham's [25] line drawing algorithm.

APPENDIX II
PROOF OF PROPOSITION 2.1

Proof: When $K = 2$, we have a length-3 sequence. Let the first bit indicate whether the center element "2" is inside the substring or not. If it is, then the two more bits will code whether the remaining two elements are inside the substring or not; if the center element is *not* inside the substring, then it takes one more bit to code whether the *prior* or the *post* element forms a single-element substring. In summary, the proposition holds at $K = 2$.

Now, induction is used. Assume that when $K = k_0$, there is a coding scheme that satisfies the conditions in Proposition 2.1. Now, consider $K = k_0 + 1$. Element 2^{k_0} becomes the central element. Use the first bit to code whether the central element is inside the substring or not. There are two cases.

1) If the central element is inside the substring, there are 2^{k_0} possibilities for the beginning and the end of the substring—including the central element itself—hence, it takes $2(k_0)$ bits to code. Overall, it takes $2(k_0) + 1$ bits.

2) If the central element is *not* inside the substring, it takes one bit to code which side of the central element (i.e., before or after) the substring occurs. After this, it becomes a coding substring problem for a length-$(2^{k_0} - 1)$ sequence, for which there is a progressive coding scheme that takes no more than $2k_0 - 1$ bits. Overall, the coding scheme takes no more than $2k_0 - 1 + 2 = 2(k_0 + 1) - 1$ bits.

This proves the first two statements.

To prove statement 3), due to the inductiveness of the above construction, we only need to prove that when 2 bits are available, the absolute distance of the end positions can be limited at 2^{K-1}. This is evident because there are $2 \cdot 2^{K-1} - 1$ elements. By knowing if the middle element is inside the interval (based on the first bit) and the side of the interval (second bit), there are 2^{K-1} possible positions of the boundary elements. Hence, we proved 3).

To prove 4), we use a standard argument from Information Theory, namely Kolmogorov entropy. Consider all the possible pairs (i, j)s, which are the starting and end positions of intervals. Assume $1 \le i < j \le 2^K - 1$. These pairs form the upper triangle of a $(2^K - 1) \times (2^K - 1)$ two-dimensional array, with (i, j) being the coordinates of the points. A coding scheme with maximum-absolute-distance distortion can be viewed as covering such a triangular region with squares, having the same side length. To achieve 2^{K-k} distortion, the side length of these squares must be 2^{K-k}. Simple calculation will reveal that it takes at least $(2^k)^2/2 + 2^k/2 = 2^{2k-1} + 2^{k-1}$ squares to cover the entire triangle. The required number of bits is given by $\lceil \log_2(2^{2k-1} + 2^{k-1}) \rceil = 2k$.

From all the above, the proposition is proved. ∎

APPENDIX III
A TUNE-UP

The point-to-beamlet distance $\rho_1(x, b)$ in (2) can be computed as follows. Let (x_1, y_1) and (x_2, y_2) be the two end pixels of beamlet b. Imagine a continuous beamlet, which is a straight line between two points: $(x_1 - 0.5, y_1 - 0.5)$ and $(x_2 - 0.5, y_2 - 0.5)$. Consider another pixel x with index (x_3, y_3). The distance between the pixel x and beamlet b is the Euclidean distance from the center of the pixel $x - (x_3 - 0.5, y_3 - 0.5)$—to the straight line given by points $(x_1 - 0.5, y_1 - 0.5)$ and $(x_2 - 0.5, y_2 - 0.5)$. The distance $\rho_1(p, b)$ can be computed by

$$\rho_1(x, b) = \frac{2 \cdot |\det(\Delta)|}{|b|} \tag{A1}$$

where

$$\det(\Delta) = \frac{1}{2} \begin{vmatrix} 1 & 1 & 1 \\ x_1 - 0.5 & x_2 - 0.5 & x_3 - 0.5 \\ y_1 - 0.5 & y_2 - 0.5 & y_3 - 0.5 \end{vmatrix}$$
$$= \frac{1}{2} \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix}$$

which is the signed area of a triangle determined by points $(x_i, y_i), i = 1, 2, 3$, and $|b|$ is the length of a line segment between two points $(x_i, y_i), i = 1, 2$

$$|b| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

The above procedure can be justified as follows. Notice that in the right hand side (R.H.S.) of (A1), the numerator is twice of the area of the triangle that is determined by the line segment b and the point x; in addition, the denominator is the length of the beamlet b. Hence, while the beamlet b is considered as a base, the ratio (on the R.H.S.) is the height of the triangle, which is equal to the point-to-line distance between the point x and the beamlet b.

When pixel x is on a digital beamlet b, one should expect $\rho_1(x, b) = 0$. In order to achieve this, a decision rule is designed before applying the formula in (A1). The intuition is that if the continuous beamlet—which is a line segment—traverses a square associated with a pixel, then the distance $\rho_1(x, b)$ is set to zero.

Following the previous notations, let (x_3, y_3) be the index of pixel x. Let (x_1, y_1) and (x_2, y_2) be the two endpoints of a digital beamlet b. The continuous beamlet is a line segment that connects two points: $(x_1 - 0.5, y_1 - 0.5)$ and $(x_2 - 0.5, y_2 - 0.5)$. The pixel x is associated with a square that is centered at point $(x_3 - 0.5, y_3 - 0.5)$ with four corners: $(x_3 - 1, y_3 - 1), (x_3, y_3 - 1), (x_3, y_3)$, and $(x_3 - 1, y_3)$. This square does *not* intersect

the continuous beamlet if the following four quantities have the same sign:

$$(V1) = \begin{vmatrix} 1 & 1 & 1 \\ x_1 - 0.5 & x_2 - 0.5 & x_3 - 1 \\ y_1 - 0.5 & y_2 - 0.5 & y_3 - 1 \end{vmatrix}$$

$$= \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 - 0.5 \\ y_1 & y_2 & y_3 - 0.5 \end{vmatrix}$$

$$(V2) = \begin{vmatrix} 1 & 1 & 1 \\ x_1 - 0.5 & x_2 - 0.5 & x_3 \\ y_1 - 0.5 & y_2 - 0.5 & y_3 - 1 \end{vmatrix}$$

$$= \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 + 0.5 \\ y_1 & y_2 & y_3 - 0.5 \end{vmatrix}$$

$$(V3) = \begin{vmatrix} 1 & 1 & 1 \\ x_1 - 0.5 & x_2 - 0.5 & x_3 \\ y_1 - 0.5 & y_2 - 0.5 & y_3 \end{vmatrix}$$

$$= \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 + 0.5 \\ y_1 & y_2 & y_3 + 0.5 \end{vmatrix} \quad \text{and}$$

$$(V4) = \begin{vmatrix} 1 & 1 & 1 \\ x_1 - 0.5 & x_2 - 0.5 & x_3 - 1 \\ y_1 - 0.5 & y_2 - 0.5 & y_3 \end{vmatrix}$$

$$= \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 - 0.5 \\ y_1 & y_2 & y_3 + 0.5 \end{vmatrix}.$$

Simplifying the above functions will render an efficient algorithm to determine whether x is on the beamlet b. The function $\rho_1(x, b)$ is modified as follows:

$$\rho_1(x, b) = \begin{cases} 0, & \text{if quantities } (V1), (V2), (V3), \\ & \text{and } (V4) \text{ have the same sign} \\ (A1), & \text{otherwise.} \end{cases}$$

ACKNOWLEDGMENT

REFERENCES

[1] Data Web Site [Online]. Available: http://www.ee.columbia.edu/~luoht/research/rvSeg/

[2] Jbeam Software [Online]. Available: http://www.isye.gatech.edu/~xiaoming/jbeam2004/

[3] S. M. Aghito and S. Forchhammer, "Contex based coding of binary shapes by object boundary straightness analysis.," in *Proc. Data Compression Conf.*, Mar. 2004, pp. 399–408.

[4] A. Averbuch, R. R. Coifman, D. L. Donoho, M. Israeli, and J. Walden, "Fast slant stack: A notion of radon transform for data in a Cartesian grid which is rapidly computable, algebraically exact, geometrically faithful and invertible," Stanford Univ., Stanford, CA, http://www-stat.stanford.edu/donoho/Reports/200I/FastSlantStack.pdf, Tech. Rep., 2001.

[5] S. Biswas, "Contour coding through stretching of discrete circular arcs by affine transformation," *Pattern Recognit.*, vol. 34, pp. 63–77, 2001.

[6] L. Breiman, J. Friedman, R. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, CA: Wadsworth, 1983.

[7] R. R. Coifinan and M. V. Wickerhauser, "Entropy-based algorithms for best-basis selection," *IEEE Trans. Inf. Theory*, vol. 38, no. 2, pp. 713–718, Apr. 1992.

[8] D. L. Donoho, "Wedgelets: nearly minimax estimation of edges," *Ann. Stat.*, vol. 27, pp. 859–897, 1999.

[9] D. L. Donoho and X. Huo, Beamlets and multiscale image analysis, in In Multiscale and Multiresolution Methods, Lecture Notes in Computational Science and Engineering, vol. 20, 2001.

[10] M. Etoh, J. Ostermann, and T. Sikora, "Editorial: Special issue on shape coding for emerging multimedia applications," *Signal Process.: Image Commun.*, vol. 15, no. 7/8, pp. 581–583, 2000.

[11] H. Freeman, "On the coding of arbitrary geometric configurations," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 260–268, Jun. 1961.

[12] ——, "Application of the generalized chain coding scheme to map data processing," in *Proc. IEEE Comput. Soc. Conf. Pattern Recognition and Image Processing*, May 1978, pp. 220–226.

[13] D. Hearn and M. P. Baker, *Computer Graphics: C Version*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[14] P. G. Howard *et al.*, "The emerging jbig2 standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, no. 7, pp. 838–848, Nov. 1998.

[15] A. K. Katsaggelos, L. P. Kondi, J. Ostermann, F. W. Meier, and G. M. Schuster, "Mpeg-4 and rate-distortion-based shape-coding techniques," *Proc. IEEE*, vol. 86, no. 6, pp. 1126–1154, Jun. 1998.

[16] J. Kim, A. C. Bovik, and B. L. Evans, "Generalized predictive binary shape coding using polygon approximation," *Signal Process.: Image Commun.*, vol. 15, pp. 643–663, 2000.

[17] J. Koplowitz and J. DeLeone, "Hierarchical representation of chain-encoded binary image contours," *Comput. Vis. Image Understand.*, vol. 63, no. 2, pp. 344–352, 1996.

[18] C. C. Lu and J. G. Dunham, "Highly efficient coding schemes for contour lines based on chain code representation," *IEEE Trans. Commun.*, vol. 39, no. 10, pp. 1511–1514, Oct. 1991.

[19] P. Meer, A. Sher, and A. Rosenfeld, "The chain pyramid: Hierarchical contour processing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 4, pp. 363–376, Apr. 1990.

[20] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 243–250, Jun. 1996.

[21] G. M. Schuster and A. K. Katsaggelos, "An optimal polygonal boundary encoding scheme in the rate distortion sense," *IEEE Trans. Image Process.*, vol. 7, no. 1, pp. 13–26, Jan. 1998.

[22] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Process.*, vol. 41, no. 12, pp. 3445–3462, Dec. 1993.

[23] R. Shukla, P. L. Dragotti, M. N. Do, and M. Vetterli, "Rate-distortion optimized tree structured compression algorithms for piecewise smooth images," *IEEE Trans. Image Process.*, vol. 14, no. 3, pp. 343–359, Mar. 2005.

[24] D. S. Taubman and M. W. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards, and Practice*. Boston, MA: Kluwer, 2002.

[25] R. Ulichney, "Bresenham-style scaling," in *Proc. IS&T Annu. Conf.*, 1993, pp. 101–103.

[26] M. B. Wakin, J. K. Romberg, H. Choi, and R. G. Baraniuk, Wavelet-domain approximation and compression of piecewise smooth images, to be published.

[27] H. Wang, G. M. Schuster, A. K. Katsaggelos, and T. N. Pappas, "An efficient rate-distortion optimal shape coding approach utilizing a skeleton-based decomposition," *IEEE Trans. Image Process.*, vol. 12, no. 10, pp. 1181–1193, Oct. 2003.

[28] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, Jun. 1987.

Xiaoming Huo received the B.S. degree in mathematics from the University of Science and Technology, China, in 1993, and the M.S. degree in electrical engineering and the Ph.D. degree in statistics from Stanford University, Stanford, CA, in 1997 and 1999, respectively.

He is an Assistant Professor with the School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta. His research interests include statistics and multiscale image analysis.

Dr. Huo received first prize in the 30th International Mathematical Olympiad (IMO), which was held in Braunschweig, Germany.

Jihong Chen received the B.E. degree in biomedical engineering from Tsinghua University, China, in 1999, and the Ph.D. degree in industrial and systems engineering from the Georgia Institute of Technology, Atlanta, in 2004.

Currently, she is Postdoctoral Research Fellow with the Biomedical Imaging Technology Center, Emory University, Atlanta. Her research interests include statistical signal processing and machine learning.