An improved normal-mesh-based image coder

Un codeur d'image à base de maille normale amélioré

Di Xu and Michael D. Adams*

The normal-mesh-based image coder of Jansen, Baraniuk, and Lavu (JBL) is studied. First, the JBL coder is introduced, and several of its shortcomings are identified. Then, to address these shortcomings, three modifications to this coder are proposed, namely, the use of a data-dependent base mesh, an integer representation for normal/vertical offsets, and a different scan-conversion scheme based on bicubic interpolation. Experimental results show that these proposed changes lead to improved coding performance in terms of both objective and subjective image-quality measures. In particular, the use of a data-dependent base mesh helps to locate horizons more quickly and preserve image edges better. The number of bits required to encode the normal/vertical offsets is reduced by representing this information with integers (as opposed to real numbers). Lastly, bicubic interpolation is found to yield higher-quality image reconstructions, while still maintaining sharp edges.

Le codeur d'image à base de maille normale de Jansen, de Baraniuk, et de Lavu (JBL) est etudié. D'abord, le codeur de JBL est présenté et plusieurs de ses points faibles sont identifiés. Puis, pour adresser ces points faibles, on propose trois modifications à ce codeur, à savoir, l'utilisation d'une maille de base dépendant des données, une représentation par nombres entiers des excentrages normaux/verticaux, et un arrangement différent de balayage-conversion basé sur l'interpolation bicubique. Les résultats expérimentaux prouvent que ces changements proposés mènent à l'exécution de codage améliorée en termes de mesures objective et subjective de qualité d'image. En particulier, l'utilisation d'une maille de base dépendant des données aide à localiser des horizons plus rapidement et à mieux préserver des bords d'image. Le nombre de bits requis pour coder les excentrages normaux/verticaux est réduit en représentant cette information par nombres entiers (au lieu de nombres réels). Pour finir, l'interpolation bicubique s'avère pour rapporter des reconstructions d'image de plus haute qualité, tout en maintenant les bords pointus.

Keywords: geometric image representations; image coding; normal meshes; triangulations

I Introduction

Many of today's best image coders are based on wavelet transforms [1]–[3]. Unfortunately, such coders cannot efficiently represent the geometric features inherent in images (i.e., edges). This shortcoming has led to an interest in schemes that better exploit the geometric properties of images. Such schemes include ridgelets [4], curvelets [5], edgelets [6], contourlets [7], wedgelets [8], and bandelets [9]. Recently, an image coder based on normal (triangle) meshes was proposed by Jansen, Baraniuk, and Lavu (JBL) [10], which we shall henceforth refer to as the JBL coder. Unlike wavelets, meshes are well suited to efficiently capturing the geometric information in images. For example, consider images from the so-called *horizon class* (i.e., images consisting of constant-intensity regions separated by smooth contours of discontinuity). It has been shown [10] that, under certain conditions, the normal-mesh representation employed by the JBL coder can represent horizon-class images more efficiently than wavelet-based schemes. In particular, for a representation with n coefficients, the normal-mesh and wavelet schemes achieve asymptotic error-decay rates of $O(n^{-1})$ and $O(n^{-1/2})$, respectively. Furthermore, since many real-world images behave somewhat like horizon-class images, the preceding error-decay-rate result suggests that the JBL coder has considerable promise for practical image-coding applications.

In spite of its merits, the JBL coder has some deficiencies that unnecessarily restrict its performance. In this paper (which is an extended version of [11]), we identify some of these shortcomings and propose three modifications to the coder aimed at addressing these weaknesses. Furthermore, we show that these modifications lead to improved coding performance.

*Di Xu is with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, British Columbia, Canada V6T 1Z4. E-mail: dixu@ece.ubc.ca. Michael Adams is with the Department of Electrical and Computer Engineering, University of Victoria, Victoria, British Columbia, Canada V8W 3P6. E-mail: mdadams@ece.uvic.ca.

The remainder of this paper is structured as follows. First, Section II comments on some of the notation and terminology used herein. Then, Sections III and IV, respectively, introduce the JBL coder and the implementation of it used in our work. In Section V, several shortcomings of the JBL coder are identified. This then leads us to propose, in Section VI, three modifications to the JBL coder aimed at addressing these shortcomings. The implementation of our enhanced (i.e., with our proposed modifications) coder is introduced in Section VII. Then, by way of experimental evidence provided in Section VIII, we demonstrate that each of our proposed modifications yields improved coding performance. Finally, we conclude in Section IX with a summary of our work.

II Notation and terminology

Before proceeding further, we introduce some of the notation and terminology used herein. The sets of integers, odd integers, and real numbers are denoted as $\mathbb{Z}$, $\mathbb{Z}_{\text{odd}}$, and $\mathbb{R}$, respectively. In what follows, we assume the reader to be familiar with basic geometric concepts, such as a triangulation, Delaunay triangulation (DT), and constrained DT. For more information on such concepts, the reader is referred to [12]. For an image with P bits per sample, the peak-signal-to-noise ratio (PSNR) distortion measure is defined as $\mathrm{PSNR} = 20\log_{10}([2^P - 1] / \sqrt{\mathrm{MSE}})$, where MSE corresponds to the mean-squared error.

III JBL coder

As mentioned previously, the focus of our work is the JBL coder. In the sections that follow, we provide a brief introduction to this coder. First, we describe the normal-mesh-based image representation employed by the JBL coder and explain how this representation is constructed

through a process known as mesh refinement or subdivision. Then, we discuss scan conversion and a few other details relevant to this coder.

III.A Normal-mesh-based image representation

A grayscale image is a function f of two variables x and y, where x and y correspond to position, and $z = f(x, y)$ corresponds to image intensity. In this way, an image can be viewed as a surface parameterized over the xy-plane. Thus, mesh-based techniques for representing surfaces can be used for images. In the case of the JBL coder, a normal (triangle) mesh is employed for this purpose.

For our purposes, a normal mesh [13] is a multiresolution surface representation that consists of a nested sequence of meshes $\{M_0, M_1, \ldots, M_{L-1}\}$, generated by repeated refinement of a base mesh M_0. The base mesh consists of a small number of points from the true surface (i.e., the actual image-intensity surface to be approximated). The refinement process then generates a finer mesh by adding new points from the true surface to a coarser mesh. This is done in such a way that each new vertex on the finer mesh can be expressed as a displacement from a *base point* on the coarser mesh in the direction of the base point's surface normal. In other words, the new vertices added during refinement are located where surface normals from base points on the coarser mesh pierce the true surface (i.e., new vertices are located at so-called *piercing points*). The line passing through the base point and along some search direction for locating a piercing point is called a *search line*. Since each base point and its corresponding normal direction are completely determined by the coarser mesh, only a single scalar value (i.e., a *normal offset*) is needed to identify the location of each new vertex on the finer mesh. Thus, a normal mesh can be completely characterized by its base mesh and a set of normal offsets.

As mentioned previously, an image can be represented as a surface parameterized over the xy-plane. In what follows, we refer to this plane as the *parameter plane*. Since some aspects of the JBL coder are more easily explained in terms of the parameter plane than by explicit three-dimensional (3D) geometry, we will largely adopt a parameter-plane perspective in our description of this coder. In essence, the JBL coder creates a partitioning of the parameter plane using a triangulation, and then forms an interpolant over each of the resulting triangles in order to construct a surface in three dimensions (i.e., the image surface). In what follows, unless otherwise noted, the term *vertex* will always refer to a vertex in the parameter-plane triangulation. Vertices are associated with height (i.e., z-coordinate) values. In this way, each vertex/height-value pair corresponds to a point in three dimensions. With the JBL coder, the three points associated with the vertices of each triangle are used to form a planar interpolant. By combining these interpolants, a piecewise-planar image surface in three dimensions is formed.

To represent discontinuities, the JBL coder models edges explicitly using the so-called *horizon model*. As a matter of terminology, a contour in the parameter plane that corresponds to a discontinuity contour (i.e., image edge) is called a *horizon*. A vertex that is on a horizon is said to be a *horizon vertex*, and an edge (in the triangulation) whose two endpoints are horizon vertices is said to be a *horizon edge*. The number of height values associated with a particular vertex depends on whether or not the vertex is a horizon vertex. A nonhorizon vertex is associated with only one height value, while a horizon vertex is associated with two, in order to represent the height of the image surface on both sides of the horizon. To distinguish between these two cases, each vertex is associated with a bit, called a *horizon bit*, indicating if the vertex is a horizon vertex.

III.B Mesh refinement via subdivision

Although the JBL coder employs a normal mesh, the base mesh and its subsequent refinement are more easily described in terms of the parameter-plane triangulation (introduced above) than directly in terms of the 3D mesh itself. First, the refinement process requires the notion of a true surface. Since images are essentially assumed to be piecewise constant in [10], the true surface is constructed with piecewise-constant interpolation of the original image sample data. As a matter of convenience, for the purposes of this interpolation process, the original sample values are aligned with points on the lattice $(1/2)\mathbb{Z}_{\text{odd}}$ (rather than the lattice $\mathbb{Z}$). The base mesh is associated with a particular base (i.e., initial) triangulation of the parameter plane. In the JBL scheme, the base triangulation is chosen to have four vertices, corresponding to the four corner points of the image bounding box. The refinement of the mesh then corresponds to a refinement of the parameter-plane triangulation through the addition of new vertices. In particular, refinement of the triangulation is performed by *quaternary subdivision*, whereby a new vertex is added for each edge in the triangulation, such that each triangle is split into four new triangles. Because of the manner in which the refinement of the parameter-plane triangulation is performed (i.e., using numerous normal directions), this entire process can essentially be viewed as the refinement of a normal mesh.

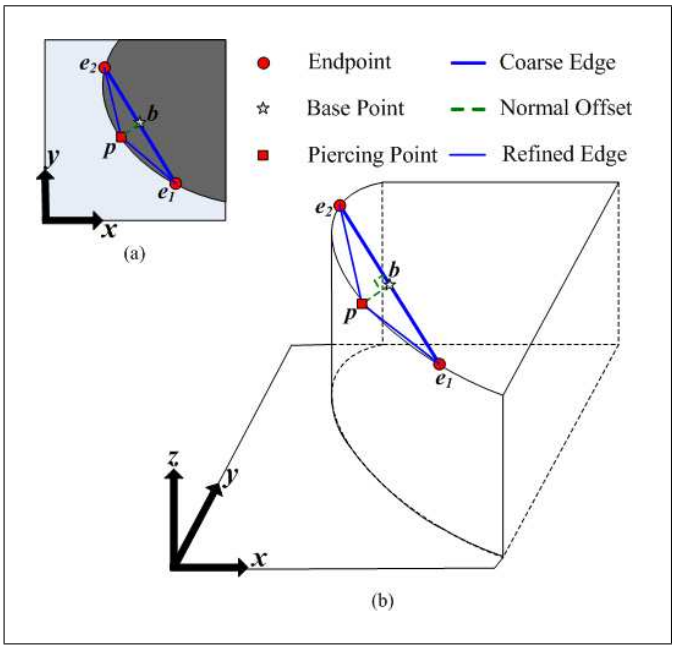


Figure 1: *Subdivision of a horizon edge along the normal direction: (a) parameter-plane view, and (b) 3D view.*

When subdivision is performed as described above, the location of the new vertex to be added for each edge is determined in one of two ways, depending on whether the edge is a horizon or nonhorizon edge. Horizon and nonhorizon edges are treated differently, since the goals of refining these two types of edges are not the same. In the horizon-edge case, the objective is to obtain a better polyline approximation of a horizon, whereas in the nonhorizon-edge case, the objective is to quickly locate new horizon vertices. We will now describe each of these two types of subdivision in more detail. To simplify the explanation that follows, some exceptional cases will not be considered in this discussion.

III.B.1 Horizon-edge subdivision

First, we consider the subdivision of a horizon edge. Since our goal in this case is to construct a refined polyline approximation of the horizon, we would like the piercing point associated with the horizon edge to be a point on the horizon. To perform the subdivision, we first define the base point as the midpoint of the edge. Then, the search line for the new horizon vertex is chosen to be normal to the edge and parallel to the xy-plane. The new vertex is added where the search line pierces the vertical surface through the horizon. The above process is illustrated in Fig. 1, where Figs. 1(a) and 1(b) are from the viewpoints of the parameter plane and 3D space, respectively.

In the diagram, a filled circle denotes an endpoint associated with a horizon vertex. The thick solid segment $\overline{e_1 e_2}$ is a coarse horizon edge. The star b at the middle of the coarse edge represents a base

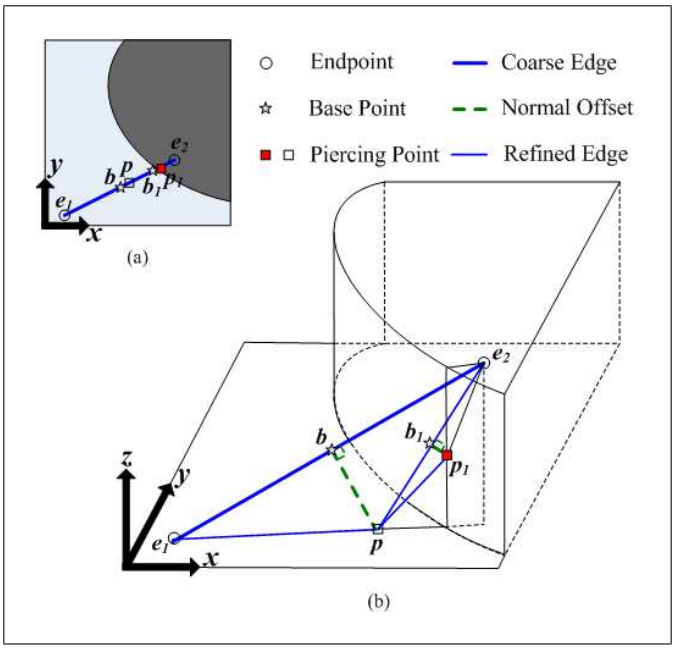


Figure 2: *Subdivision of a nonhorizon edge along the normal direction: (a) parameter-plane view, and (b) 3D view.*

point. The dotted segment $\overline{bp}$ is on the search line, which is normal to the coarse edge $\overline{e_1e_2}$ and parallel to the parameter plane. The filled square p represents a new piercing point on the horizon, the point at which the search line intersects the vertical surface through the horizon. The (signed) length of the dotted segment between the base point b and the piercing point p corresponds to the normal offset. The thin solid segments $\overline{e_1p}$ and $\overline{e_2p}$ connecting the endpoints of the coarse edge and the piercing point are refined horizon edges of the coarse horizon edge. The refined edges $\overline{e_1p}$ and $\overline{e_2p}$ form a polyline refinement of the coarse horizon edge $\overline{e_1e_2}$. Having located the piercing point, we also need to determine the horizon-bit and z-coordinate information associated with the newly added vertex. In this case (i.e., subdividing a horizon edge along the normal direction), the piercing point is the intersection of the search line with the vertical surface through the horizon. Therefore, the piercing point is always a point on the horizon. For this reason, no horizon bit is required for the new vertex. Let z_1^- and z_1^+ be the two z-coordinates associated with one endpoint of the horizon edge, and let z_2^- and z_2^+ be the two z-coordinates associated with the other endpoint of the edge. Then, the two z-coordinates z^- and z^+ associated with the new vertex are determined as $(1/2)\{\min\{z_1^-, z_1^+\} + \min\{z_2^-, z_2^+\}\}$ and $(1/2)\{\max\{z_1^-, z_1^+\} + \max\{z_2^-, z_2^+\}\}$, respectively.

III.B.2 Nonhorizon-edge subdivision

Now, we consider the subdivision of a nonhorizon edge. Before proceeding, we need to determine which of the z-coordinates associated with the two endpoints of the nonhorizon edge are appropriate to use for determining the nonhorizon edge in three dimensions. If both endpoints of the nonhorizon edge are associated with nonhorizon vertices, the z-coordinates of the endpoints are unambiguously determined. Suppose now that one of the endpoints of the nonhorizon edge is associated with a horizon vertex. Since, at most, one vertex of a nonhorizon edge can be a horizon vertex, the other endpoint of the edge must be associated with a nonhorizon vertex. Let us assume that the horizon vertex is associated with the two z-coordinates z_1^- and z_1^+ and that the nonhorizon vertex is associated with the z-coordinate z_2. The z-coordinate associated with the horizon vertex is chosen to be whichever of $\{z_1^-, z_1^+\}$ is closer to z_2. Then, the nonhorizon edge in three dimensions is completely determined by its two endpoints and their chosen z-coordinates. The base point is determined as the midpoint of this edge. The search line is through the base point, normal to the edge, and in the vertical plane containing the edge. The new point is added where the search line pierces the image surface. This process is illustrated in Fig. 2, where Figs. 2(a) and 2(b) are from the viewpoints of the parameter plane and 3D space, respectively. The unfilled circles e_1 and e_2 are the endpoints of an edge. The thick solid segment $\overline{e_1e_2}$ is a coarse nonhorizon edge. The star b represents the base point of the edge $\overline{e_1e_2}$. The normal search line is perpendicular to $\overline{e_1e_2}$ and in the vertical plane containing $\overline{e_1e_2}$. At the unfilled square p, where the search line intersects the image surface, a new nonhorizon vertex is added. Consequently, the two segments $\overline{e_1p}$ and $\overline{pe_2}$, shown by the thin solid segments, form a refinement of the coarse nonhorizon edge $\overline{e_1e_2}$. To illustrate the adaptivity of nonhorizon-edge subdivision along the normal direction, Fig. 2 also shows the refinement of a new nonhorizon edge $\overline{pe_2}$ for the next level of subdivision. Through the corresponding base point b_1, the search line is perpendicular to the edge $\overline{pe_2}$. At the filled square p_1, the normal search line intersects the image surface. A piercing point p_1 associated with a horizon vertex is located during the subdivision. By two iterations of subdivision to the coarse nonhorizon edge $\overline{e_1e_2}$, a new horizon vertex p_1 is found.

III.B.3 Exceptional cases during subdivision

As suggested earlier, some exceptional cases can occur during the refinement process. This is because, with the approach described above, the piercing point found in a normal direction does not always lead to a valid triangulation in the parameter plane. To avoid this and other problems, we must, in some exceptional circumstances, include an offset in the vertical direction, in lieu of or in addition to the normal direction. Therefore, an additional value, called the *direction value*, is required for each offset to capture which combination of normal/vertical directions is employed. Because of space constraints, the exceptional cases are not discussed further here. The interested reader is referred to the first author's master's thesis [14] for a detailed treatment of these cases.

In all of the subdivision cases, once the search line is fixed, the offset is calculated by measuring the (signed) distance between the base point and the piercing point. Offsets are signed quantities, since these displacements can be in either of two directions along the search line. In all of the subdivision cases other than the horizon-edge case along the normal direction, once a piercing point is found, this point is projected vertically onto the parameter plane to obtain a corresponding (triangulation) vertex. The horizon bit for the offset associated with this new vertex is then set to 1 if the vertex is on a horizon, and 0 otherwise. The z-coordinate associated with the new vertex is the z-coordinate of the image surface at the piercing point.

It is worth noting that the normal search direction used in the subdivision of nonhorizon edges contributes very significantly to the fast location of new horizon vertices, since the normal direction tends to point towards a nearby function discontinuity if one exists. In essence, the choice of a normal search direction makes the subdivision process adaptive to the image data (i.e., data-dependent). In contrast, if we were instead to perform these searches in the vertical direction, new vertices would always be added at the midpoints of the edges in the parameter-plane triangulation, making the fast location of new horizon vertices more difficult.

III.C Other comments on the JBL coder

The normal-mesh-based representation produced by the JBL coder is completely characterized by the base mesh, normal/vertical offsets, horizon bits, and direction values. Using this information, the corresponding mesh can be reconstructed. Since the resulting mesh representation is a surface defined on a continuous domain, a scan-conversion process is needed to convert the mesh data from the continuous domain to points on a raster grid. It is implied in [10] that planar interpolation is being used in the scan conversion by the JBL coder. By sampling the image surface on a regular grid (aligned with the centres of pixels in the parameter plane), a rasterized image is produced.

III.D Simplified JBL coder

Having introduced the JBL coder, we now take a moment to introduce a simplified version of this coder that is implicitly suggested in [10] (in the context of natural images). We refer to this simplified version

of the coder by the name JBL-S. Conceptually, the key difference between the JBL and JBL-S coders is the way in which the original image surface is formed. In the JBL-S coder, the image surface is constructed using piecewise-planar interpolation rather than piecewise-constant interpolation. Because a piecewise-planar interpolant is used, the image surface has no discontinuities, and hence no horizons exist either. Thus, the JBL-S coder is essentially the JBL coder without the horizon model. Unlike in the JBL case, horizon bits are not needed, since horizon vertices are effectively not used in the JBL-S coder. Furthermore, in the JBL-S coder, the exceptional subdivision cases mentioned earlier can never occur, and subdivision always employs a normal search direction. Since the search direction is always normal, direction values are not required. Thus, in the case of the JBL-S coder, the normal-mesh-based image representation is completely characterized by only the base mesh and normal offsets.

IV Our implementation of the JBL coder

Before proceeding further, we briefly describe our implementation of the JBL coder, which was used as the basis for our work herein. Generally, our implementation is written in MATLAB. In [10], the JBL coder is applied exclusively to piecewise-constant images, most likely because the horizons in such images can be detected with a very trivial edge detector. To apply the JBL coder to a larger class of images, however, a more sophisticated edge detector is required. In our implementation of the JBL coder, we employ a Canny edge detector [15] to assist in the identification of horizons. After the horizons have been identified, a normal-mesh-based representation is constructed, the corresponding normal/vertical offsets are quantized, and a final bit rate is estimated based on a simple, implicitly assumed coding scheme.

In our implementation, the normal/vertical offsets are quantized with a separate uniform scalar quantizer for the offsets of each subdivision level. Rather than being specified individually, all quantizer step sizes are computed from a single encoder parameter q. In particular, the step size Δ_k for the offsets of the k-th subdivision level (where $k = 0$ corresponds to the base mesh) is chosen as $\Delta_k = q2^{k-1}$ for $k \geq 1$. Here, $k \neq 0$ since no offsets are associated with the base mesh. This choice of Δ_k causes offsets from coarser levels to be weighted more heavily than those from finer levels. Such weighting is desirable since, in a normal mesh, errors in the coarser-level offsets introduce considerably more distortion than errors in the finer-level offsets, due to the fact that errors in the reconstructed vertices and their height values introduced by offset quantization propagate from coarser to finer levels of the mesh.

In [10], the authors do not attempt to estimate the bit rate required to code the data of the normal-mesh-based image representation. In our work, however, we assume a simple coding scheme for the data and determine the corresponding bit rate based on entropy estimates. For the remainder of this section, let W, H, and P denote the width, height, and number of bits per sample in the image being coded, and let L denote the number of subdivision levels. Table 1 identifies the mesh data that needs to be coded for the JBL coder. The general structure of the code stream used to encode this information is as follows. The code stream begins with a simple header which includes basic information such as W, H, P, L, and the quantization parameter q. This is followed by the base-mesh information. Since the base mesh consists simply of four vertices corresponding to the four corners of the image bounding box, the x- and y-coordinates of the vertices can be derived from W and H. Only the z-coordinates need to be included in the code stream, where P bits are used for each coordinate. Finally, the data for each subdivision level is appended to the code stream in order of increasing subdivision-level index. For each level of subdivision, the new-vertex information includes offsets, horizon bits, direction values, a scale parameter λ for offset data (to be discussed shortly), and the actual probabilities for normal-bit and direction-value data. The offsets, horizon bits, and direction values are assumed to be entropy-coded (e.g., by means of arithmetic coding [16]).

Table 1
Summary of mesh data for the JBL coder

Data name	Type	Range
Offset	$\mathbb{R}$	$\pm \max\left\{2^P - 1, \frac{W-1}{2}, \frac{H-1}{2}\right\}$
Direction value	$\mathbb{Z}$	$\{0, 1, 2\}$
Horizon bit	Boolean	$\{0, 1\}$
Base mesh (z-coordinate)	$\mathbb{Z}$	0 to $2^P - 1$

Now, we explain how we estimate the rate for the entropy-coded parts of the code stream, namely, the normal/vertical offsets, horizon bits, and direction values. First, we consider the normal/vertical offsets. Since the offsets have a symmetric, sharply peaked probability distribution with zero mean, we employ a (zero-mean) Laplacian distribution to model this information. In particular, we employ a probability density function p_{off} of the form

$$p_{\mathsf{off}}(x) = \frac{\lambda}{2} e^{-\lambda|x|}, \tag{1}$$

where λ is a scale parameter. To determine the value of λ, we measure the variance σ^2 of the offset data and then match the variance of the Laplacian distribution to σ^2, yielding the choice $\lambda = \sqrt{2}/\sigma$. Since the offsets from different subdivision levels typically have distinct distributions, the offsets are modelled on a per-subdivision-level basis. Thus, we must choose a parameter λ_l for each subdivision level l, and in our implementation 32 bits are employed to represent each λ_l. The entropy $E_{\mathsf{off},l}$ of the quantized offsets at level l can be estimated as

$$E_{\mathsf{off},l} = -\sum_{k=1}^{n} f_k \log_2 p_k, \tag{2}$$

where n is the total number of quantization bins used, f_k is the fraction of offsets quantized to the k-th bin, and p_k is the probability that an offset will be in the k-th bin (for the l-th subdivision level). For a quantization bin associated with the interval $[a, b]$, the quantity p_k is simply computed as $p_k = \int_a^b p_{\mathsf{off}}(x)dx$, where $p_{\mathsf{off}}(x)$ is as defined in (1) with $\lambda = \lambda_l$.

Next, we consider the rate estimation for the horizon-bit and direction-value data. In this case, a first-order entropy estimate is used. The first-order entropy E of a source is calculated as

$$E = -\sum_{k=1}^{n} p_k \log_2 p_k, \tag{3}$$

where n is the alphabet size and p_k is the probability of the k-th symbol. Let $E_{\mathsf{hor},l}$ and $E_{\mathsf{dir},l}$, respectively, denote the entropies of the horizon-bit and direction-value data for the l-th subdivision level. To compute $E_{\mathsf{hor},l}$ using (3), we let $n = 2$ (since the alphabet is binary), and the probabilities $\{p_k\}_{k=1}^{2}$ are set to the first-order probabilities of the actual horizon-bit data. To compute the entropy $E_{\mathsf{dir},l}$ using (3), we let $n = 3$ (since the alphabet is ternary), and the probabilities $\{p_k\}_{k=1}^{3}$ are set to the first-order probabilities of the actual direction-value data. In the case of both horizon-bit and direction-value data, the quantities $\{p_k\}_{k=1}^{n-1}$ are included in the code stream, while p_n is not, as it can be deduced from the relationship $\sum_{k=1}^{n} p_k = 1$.

Given the above results, the total number of bits R required for the coded image can be computed as

$$R = \sum_{l=1}^{L} [(E_{\mathsf{off},l} + E_{\mathsf{dir},l}) N_{\mathsf{off},l} + E_{\mathsf{hor},l} N_{\mathsf{hor},l}] + R_{\mathsf{overhead}}, \tag{4}$$

where $N_{\mathsf{off},l}$ and $N_{\mathsf{hor},l}$ are, respectively, the number of offsets and horizon bits at subdivision level l, and R_{overhead} is the number of

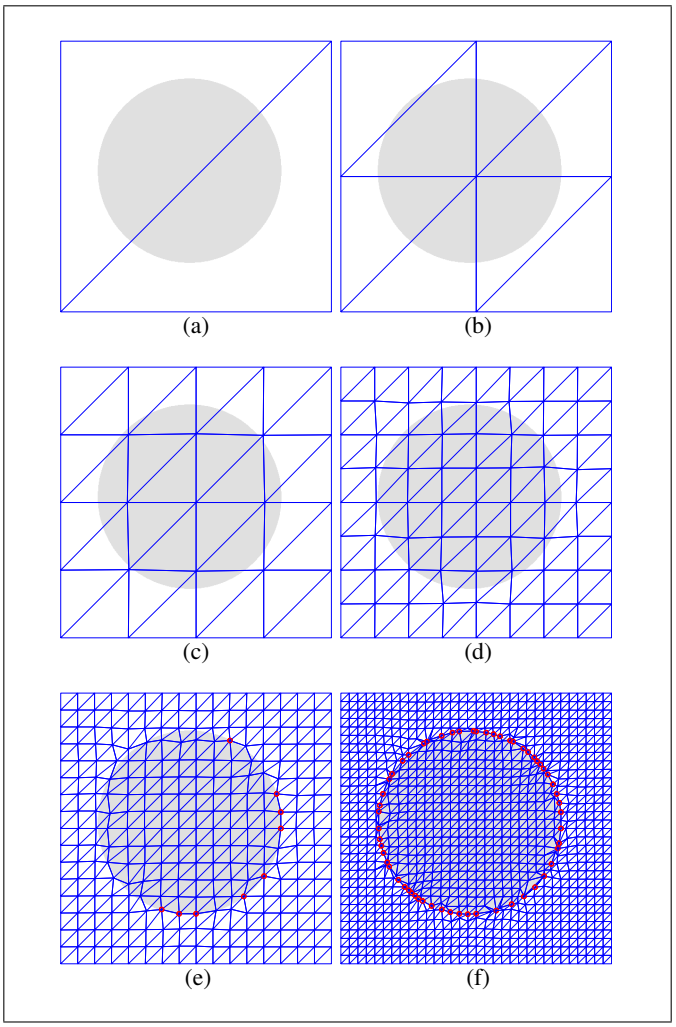


Figure 3: *Ineffectiveness of a data-dependent base mesh for the* `circle3` *image: the original image superimposed on the (a) base mesh, and the mesh after (b) one, (c) two, (d) three, (e) four, and (f) five levels of subdivision.*

bits devoted to other overhead information. The overhead information accounted for by R_{overhead} includes (a) 16 bits for each of W and H, (b) 8 bits for each of P and L, (c) 32 bits for the quantization parameter q (introduced above), (d) 32 bits representing the parameter λ_l for each subdivision level l, (e) 32 bits for the actual-probability information for horizon-bit data for each subdivision level, and (f) $32 \cdot 2 = 64$ bits for actual-probability information for direction-value data for each subdivision level. Since we use only first-order probabilities in the rate estimation, the rate does not depend on the data-scanning order used in the coding process.

Because of the similarities between the JBL and JBL-S coders as explained in Section III.D, the JBL-S coder is essentially implemented trivially as a special case of the JBL coder with a piecewise-planar image model. In this case, the set of horizons is empty, and consequently the horizon bits and direction values do not need to be coded.

V Shortcomings of the JBL coder

Having introduced the JBL coder, we now discuss some of its shortcomings. By understanding the weaknesses of this coder, we can gain better insight into how we might improve upon them. The modifications to the coder that we propose later are motivated by the desire to overcome these deficiencies.

V.A Choice of base mesh

One of the strengths of the JBL coder is its fast asymptotic error-decay rate for horizon-class images. This fast rate is, in part, due to the ability of the JBL coder to quickly locate new horizon vertices. Two different mechanisms are available to the JBL coder to assist in the location of horizon vertices: (1) the choice of base mesh, and (2) the adaptivity inherent in normal subdivision. Unfortunately, the JBL coder (which uses a trivial data-independent base mesh) relies solely on the second of these mechanisms in order to quickly locate enough horizon vertices to form good polyline approximations of horizons. As a result, normal subdivision usually introduces many nonhorizon vertices, and such vertices do not help to improve polyline approximations of horizons. Furthermore, many of these new nonhorizon vertices will also be positioned far away from any horizon, making a relatively smaller contribution to achieving a good surface approximation as a result. This situation is undesirable, as it ultimately leads to reduced coding efficiency. Furthermore, this degradation in coding efficiency can be especially significant at low bit rates, where performance often depends very critically on the fast location of horizon vertices. In essence, the problem here is that, by using a data-independent base mesh (as opposed to a data-dependent one), the JBL coder severely restricts its ability to quickly locate horizons.

The above problem is illustrated by way of the example shown in Fig. 3. In this example, we have the image `circle3`, consisting of a single solid-gray circle that we wish to code using the JBL coder. The figure shows the mesh obtained after each of several levels of subdivision (superimposed on the original image). In a good polyline approximation of a horizon, triangle edges should not cross the horizon; they should instead be tangential to the horizon curve. Unfortunately, even after five levels of subdivision, the resulting very dense mesh does not form a particularly good polyline approximation of the circle boundary. There is, however, good reason to believe that, with an intelligently chosen data-dependent base mesh, a better approximation of horizons can be achieved.

V.B Normal/vertical offset format

The second shortcoming of the JBL coder involves the representation it employs for normal/vertical offsets. An offset measures the distance from a base point to its associated piercing point. Since neither the base point nor the piercing point falls on an integer grid, both can be anywhere on the image surface. The offset measuring the distance between the two points is a real number.

By further observation, we notice that the image surface is not arbitrary. Because of the piecewise-constant interpolation used to generate the image surface, at least one of the x-, y-, or z-coordinates of any point on the surface is an integer. This suggests that the JBL coder might be improved by exploiting this special property of the image surface.

V.C Scan conversion

The third shortcoming of the JBL coder involves the scheme employed for scan conversion. Ideally, we desire a scan-conversion scheme that preserves both smooth regions in an image and sharp intensity changes along horizons. The piecewise-planar interpolation scheme employed by the JBL coder yields an interpolant that is smooth within each triangular domain (of the parameter-plane triangulation), but is not usually smooth at the boundaries of these domains, because of mismatches in partial derivatives along the boundaries of neighbouring domains. A higher-order interpolation scheme could improve the smoothness of the interpolant along domain boundaries. For this reason, there is quite likely room for improvement in the scan-conversion method employed by the JBL coder.

VI Proposed modifications to the JBL coder

Having identified some shortcomings of the JBL coder, we now propose three modifications to it in order to overcome these weaknesses.

As we will later show, each of these changes leads to improved coding performance.

VI.A Choice of base mesh

Our first modification to the JBL coder affects the choice of base mesh. In particular, we propose the use of an intelligently chosen data-dependent base mesh (instead of a data-independent one). As mentioned earlier, a data-dependent base mesh can help to achieve good polyline approximations of horizons using relatively few vertices. In simple terms, our base-mesh generation method uses image-edge information to produce a set of horizon vertices to be employed in the base mesh. These horizon vertices along with some extra points are then triangulated to form the base mesh. In what follows, we describe our base-mesh generation method in more detail.

Since our base-mesh generation method requires image-edge information, the first step in our method is to locate all of the edge pixels in the image. This is accomplished by using a Canny edge detector. To avoid potential problems in subsequent processing, any intersecting edges are split at their intersection points. Thus, the output of the edge-detection process is always a set of edges that do not intersect each other, except possibly at their endpoints.

Once the image edges have been found, we must select, for each edge, a subset of its pixels, called a critical set, that effectively captures its shape. To do this, curvature information for the edge is employed. Using the method of [17], we compute an estimate of the curvature of the edge at each of its constituent pixels. Initially, we select as the critical set the first and last pixels of the edge as well as every pixel whose curvature value is above a certain threshold. In this way, an edge with sharp cusps is divided into several pieces. This approach solves the potential problem of needing many pixels to form a good polyline approximation at places with large curvature. The preceding critical set is then augmented by including more pixels, such that the distance between the neighbouring pixels is smaller than the reciprocal of the local curvature of the edge. In this way, we include more pixels in regions where the edge bends sharply, and fewer pixels in regions where the edge is relatively straight.

Once the critical sets of pixels for the image edges have been determined, this information must be mapped into geometric structures in the continuous domain. For each edge, the critical set of pixels is converted into a polyline approximation of a horizon in the continuous domain. This is accomplished as follows. First, pixels from the critical set are mapped to points in the parameter plane. Then, the resulting points are joined by line segments in such a way as to maintain the same connectivity that these points have on the image edge from which they were derived. Since pixel centres are aligned on the lattice $(1/2)\mathbb{Z}_{\text{odd}}$ and horizons fall on (unit-square) pixel boundaries, at least one of the x- or y-coordinates of all horizon points will always be an integer. This preceding discrete-to-continuous-domain mapping process yields a set of polyline approximations of horizons. In what follows, let V and E denote, respectively, the set of horizon points and the line segments associated with the polyline approximations of horizons.

The last step in our base-mesh generation method is to produce a triangulation of the parameter plane from which the base mesh is trivially obtained. In particular, we want to construct a triangulation containing V as triangle vertices and E as triangle edges. Since some edge constraints are imposed on the triangulation process, we cannot use a DT (since a DT may not exist). In an attempt to obtain a triangulation with good angle properties, we use a constrained DT. Furthermore, for any given set of points and constrained segments, we would like to produce a constrained DT that is unique. If the triangulation is unique, we do not need to code the complete connectivity information for the triangulation, since a knowledge of the constrained segments alone will suffice. This is highly desirable, as any additional connectivity information that must be coded will negatively impact coding efficiency. For this reason, we employ a constrained DT with preferred directions [18] in order to ensure the uniqueness of the triangulation.

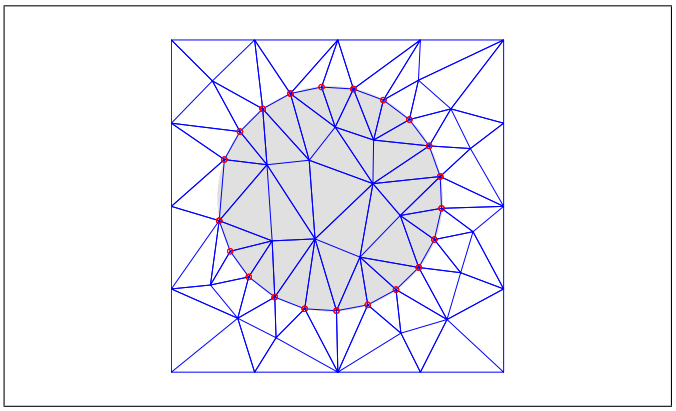


Figure 4: *Example of a data-dependent base mesh for the* `circle3` *image.*

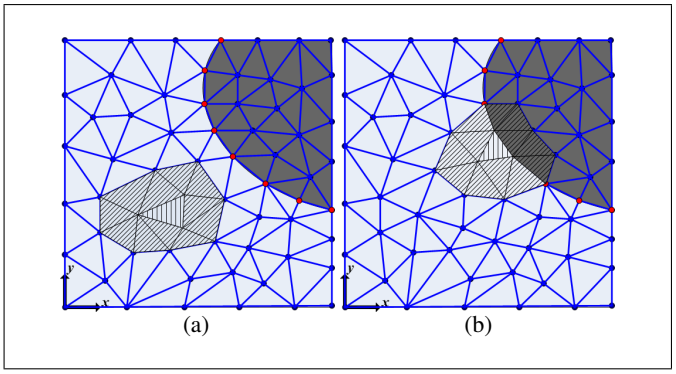


Figure 5: *Two cases for the proposed interpolation scheme: interpolation (a) away from any horizons, and (b) near a horizon.*

Although we could use a constrained DT of the points V and segments E to form the base mesh, we elect not to do so. Instead, we add some extra points V_{s} (called Steiner points) to V. Then, we perform a constrained DT of the points $V \cup V_{\text{s}}$ and segments E. This step further improves the quality of the triangulation by providing a more uniform vertex distribution and preventing sliver triangles. The Steiner points V_{s} are generated by the Triangle software [19]. Unfortunately, the Steiner points have real x- and y-coordinates. To overcome the inefficiency of coding real coordinates, the coordinates of Steiner points are rounded to the nearest pixel centres. In other words, both the x- and y-coordinates of the rounded Steiner points are elements of $(1/2)\mathbb{Z}_{\text{odd}}$. Clearly, no Steiner point could be a horizon vertex, since none of the coordinates of the Steiner points is an integer. Therefore, the rounding operation also eliminates the need to store horizon bits pertaining to the Steiner points. Furthermore, since the rounding of the Steiner-point coordinates changes the vertex geometry only very slightly, the good vertex distribution is maintained in the data-dependent base mesh.

As it turns out, because of the geometry of the vertices in our data-dependent base mesh, many edges in the constrained DT are also edges in the DT (i.e., many constraints are inactive). The large number of inactive constraints can be attributed partly to the short segments in E and good distribution of vertices in the planar straight-line graph. As an optimization, we encode only the constrained segments that affect the resulting constrained DT. In this way, the (constrained-segment) information that needs to be coded for the base mesh can be significantly reduced.

To illustrate the benefits of a data-dependent base mesh, we provide the example of a base mesh generated by our (above) method. In particular, we consider the `circle3` image from an earlier example. For this image, the base mesh produced using our method is shown in Fig. 4 (superimposed on the original image). Although the mesh contains relatively few vertices, it still manages to provide a good approximation of the horizon (i.e., circle boundary). For comparison purposes,

Table 2
Summary of features for the various coders

Coder	Image interpolant	Model	Base mesh	Offset format	Scan conversion
JBL-S	Piecewise-planar	Nonhorizon	Data-independent	$\mathbb{R}$	Piecewise-planar
JBL	Piecewise-constant	Horizon	Data-independent	$\mathbb{R}$	Piecewise-planar
Enhanced	Piecewise-constant	Horizon	Data-dependent/independent	$\mathbb{R}$ or $\mathbb{Z}$	Piecewise-planar/bicubic

recall the earlier results from Fig. 3, which show the refined meshes for the same image obtained from a data-independent base mesh. Clearly, despite having significantly fewer vertices, the base mesh in Fig. 4 generated using our method has a much better polyline approximation of the horizon than the refined mesh in Fig. 3(f) generated from a data-independent base mesh. This example clearly demonstrates that, by carefully choosing the base mesh with our method, superior polyline approximations of horizons can be obtained with fewer vertices (compared to the case in which a data-independent base mesh is employed).

VI.B Normal/vertical offset format

Our second modification to the JBL coder involves the representation of normal/vertical offsets. Since integers can be more efficiently coded than real numbers, we propose to identify each piercing point with an integer instead of a real number. Recall that an offset is a real value measuring the distance between a base point and its corresponding piercing point. Because of the piecewise-constant interpolation process used to generate the image surface, we observe that at least one of the x-, y-, or z-coordinates of each piercing point must be an integer. Therefore, along the normal/vertical search line through the base point, all possible piercing points can be identified by finding all intersections of the search line with planes of the form $x = c$, $y = c$, and $z = c$, where $c \in \mathbb{Z}$. In this way, all possible piercing points can be enumerated with an integer index. This index can be used to specify which of the possible piercing points is the actual piercing point. By using an integer index instead of a real number for each offset, coding efficiency can likely be improved.

VI.C Scan conversion

Our third modification to the JBL coder is in the interpolation scheme used for scan conversion. In short, we propose the use of a higher-order interpolant to improve the smoothness of the reconstructed images at the boundaries of triangular domains in the parameter-plane triangulation, while still maintaining sharp image edges. Our method is based on the bicubic interpolation technique described in [20, pp. 446–449], which yields C^1-continuous surfaces (i.e., surfaces with continuous first-order partial derivatives).

Since we wish to preserve sharp edges in the image, we cannot simply apply the above technique from [20] without modification, as this would have the undesirable effect of badly blurring edges. To avoid unnecessary blurring, we modify the behaviour of the preceding technique in the vicinity of horizons. In effect, this leads to two distinct cases, depending on whether or not the triangular domain being processed borders on a horizon edge. These two cases are illustrated in Figs. 5(a) and 5(b). In each case, part of the parameter-plane triangulation is shown, and the triangular domain over which we wish to form an interpolant is denoted by a vertically hatched triangle. The light-shaded and dark-shaded areas denote two different regions separated by a horizon.

In the first case, shown in Fig. 5(a), the triangular domain being processed does not border on a horizon edge. Here, we directly apply the method of [20], which generates an interpolant that passes through the three points associated with the three vertices of the triangle in the mesh and also has first-order partial derivatives that are continuous along the boundary of (as well as inside) the triangular domain. All 1-ring neighbours (i.e., diagonally hatched triangles) of the triangular domain being processed are used to determine the necessary partial-derivative information; that is, all vertex/height-value pairs in and on the boundary of the diagonally hatched regions are used for interpolation.

In the second case, shown in Fig. 5(b), the triangular domain being processed borders on a horizon edge. Here, when determining the interpolant for a particular triangular domain, we use only vertex/height-value pairs from the same side of the horizon as the triangular domain being processed. The region comprised of neighbouring triangles (i.e., the diagonally hatched region) straddles the horizon. We use only vertex/height-value pairs in and on the boundary of the diagonally hatched light-shaded regions for interpolation, since the triangular domain being processed is also from the same side of the horizon (i.e., the light-shaded as opposed to the dark-shaded region).

By combining the interpolants for each of the individual triangular domains, we obtain a complete interpolated image surface. By sampling this surface on a rectangular grid aligned to the pixel centres in the parameter plane, we generate a rasterized image.

VII Enhanced coder

In order to facilitate the further analysis of our three proposed JBL-coder modifications, we added support for these changes to our original implementation of the JBL coder, resulting in what we refer to as our enhanced coder. For convenience, we summarize the main features of the various coders (i.e., the JBL, JBL-S, and enhanced coders) in Table 2. The enhanced coder supports all combinations of data-independent/data-dependent base mesh, real/integer offsets, and planar/bicubic scan conversion. In what follows, we introduce some details regarding the enhanced coder. Since the enhanced coder is similar in many ways to the original version, we focus our attention only on the details that differ.

Table 3 shows the data that needs to be coded for the enhanced coder. The real- or integer-offset data is assumed to be entropy-coded, where the corresponding bit rate can be estimated using (2). When integer offsets are employed, the specified quantizer step-size parameter q should satisfy $q \geq 1$. If $q = 1$, quantization is effectively bypassed, and no information is discarded by quantization. The bit rate corresponding to the direction-value and horizon-bit data can be estimated using (3), as in the JBL coder. The total number of bits required for the coded image can be calculated using (4), with R_{overhead} being modified to include the extra information needed for the data-dependent base mesh (e.g., the numbers of horizon vertices and Steiner points in the base mesh, vertex locations, height values, and active-constraint segments for the constrained DT).

Consider now the coding of the base mesh. Here, we focus only on the case of a data-dependent base mesh, since the data-independent case is simply handled as described earlier in Section IV. We use 16 bits for each of the numbers of horizon vertices and Steiner points in the data-dependent base mesh. In the subsequent discussion, let W, H, and P denote the width, height, and the number of bits per sample of the original image, respectively. Since the x-, y-, and z-coordinates of vertices of the base mesh tend to have fairly uniform distributions, we choose not to use any entropy coding for this data. For the horizon vertices in the base mesh, each of the x- and y-coordinates is an element of $(1/2)\mathbb{Z}$. Furthermore, we know that if the x-coordinate is an integer,

the corresponding y-coordinate must be an element of $(1/2)\mathbb{Z}_{\text{odd}}$, and vice versa. Consequently, we need only code the (single-bit) fractional part of either the x- or y-coordinate. In our case, we choose to code only the fractional part of the x-coordinate. Thus, each of the x- and y-coordinates is represented using $\lceil \log_2 W \rceil + 1$ and $\lceil \log_2 H \rceil$ bits, respectively. In the case of Steiner points in the base mesh, we observe that the x- and y-coordinates of such points are always elements of $(1/2)\mathbb{Z}_{\text{odd}}$. Thus, we need not code the (single-bit) fractional part of these values (which is always one). So, for Steiner points, we use $\lceil \log_2 W \rceil$ and $\lceil \log_2 H \rceil$ bits to represent each x- and y-coordinate, respectively. The four corner points of the image bounding box are always chosen as vertices in the base mesh. Their x- and y-coordinates can be derived from the size of the image and therefore do not need to be coded. Each z-coordinate in the base mesh is represented with P bits. The (active) constrained line segments for the base mesh are coded as a list of pairs of vertex indices, where vertices are indexed according to their order of appearance in the code stream.

The code-stream format employed by the enhanced coder is very similar to that of the original coder. Some basic information is included in the header (e.g., W, H, etc.). Then, the x-, y-, and z-coordinates of the base mesh are coded as explained above. Finally, the data associated with different levels of subdivision are coded, starting from data associated with the smallest subdivision-level index and proceeding to the data associated with the largest subdivision-level index.

Note that, since we do not entropy-code the base-mesh information, there are probably more efficient schemes for handling this data. At high bit rates, the base-mesh information constitutes only a small fraction of the entire code stream. Consequently, the price paid for not entropy-coding the base-mesh information is relatively small. At low bit rates, however, the base-mesh information can consume a more significant fraction of the entire code stream. So, the cost of not entropy-coding the base-mesh information is more significant in this case. In spite of our choice not to entropy-code the base-mesh data, however, we still obtain reasonably good performance at low rates. We leave it as a subject of future work to explore the use of more sophisticated coding schemes for the base-mesh information (perhaps by performing differential coding of the vertex coordinates).

VIII Experimental results

Earlier, we suggested that our proposed modifications to the JBL coder would improve its performance. In the sections that follow, we support our claim through experimental evidence. Although numerous 8-bit grayscale test images were employed in our work (for details on the other test images employed in our work, see [14]), we focus our attention on the results for two representative images herein, namely, the `paw` and `peppers` images. The `peppers` image is taken from the well-known USC image database [21] and has dimensions of 512×512 pixels, while the `paw` image is our own synthetic test image with dimensions of 1024×1024 pixels.

Since we are primarily interested in low bit rates in our work, when we subsequently use qualifiers like "low" or "high" for the bit rate, these qualifiers should be understood in relative terms (i.e., relative to the range of bit rates under consideration in our study). In what follows, we evaluate the performance of each of our three proposed modifications to the JBL coder in turn.

VIII.A Choice of base mesh

To begin, we consider our proposed modification to the JBL coder of employing a data-dependent base mesh. In what follows, we refer to the JBL coder with this change by the name "XA." To assess the value of our proposed change, we compressed numerous test images at various bit rates with both the JBL and XA methods and examined the results. In what follows, we provide a representative subset of these results for the `paw` and `peppers` images. For reference purposes, we also include results obtained from the JBL-S and (in some cases)

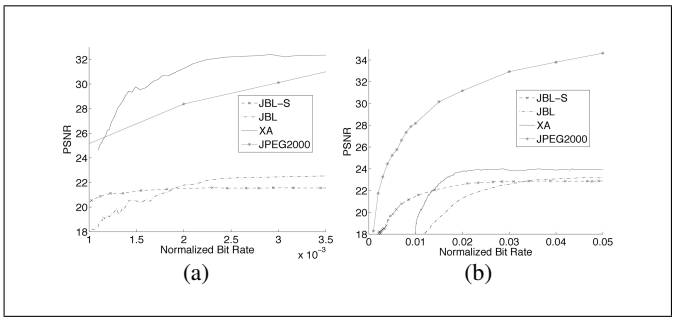


Figure 6: *Coding performance for the (a)* `paw` *and (b)* `peppers` *images using the JBL-S, JBL, XA, and JPEG-2000 methods.*

Table 3
Summary of data for the enhanced coder

Data name	Type	Range of value
Offset	$\mathbb{R}$ or $\mathbb{Z}$	$\pm \max\{2^P - 1, \frac{W+H}{2}\}$
Direction value	$\mathbb{Z}$	$\{0, 1, 2\}$
Horizon bit	Boolean	$\{0, 1\}$
Base mesh		
• No. of horizon vertices*	$\mathbb{Z}$	—
• No. of Steiner points*	$\mathbb{Z}$	—
• x^* horizon	$(1/2)\mathbb{Z}$	0 to $W - 1$
• x^* Steiner	$(1/2)\mathbb{Z}_{\text{odd}}$	0 to $W - 1$
• y^* horizon	$(1/2)\mathbb{Z}$	0 to $H - 1$
• y^* Steiner	$(1/2)\mathbb{Z}_{\text{odd}}$	0 to $H - 1$
• z	$\mathbb{Z}$	0 to $2^P - 1$
• Active line segment*	$\mathbb{Z}$	—

*Employed only when using data-dependent base mesh.

JPEG-2000 [3] coders. To maintain a fair comparison for the mesh-based methods, the number of subdivision levels for each method was chosen so that the final-mesh vertex counts would be as close to one another as possible (without giving an unfair advantage to our XA method). Since these counts can be controlled only very coarsely, it is possible only to have them match to within a factor of about three. More specifically, for the `paw` image, the JBL and JBL-S coders use six levels of subdivision, resulting in 4225 vertices, while the XA coder uses two levels of subdivision, resulting in 3308 vertices. For the `peppers` image, the JBL and JBL-S coders use seven levels of subdivision, resulting in 16 641 vertices, while the XA coder uses two levels of subdivision, resulting in 6543 vertices. In what follows, we examine the results obtained in detail.

The rate-distortion plots obtained for the `paw` and `peppers` images using the various methods are shown in Fig. 6. From these results, we can see that, at high bit rates, the XA coder outperforms the JBL and JBL-S coders, and the XA coder outperforms even the JPEG-2000 coder in the case of the `paw` image. At low bit rates, however, the XA coder can sometimes perform more poorly than the other three methods because of the rate overhead associated with the base mesh (which is not entropy-coded in our scheme). As mentioned earlier, clever schemes for coding the base mesh could reduce this overhead and significantly improve the coding efficiency of the XA method at low bit rates. From the coding results, we can also see that, at low bit rates, the JBL coder performs worse than the JBL-S coder, while at high bit rates, the JBL coder performs better than or comparably to the JBL-S coder. The superior performance of the JBL coder at high bit rates can be attributed to the greater efficiency of the horizon model, while the inferior performance at low bit rates can be attributed to the overhead of encoding horizon bits and direction values.

Now, we consider the subjective performance of the various methods. For the case of the `paw` and `peppers` images, examples of the

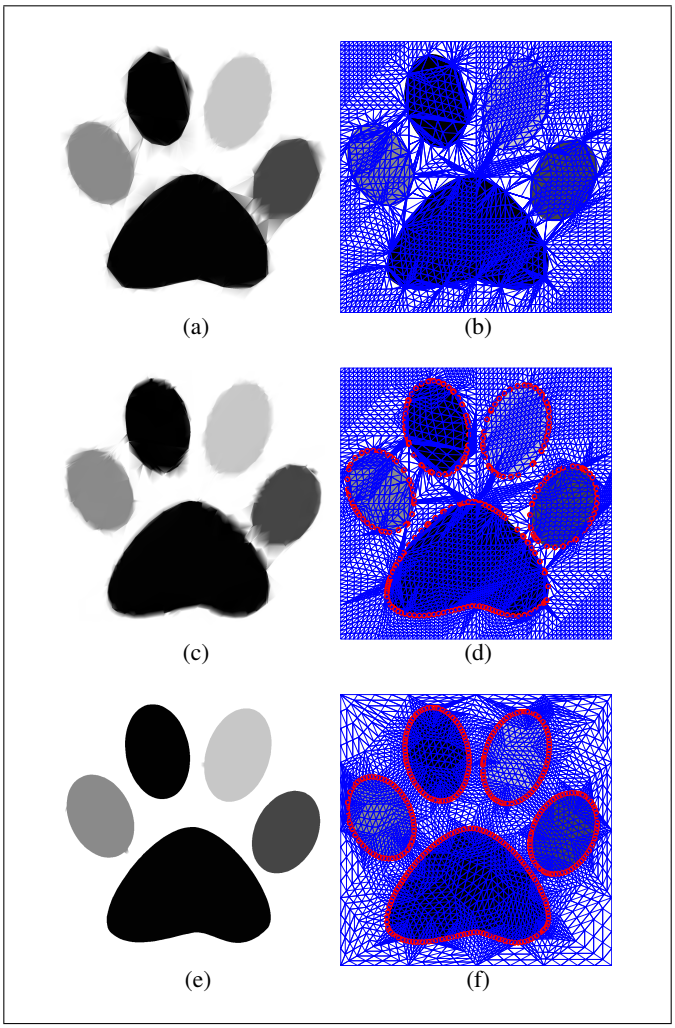


Figure 7: *Coding example for the* `paw` *image: lossy reconstructions obtained at about 400:1 compression using the (a) JBL-S, (c) JBL, and (e) XA methods, along with the corresponding final meshes employed by the (b) JBL-S, (d) JBL, and (f) XA methods.*

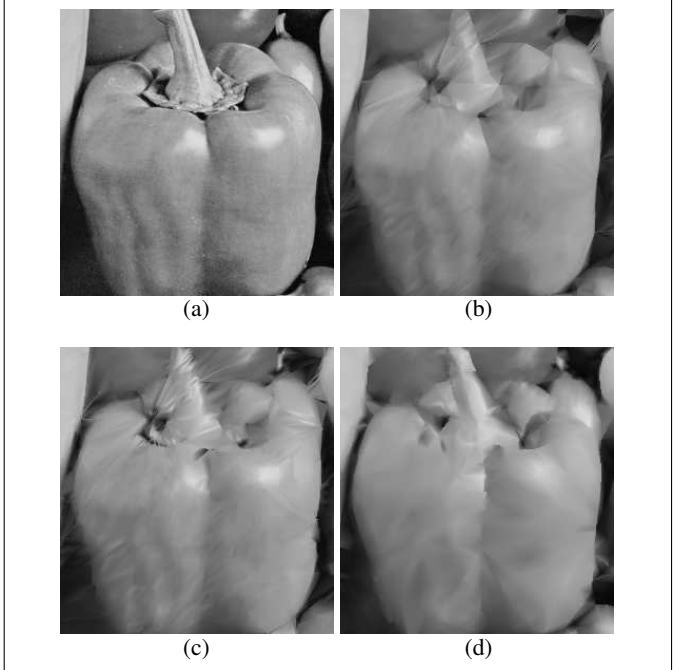


Figure 8: *Coding example for the* `peppers` *image: portions of the (a) original image and the lossy reconstructions obtained at about 29:1 compression using the (b) JBL-S, (c) JBL, and (d) XA methods.*

obtained reconstructed images are shown in Figs. 7 and 8, respectively. In the first case, the final mesh employed by each method is shown superimposed on the original image, with the horizon vertices denoted by circles. Examining the results for the `paw` image in Fig. 7, we can see that very significant edge distortions occur in the case of the JBL and JBL-S methods, while the XA scheme has little noticeable distortion. Clearly, the XA method approximates horizons much better than the JBL and JBL-S methods. Examining the reconstructions of the `peppers` image in Fig. 8, we can see that the results obtained with the XA method are comparable to those obtained with the JBL and JBL-S coders, in spite of the fact that the JBL and JBL-S methods have meshes with about 2.5 times more vertices than the XA case. On this basis, it is reasonable to conclude that the XA scheme is superior to the JBL and JBL-S schemes.

Let us again consider the subjective results for the `paw` image, including the final meshes produced by the various methods, as shown in Fig. 7. By examining the final meshes, we can see why the XA method is able to outperform the JBL and JBL-S methods. The mesh for the JBL-S method has some larger-area triangles that straddle horizons. This leads to very visually disturbing artifacts such as those near the rightmost pad of the paw in Fig. 7(a). By explicitly modelling horizons, the JBL and XA methods are able to locate horizon vertices faster and reduce distortions in large regions. Furthermore, the XA method, with a data-dependent base mesh, locates horizon vertices faster and approximates horizons better than the JBL and JBL-S schemes. As an aside, we note that the small triangular teeth occurring in the reconstructed image for the XA coder can be attributed to inaccuracies in the estimation of the location and curvature of horizons.

VIII.B Real versus integer offsets

Now, we consider our second proposed change to the JBL coder, which is to use integers rather than real numbers to represent normal/vertical offsets. To evaluate the effectiveness of our proposed change, we coded several test images with the XA coder at various bit rates, using both real and integer representations of offsets. Some representative results obtained in the case of the `paw` image (using two levels of subdivision) are shown in Fig. 9. From this graph, we can see that, at low-to-medium bit rates, integer offsets yield better results than real offsets, with the difference being more pronounced at medium rates, while in the high-bit-rate case, comparable results are obtained with integer and real offsets. It is worth noting that results similar to those above also hold in terms of subjective image quality (i.e., integer offsets are better than or as good as real offsets).

VIII.C Planar versus bicubic interpolation

Lastly, we consider our third proposed change to the coder, which is to use bicubic instead of planar interpolation for scan conversion. To assess the value of this change, both planar and bicubic interpolation were employed in the XA coder to compress numerous images at various bit rates. Some representative results obtained in the case of the `peppers` image (using two levels of subdivision) are shown in Fig. 10. From these results, we can see that bicubic interpolation outperforms planar interpolation, especially at high bit rates. In terms of subjective image quality, bicubic interpolation also leads to superior results, as it tends to better preserve smoother regions in images, without destroying sharp intensity changes at horizons.

IX Conclusion

In this paper, we proposed three modifications to the JBL coder and demonstrated through experimental results that these changes lead to improved coding performance. For example, we showed that good

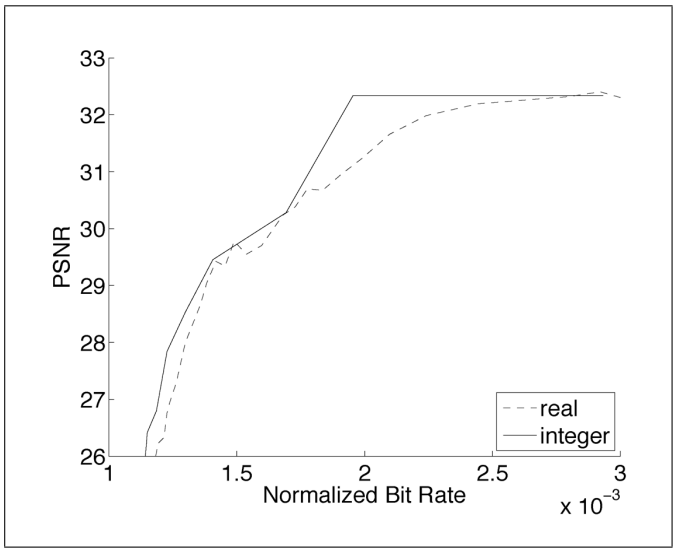


Figure 9: *Coding performance for the* `paw` *image using real and integer offsets.*

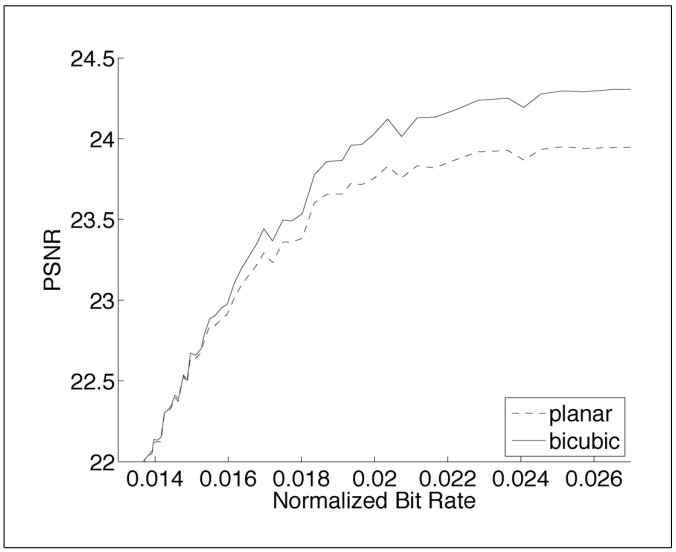


Figure 10: *Coding performance for the* `peppers` *image using planar and bicubic interpolation.*

data-dependent base meshes can help to locate horizons faster and preserve edges better. Also, we showed that using a normal/vertical-offset representation based on integers (instead of real numbers) yields superior performance. Finally, we demonstrated that, by exploiting horizon information, bicubic interpolation can be made to provide smoother reconstructed images while still maintaining sharp edges.

References

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

[2] 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, no. 3, June 1996, pp. 243–250.

[3] ISO/IEC 15444-1, *Information Technology, JPEG 2000 Image Coding System, Part 1: Core Coding System*, 2000.

[4] D.L. Donoho, "Orthonormal ridgelets and linear singularities," *SIAM J. Math. Anal.*, vol. 31, no. 5, 2000, pp. 1062–1099.

[5] E. Candès and D. Donoho, "Curvelets: A surprisingly effective nonadaptive representation of objects with edges," in *Curves and Surfaces*, ed. L.L. Schumaker et al., Nashville, Tenn.: Vanderbilt University Press, 1999.

[6] D. Donoho and X. Huo, "Combined image representation using edgelets and wavelets," in *Proc. SPIE (Wavelet Applic. Signal Image Processing VII)*, vol. 3813, Denver, Colo., Oct. 1999, pp. 468–476.

[7] M.N. Do and M. Vetterli, "The contourlet transform: An efficient directional multiresolution image representation," *IEEE Trans. Image Processing*, vol. 14, no. 12, Dec. 2005, pp. 2091–2106.

[8] D.L. Donoho, "Wedgelets: Nearly minimax estimation of edges," *Ann. Stat.*, vol. 27, no. 3, 1999, pp. 859–897.

[9] E.L. Pennec and S. Mallat, "Sparse geometric image representation with bandelets," *IEEE Trans. Image Processing*, vol. 14, no. 4, Apr. 2005, pp. 423–438.

[10] M. Jansen, R. Baraniuk, and S. Lavu, "Multiscale approximation of piecewise smooth two-dimensional functions using normal triangulated meshes," *Appl. Comput. Harmonic Analysis*, vol. 19, no. 1, 2005, pp. 92–130.

[11] D. Xu and M.D. Adams, "An improved multiscale normal-mesh-based image coder," in *Proc. IEEE Pacific Rim Conf. Communications, Computers and Signal Processing*, Victoria, B.C., Aug. 2007, pp. 50–53.

[12] J. O'Rourke, *Computational Geometry in C*, 2nd ed., Cambridge, U.K.: Cambridge University Press, 1998.

[13] I. Guskov, K. Vidimce, W. Sweldens, and P. Schroder, "Normal meshes," in *Proc. Computer Graphics*, 2000, pp. 95–102.

[14] D. Xu, "Improved subband-based and normal-mesh-based image coding," M.A.Sc. thesis, Dept. of Electrical and Computer Engineering, University of Victoria, Victoria, B.C., Aug. 2007, http://hdl.handle.net/1828/284.

[15] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 8, no. 6, Nov. 1986, pp. 679–698.

[16] I.H. Witten, R.M. Neal, and J.G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, June 1987, pp. 520–540.

[17] M. Marji and P. Siy, "A new algorithm for dominant points detection and polygonization of digital curves," *Pattern Recognition*, vol. 36, no. 10, Oct. 2003, pp. 2239–2251.

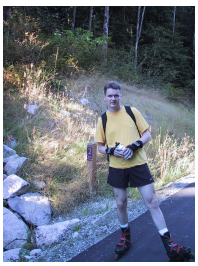
[18] C. Dyken and M.S. Floater, "Preferred directions for resolving the non-uniqueness of Delaunay triangulations," *Comput. Geometry: Theory Applic.*, vol. 34, no. 2, 2006, pp. 96–101.

[19] J.R. Shewchuk, "Delaunay refinement algorithms for triangular mesh generation," *Comput. Geometry: Theory Applic.*, vol. 22, no. 1–3, May 2002, pp. 21–74.

[20] T.Y. Yang, *Finite Element Structural Analysis*, Englewood Cliffs, N.J.: Prentice Hall, 1986.

[21] Signal and Image Processing Institute, "The USC-SIPI image database," Los Angeles, Calif.: Signal and Image Processing Institute, University of Southern California, 2007, http://sipi.usc.edu/database/.

Di Xu was born in Jilin City, Jilin, China. She received the B.S. degree in electrical engineering from Beijing Normal University, Beijing, China, in 2003, and the M.A.Sc. degree in electrical engineering from the University of Victoria, Victoria, British Columbia, Canada, in 2007. She is currently a Ph.D. candidate in the Department of Electrical and Computer Engineering at the University of British Columbia, Vancouver, British Columbia, Canada. Her research interests include multimedia signal processing, image coding, wavelets, filter banks, and digital geometric processing.



Michael D. Adams was born in Kitchener, Ontario, Canada. He received the B.A.Sc. degree in computer engineering from the University of Waterloo, Waterloo, Ontario, Canada, in 1993, the M.A.Sc. degree in electrical engineering from the University of Victoria, Victoria, British Columbia, Canada, in 1998, and the Ph.D. degree in electrical engineering from the University of British Columbia, Vancouver, British Columbia, Canada, in 2002. From 1993 to 1995, Michael was a member of technical staff at Bell-Northern Research (now Nortel Networks) in Ottawa, Ontario, Canada, where he developed real-time software for fibre-optic telecommunication systems. Since 2003, Michael has been Assistant Professor with the Department of Electrical and Computer Engineering at the University of Victoria. Michael is the recipient of a Natural Sciences and Engineering Research Council of Canada Postgraduate Scholarship. He is a voting member of the Canadian Delegation to ISO/IEC JTC 1/SC 29 (i.e., coding of audio, picture, multimedia, and hypermedia information), and has been an active participant in the JPEG-2000 standardization effort, serving as co-editor of the JPEG-2000 Part-5 standard and principal author of one of the first JPEG-2000 implementations (i.e., JasPer). His research interests include multimedia signal processing, wavelets, and multirate signal processing.