Segmentation based compression for graylevel images

Sambhunath Biswas*

Machine Intelligence Unit, Indian Statistical Institute, 203 B. T. Road, Calcutta 700 108, India

Abstract

This paper presents a segmentation based lossy image compression (SLIC) algorithm. The segmentation scheme (Biswas and Pal, Pattern Recog. Lett. 21 (2000)), entropy based and hierarchical in nature, provides sub-images of homogeneous regions. The compression algorithm encodes a graylevel image through global approximations of sub-images by 2-d Bezier–Bernstein polynomial along with corrections, if needed, over regions in sub-images by local approximation; contours by 1-d Bezier–Bernstein polynomial and texture, if present, by Huffman coding scheme using Hilbert scan on texture blocks. Order of the 2-d polynomials has been computed with the help of an image quality index (IQI). The proposed compression algorithm also examines the compression result by encoding contours through their approximation based on stretching of discrete circular arcs. Stretching is done by affine transformation. Compression results in both the cases have been compared with JPEG results. Attempts have been made to evaluate the quality of reconstructed images through a fidelity vector whose components are different objective measures.

Keywords: Segmentation; Entropy; Texture; Hilbert scan; Affine transformation; Compression; Fidelity

1. Introduction

During the last two decades, various image compression techniques have been developed. Each of these methods has its own merits and demerits, and each has its own compression ratio. Our coding methodology is segmentation based. It is seen that the segmentation based techniques [1–3] is relatively easy to understand. In this paper, we describe an image compression algorithm which uses 2-d Bezier–Bernstein polynomial function for encoding gray values in segmented sub-images. We name this algorithm sub-image based lossy image compression (SLIC). SLIC encodes images through approximation of segmented regions by 2-d Bezier–Bernstein polynomial, contours by 1-d Bezier–Bernstein polynomial and texture by Huffman coding scheme using Hilbert scan on texture blocks. The segmentation algorithm [4] used in SLIC algorithm is hierarchical in nature and very much suitable for image compression. SLIC also examines the compression result under a different contour encoding scheme, based on approximation of contours by stretching discrete circular arcs where stretching is done by affine transformation [5].

The novelty of the proposed work lies in the concept behind the approximation of sub-images for their encoding. The approximation for hierarchical segmentation [4] is different from approximation of sub-images for their encoding. The former examines the segmentation of sub-images with the assurance that more psycho visually appealing reconstruction can be made while the latter actually does the approximation. The decomposition of contours into arc and line segments at the encoding stage and their reconstruction at the decoding stage are based on methods described in Ref. [5,6]. However, we follow a different coding strategy in the proposed work for dealing with real-life images. Due to the dynamic nature of the coding scheme, arc and line segments of different types (depending on sizes) require different number of bits. Since, for different images, the types of arc and line segments reasonably vary in number; we provide an average estimate for the number of bits,

instead of actual number of bits, for contour coding for right judgement. Computation of actual number of bits for contour coding is straightforward and is described for images of two different spatial resolutions. Computing an estimate for contour coding is already in use [2,7,8]. One can consult, in this context, different contour coding techniques (see Ref. [9] and other references in it) for a comparative study. Separation and encoding of texture parts are new in concept. We have examined the feasibility of Hilbert-scan image for texture blocks. About the evaluation of the reconstructed images, we have made an attempt to provide a fidelity vector. The components of the fidelity vector are different objective measures that examine different important features of images. Thus, smaller the values of the components of the fidelity vectors of two images, larger is the resemblance between the two images.

We describe in Section 2 the details of the SLIC algorithm and the expression for compression ratio. We, also, describe in Section 2 the approximation of sub-images for encoding and reconstruction and, texture and contour coding procedures. We explain, in details, how contours can be encoded by variable length arc and line segments under two different resolutions of images. In Section 3 we have proposed a fidelity vector for the quantitative assessment of reconstructed images. The components of the fidelity vector are the indices of different objective measures. Section 4 describes the results and discussions of the proposed algorithm under two different contour coding schemes. The algorithm has been tested on images using two spatial resolutions (64×64 and 256×256). For each size, the compression ratio and quality (PSNR) of the reconstructed images have been computed. Finally, the reconstructed images have been compared with those of JPEG results both qualitatively and quantitatively. The compression ratio is found to be either superior to or comparable with those of JPEG results.The fidelity vector, also is found to have good discriminating ability between the original and reconstructed images. Section 4 also, describes the effect of the increase of spatial resolution on compression ratio and quality through a comparative study. Finally, in Section 5 we conclude about the effectiveness of the SLIC algorithm.

2. SLIC: sub-image based lossy image compression

In the approximate coding of digital still images, one is mainly concerned with the compression ratio and the fidelity of the reconstructed images. The previous techniques [1,2] uses mainly the local approximation of the segmented patches by a low degree polynomial function. Some of the implementation difficulties of the technique described in Ref. [1] have been reported in Ref. [3]. Eden et al. [10] mentioned about the approximation of the entire image, using separable properties in the least square of bidimensional polynomial. In this process, they computed the inversion of two Hankel matrices and did four matrix multiplications. It involved some matrices of the same order as that of the original image matrix. They claimed gain in computation time in the separable case over that of the non-separable case. But the feasibility of the scheme was not tested on images.

We explore in this paper, compression by globally approximating many segmented patches by a single polynomial function together with local correction, if needed. For this, all such patches should have similar gray levels. The segmented patches to be approximated by a single polynomial are extracted under a single threshold. Such segmented patches can be viewed as different surface patches of almost similar gray values and the collection of all such patches under a single threshold is defined as a sub-image. The segmentation scheme [4] recursively uses an object/background thresholding algorithm based on conditional entropy. Thresholding based segmentation strategy provides an advantage over that done by split and merge technique [11]. The latter does not provide any group of patches or regions of similar gray levels at a time. It is, therefore, preferable to choose a thresholding based segmentation strategy for coding application. However, the gray level distribution over some of the image surface patches may be such that the global approximation is not adequate for them. We call such patches, under a given threshold, *busy* patches. To overcome this difficulty, under such circumstances, a lower order (compared to that of the global approximation) polynomial function is used for local approximation of each of the residual surface patches in the sub-image. Therefore, the sub-image is reconstructed using the global surface along with the local residual surfaces for the busy patches if they are really present. Such a hybrid approximation scheme helps to improve the compression ratio. Note that exactly the same kind of approximation has been used during segmentation of an image [4]. Thus, compression can use necessary information of approximation from the segmentation of images.

Contours are coded by line and arc segments. Sometimes very small regions are found in images in the form of a texture, and region contours are found to fluctuate very rapidly so that a large number of knots or key pixels is required on contours for approximation. Under such conditions, encoding of contours by line and arc segments is not economical. Such regions can be separated out in the form of blocks from images, if they are really present. These blocks are then suitably encoded. Fig. 1 shows the 8 bit Lena image of size 256×256 and its segmentation without and with texture regions. Contour images for some hierarchical thresholds are shown in Fig. 2. In Fig. 1(b), all the gray values in thresholded regions have been replaced by the corresponding threshold value. This approach is simple and straightforward, and displays the segmented image noticeably well, provided the difference of gray values at respective pixel positions between the two images is adequate to be visually perceived. However, to perceive difference between the two images one can use completely different values as we have done for Fig. 1(c). Here, we have chosen zero gray values

Fig. 1. (a) Input Lena image (b) segmentation without texture region (c) segmentation with texture region.

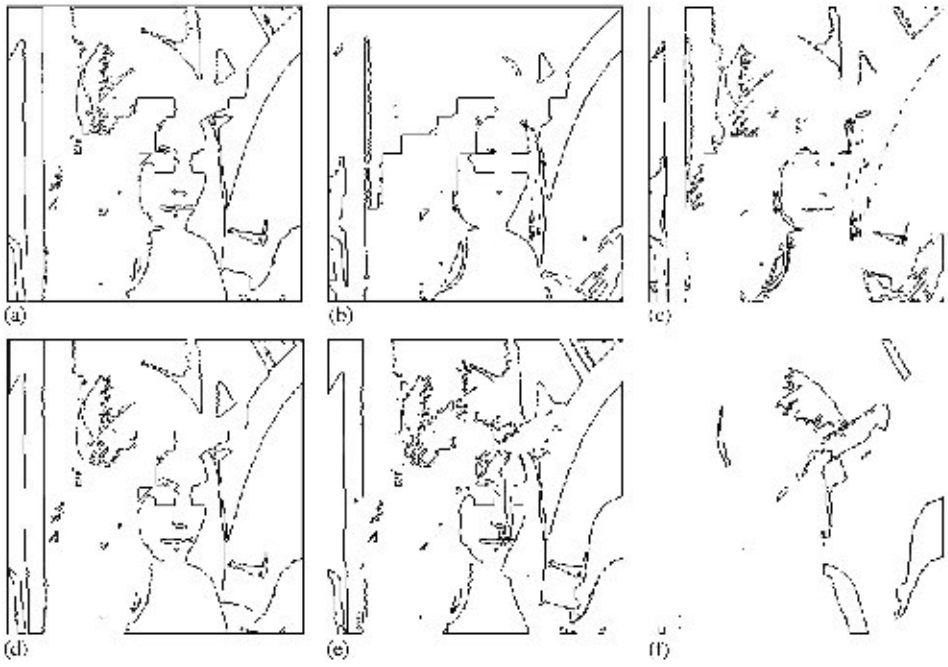

Fig. 2. Contours of Lena image for some hierarchical thresholds.

(completely dark) for textured regions. To get segmented images with texture regions, we have first found out the textured regions and then the non-textured segmented regions from the rest of the regions. Their gray values have been replaced by respective threshold values.

To see how such a segmentation scheme helps in image compression ratio, we consider the following case as an example. Suppose in a threshold band limited sub-image $F_{pq}(x, y)$ we have N_o surface patches, then for the local quadratic approximation one requires $6N_o$ coefficients. On the other hand, if we have the global quadratic approximation of the sub-image and local planar approximation of the residual surface patches, the total number of coefficients is $3N_o + 6$. For an improvement in compression ratio of the global–local approximation over the conventional local approximation we must have $6N_o > 3N_o + 6$, i.e., $N_o > 2$. This implies a positive gain in storage if the sub-image has more than two surface patches which is usually the case.

For an input image of $k - 1$ thresholds, we have k sub-images. If $N_1, N_2, \ldots, N_k$ are the number of sparse image surface patches in them, then considering variable order global approximation of k sub-images and variable order approximation for sparse residual surface patches, the compression ratio for an $M \times M$ image with L gray levels becomes,

$$R_c = \frac{\alpha\left(\sum_{i=1}^{k} C_{gi} + \sum_{i=1}^{k} C_{ie}\right) + \beta_c + \beta + \gamma}{M^2 \log_2 L}, \quad (1)$$

where $C_{g,i}$, $i = 1, 2, \ldots, k$ is the number of coefficients required for the ith sub-image for variable order global

approximation and C_{le} is the number of coefficients for regions that require residual approximation. We assign α number of bits for each coefficient. β_c is the overhead for all patches due to such correction. β is the overhead due to different orders of approximation of sub-images and γ is the number of bits for contour representation of the image.

If v is the number of bits required for encoding texture blocks, then Eq. (1) becomes

$$R_c = \frac{\alpha(\sum_{i=1}^{k} C_{gi} + \sum_{i=1}^{k} C_{le}) + \beta_c + \beta + v + \gamma}{M^2 \log_2 L}. \tag{2}$$

Note that, the number of bits for gray level approximation, in general, is

$$\beta_{gr} = \alpha \left(\sum_{i=1}^{k} C_{gi} + \sum_{i=1}^{k} C_{le} \right) + \beta_c + \beta + v. \tag{3}$$

If the global approximation itself is sufficient to meet the desired error criterion so that the approximation of residual error is not needed, then the term containing C_{le} and hence β_c in Eq. (2) do not contribute anything and under such conditions, Eq. (2) reduces to

$$R_c = \frac{\alpha \sum_{i=1}^{k} C_{gi} + \beta + v + \gamma}{M^2 \log_2 L}. \tag{4}$$

Further, when all global approximations are seen to be of fixed order and local residual approximations also are of fixed order, we get the total number N_c of coefficients as,

$$N_c = C_g k + C_l(N_1 + N_2 + \cdots + N_k), \tag{5}$$

where C_g is the number of coefficients required for global approximation of a sub-image and C_l is the number of coefficients for local residual surface approximation of each of the $N_1, N_2, \ldots, N_k$ patches. Compression ratio R_c in this case reduces to

$$R_c = \frac{\alpha N_c + v + \gamma}{M^2 \log_2 L}. \tag{6}$$

Note that when all the regions $N_1, N_2, \ldots, N_k$ in all sub-images are locally approximated for their residual surface, we do not need to store information for β and β_c. So these two terms do not contribute anything. Kunt et al. [1] observed small errors in data approximation when each surface is represented by its r pixels. These r pixels on the surface are used to recompute the coefficients. The only possible error appears in the quantization of each pixel. We have followed the same strategy and examined both the cases in our work. Since each pixel can be represented by $\log_2 L$ bits, Eq. (2) can be rewritten as

$$R_c = \frac{\log_2 L N_{pix} + \beta_c + \beta + v + \gamma}{M^2 \log_2 L}, \tag{7}$$

where N_{pix} is the total number of surface pixels. Number of bits required for gray level approximation in this case is,

$$\beta_{gr} = \log_2 L N_{pix} + \beta_c + \beta + v. \tag{8}$$

In the following we discuss about the choice of weights in the least-square approximation for the proposed coding scheme.

2.1. Approximation and choice of weights

The sub-images obtained through the segmentation scheme as described in Ref. [4] have been used for compression. The approximation algorithms are exactly the same as used for segmentation but the weights are different from unity. Details of the approximation algorithms are given in Ref. [4]. For compression, weights are chosen in a way described below. But before describing that, for completeness and clear understanding, we first briefly state the approximation problem. We have chosen the Bezier–Bernstein polynomial because our segmentation algorithm was basically designed for image compression, and Bezier–Bernstein polynomial provides a number of merits in compression and reconstruction. The Bezier–Bernstein surface is a tensor product surface and is given by

$$\begin{aligned} s_{pq}(u,v) &= \sum_{r=0}^{p} \sum_{z=0}^{q} \phi_{rp}(u) \phi'_{zq}(v) V_{rz} \\ &= \sum_{r=0}^{p} \sum_{z=o}^{q} B_{rp} D_{zq} u^r (1-u)^{p-r} v^z (1-v)^{q-z} V_{rz}, \end{aligned} \tag{9}$$

where $u, v \in [0,1]$ and $B_{rp} = p!/(p-r)!r!$, $D_{zq} = q!/(q-z)!z!$. p and q define the order of the Bezier–Bernstein surface.

To approximate an arbitrary image surface $f(x,y)$ of size $M \times M$, $f(x,y)$ should be defined in terms of a parametric surface (here s_{pq} with the parameters u and v both in $[0,1]$. Therefore, the function $f(x,y)$ can be thought in terms $g(u,v)$ where $u = (i-1)/(M-1)$; $i = 1,2,\ldots,M$ and $v = (j-1)/(M-1)$; $j = 1,2,\ldots,M$.

We choose the weighted least-square technique for estimation of parameters V_{rz} to be used for reconstruction of the decoded surface. Although, the total square error for the conventional unweighted least-square approximation may be less than that for the weighted least square, the approximation produced by the latter may be psychovisually more appealing than that by the former, provided weights are chosen judiciously. For an image, edge points are more informative than the homogeneous regions because edges are the distinct features of an image. Thus, edges should be given more emphasis while approximating an image patch and this can be done through weighted least square. Thus, the weighted squared error can be written as

$$\begin{aligned} E^2 &= \sum_u \sum_v [W(u,v)(g(u,v) - s_{pq}(u,v))]^2 \\ &= \sum_u \sum_v [W(u,v)(g(u,v) \\ &\quad - \sum_{r=0}^{p} \sum_{z=0}^{q} \phi_{rp}(u) \phi'_{zq}(v) V_{rz}]^2, \end{aligned} \tag{10}$$

where $W(u,v)$ is the weight associated with the pixel corresponding to (u,v). For $p=q$, the surface $s_{pq}(u,v)$ is defined on a square support. Since, $W(u,v)$ is the weight associated with each pixel, it can be considered constant for that pixel. Therefore, one need to find out the weight matrix before solving equations for the weighted least square. Once $W(u,v)$ is known, these equations reduce to a system of linear equations and can be solved by any conventional technique.

We emphasize that for order determination we use the unweighted approximation scheme. In the weighted least-square approximation of regions, special weights are given to boundary pixels so that the error, in the mean square sense, over the boundary is less than that in the unweighted least-square approximation. For this, we have considered the gradients of boundary pixels as their weights. One can also consider higher power of gradients. The gradients of the boundary pixels, $G(u,v)$ and hence the weights $W(u,v)$ in Eq. (10) can be calculated using the following equation.

$$W(u,v)=(G_v^2+G_u^2)^{1/2}, \tag{11}$$

where $G_u=g(u+1,v)-2g(u,v)+g(u-1,v)$ and $G_v=g(u,v+1)-2g(u,v)+g(u,v-1)$.

Image compression in our scheme is a two stage process. In stage 2, for encoding we approximate the sub-images minimizing a *weighted* least-square error with a polynomial of the same order as determined in stage 1 (for segmentation). The same order is used because the order (global and also local) of a sub-image or the nature of approximation is not expected to change due to merging of small regions. However, one can once again find the order of approximation before encoding. The reason is, the best fit surface does not necessarily represent psychovisually the most appealing (informative) surface. If we try to find the optimal order of the polynomial using weighted least square, then that optimal order is expected to be more than that for the unweighted least square. Consequently, the compression ratio will go down. Of course, the two orders cannot be widely different. Thus, there is a need to compromise. We have to find a polynomial that can approximate the surface satisfactorily and at the same time can preserve information which is psychovisually important and that is exactly what has been attempted to achieve by the proposed scheme.

2.1.1. Polynomial order determination

The order of the polynomial can be determined using either the classical approach or the image quality index (IQI) [12]. In the classical approach, the order of the Bezier–Bernstein polynomial, in one dimension is given by [13]

$$p>\frac{M_r}{2\varepsilon\delta^2}, \tag{12}$$

where M_r is the maximum value of the function present in the data set for an absolute error of approximation ε. One can search the data set and determine δ. The extension to two dimension is straightforward. On the other hand, IQI reflects the average contrast (with respect to background) per pixel in the image. So, if the original and approximated images have nearly the same IQI then the approximated image is expected to preserve the boundary contrast in the average sense. We, therefore, use very small ΔIQI between the input and approximated s b-images as an indicator of the adequacy of the polynomial order. In order to determine optimal polynomial, we increase the order of the polynomial unless the following condition is satisfied.

$$|(IQI)_{input}-(IQI)_{approximated}|\leqslant\varepsilon_a, \tag{13}$$

where ε_a is a small positive number.

For an $M\times N$ image, IQI is calculated using

$$IQI=\frac{\sum_{i=1}^{M}\sum_{j=1}^{N}|\Delta B_{ij}|/B}{MN-\sum_{i=1}^{M}\sum_{j=1}^{N}h_{ij}}, \tag{14}$$

where ΔB_{ij} is the difference of luminance B_{ij} at the (i,j)th pixel and its immediate background luminance B. It is seen that $|\Delta B_{ij}|/B$ is the contrast C_{ij} at the (i,j)th pixel according to the concept of psycho-visual perception [14].

$$c_{ij}=\left|\frac{B-B_{ij}}{B}\right|=\frac{|\Delta B|}{B}, \tag{15}$$

h_{ij} is the homogeneity of the (i,j)th pixel and is given by

$$h_{ij}=\frac{\sum_{r=1}^{8}\exp-|B_{ij}-B_r|}{8}, \tag{16}$$

where B_r indicates the intensity of a background pixel in the 3×3 neighborhood, $N_3(i,j)$ of (i,j). Thus, the background of (i,j)th pixel is defined with respect to surrounding pixels in the neighborhood without the (i,j)th pixel.

Thus, the condition in Eq. (13) follows a psycho-visual criterion. A low value of ε_a produces psycho-visually a good quality image. Note that, for an ordinary least-square approximation using polynomial surface, the error over the boundary points normally is higher than that over the interior points. Therefore, any polynomial with order determined relative to an error function measured over the boundary points, is expected to provide a good approximation for the interior points.

2.2. Texture coding

To encode the texture blocks we, first of all, Hilbert scan [15] each block. An Hilbert scanned image or simply an Hilbert image corresponding to a graylevel image, is an 1-d image with its pixels identical to those in the graylevel image through which the Hilbert curve passes. Hilbert drew a curve having the space filling property in R^2 and he found

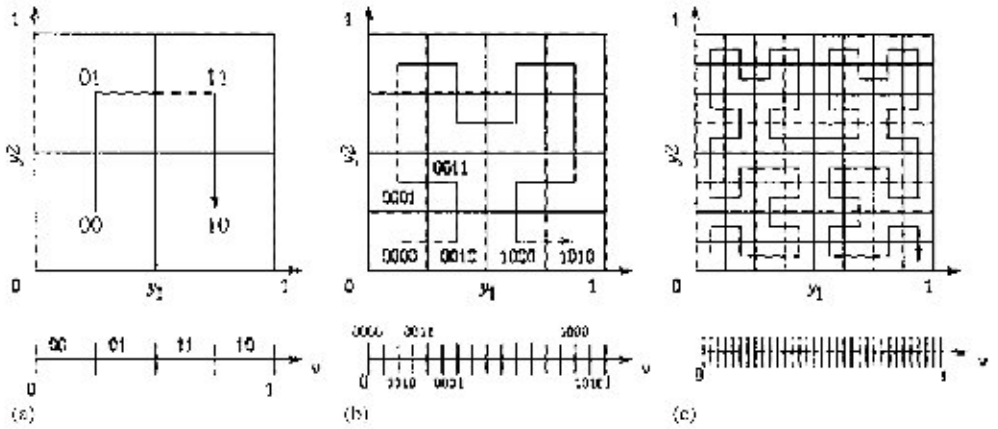

Fig. 3. Hilbert curve with different resolutions.

a one-to-one mapping between segments on the line and quadrants on the square. The merit of the curve is to pass through all points on a quadrant and always to move to the neighboring quadrant. Hilbert curves with different resolutions are shown in Fig. 3.

The efficiency of Hilbert scan has already been reported in 1-d image compression [16]. In our texture compression scheme, Hilbert scan converts each texture block into its corresponding 1-d Hilbert image. Line segments are then extracted from these Hilbert images in a straightforward way because texture blocks are all labeled by the threshold values. Also, since the blocks are textured in nature, we get tiny line segments in large number. Repetition of line segments of identical size and identical labels is very frequent. Huffman coding, therefore, provides good compression for them. Since Hilbert scan is used for texture blocks, one must be able to express the block size compatible to Hilbert image. Therefore, the choice of the window size for extraction of texture blocks can be made very easily. We have chosen the block size equal to 16. Number of bits for texture coding is the total number of bits required for all the blocks, i.e., v in Eq. (4) is given by

$$v = \sum_{i=1}^{N_{tb}} v_i, \tag{17}$$

where v_i is the number of bits for the ith texture block and N_{tb} is the total number of texture blocks.

2.3. Contour coding

Contours of segmented regions are coded using the methodologies described in Refs. [5,6]. Biswas and Pal uses, 1-dimensional Bezier–Bernstein polynomial while Biswas [5] uses stretched discrete circular arcs for encoding contour images. In encoding contours of segmented regions, they are processed once again as described below, to remove redundancy. Regions in each of the k sub-images ($k-1$, being the number of thresholds) have their own contours labeled respectively from 1 to k. All these contours need not be coded because of redundancy. We have reduced this redundancy in two stages. In the first stage, we remove the contours of all regions in the sub-image with maximum boundary or contour length compared to those in other sub-images. The reason behind this removal is that the contours of $(k-1)$ sub-images uniquely define the contour geometry for the remaining sub-image. In the second stage, the contour map for $(k-1)$ sub-images is examined to get a representation suitable for coding. Since the regions are adjacent to each other and each region is defined by its own boundary, we have "double contouring" in the contour map of an image. Note that the contour of one region defines part of other adjacent regions. In order to remove double contouring, we consider the following contour processing scheme. The part of a contour which is defined by contour of some other regions, is deleted and the non-deleted contour fragments are encoded. Let us now elaborate it.

2.3.1. Removal of double contouring

We, first of all, consider a particular region (say, the jth region of lth sub-image) Ω_{lj} of a fixed label l and examine if the regions of other labels are adjacent to Ω_{lj}. We call the contour of Ω_{lj} as primary contour and contours of the adjacent regions as adjacent contours. The primary contour is first encoded. The part of adjacent contours defined by the primary contour is then examined and deleted. Also different parts of a higher labeled contour defined by its adjacent lower labeled contours are deleted provided the deleted segments are all connected. Thus, the deletion is always done by lower labeled adjacent contours. Non-deleted contour fragments are then encoded. Lower labeled contours are encoded first. The process is repeated until all contours of different labels adjacent to a primary contour are examined for deletion and encoded. Note that the same primary contour may be considered more than once to examine and

encode all adjacent contours but the primary contour is to be encoded only once. This happens if a primary contour has more than one adjacent contours of the same label. All other primary contours having the same label are then sequentially considered. The entire process is repeated for regions of different labels. To explain the contour processing scheme more clearly, we consider a $(k+1)$ bit status word $W_s = X_s, X_1X_2, \ldots, X_k$. It indicates the status of the primary and adjacent contours. First bit, X_s in W_s always shows the status of the primary contour. $X_s = 1$ indicates that the primary contour is to be encoded along with adjacent contours but $X_s = 0$ indicates the primary contour is already encoded and only the adjacent contours need to be examined and encoded. The position of first non-zero bit in $X_1X_2, \ldots, X_k$ denotes the label of the primary contour. For an example, consider $W_s = 1,\ 111101101$. According to the status word, the primary contour has label 1 and adjacent contours have labels 2, 3, 4, 6, 7 and 9. Further the adjacent contours with labels 2, 3, 4, 6, 7, 9 must have some part of their contours defined by the primary contour. The defined part must be deleted in each case. Since $X_s = 1$, the primary contour must also be encoded. Similarly, $W_s = 0,\ 101100101$ indicates that primary contour has label 1. The primary contour must not be encoded because it has the status word $X_s = 0$. Contours with labels 3, 4, 7 and 9 are to be examined for deletion and encoded if required. Note that we consider, sequentially, all the primary contours of a fixed label. As a result, when we move on to a primary contour of label k, all the bits in W_s from 2 to $(k-1)$ are all zeros. Therefore, if N_p is the number of primary contours, the number of bits N_{bp}, required to preserve the region adjacency information, is given by

$$N_{bp} = (k+1)N_p. \tag{18}$$

2.3.2. *Encoding of primary and adjacent contours using 1-d Bezier–Bernstein polynomial: Approach 1*

Key pixels are detected on the primary contour as well as on the non-deleted contour fragments to serve as knots. Key pixels are basically points of high curvature and inflexion points. The key pixels on contours are such that an arc between any two key pixels remains always confined within a right triangle with its base as the line joining the two key pixels. As a result, between two consecutive key pixels contour fragments are decomposed either into straight line or arc segments [6,17]. Each of the arcs is approximated by a 1-d Bezier–Bernstein polynomial and so can be viewed as a Bezier–Bernstein arc. We consider the parametric representation of arcs because it is axis independent. Given the starting point, each line segment requires one point while an arc needs two points for their description. Since the selection of key pixels depend on high curvature, any segment with rapid changes of curvature will have more number of key points (dense) than a segment with less curvature change. Note that line and arc segments between knots, therefore, are of variable sizes. Obviously, the line and arc segments between key pixels have smaller length where the key pixels are dense. Thus, key pixels (dense and non-dense) captures the structure of the contour and helps to maintain curvature of the entire contour at the time of reconstruction. Now, to encode an arc e first consider the end pixel of the arc. Next, we encode the difference of co-ordinates $(\Delta x, \Delta y)$ of the pixel on the arc at the parameter value, $t = 1/2$ and the mid pixel of the base of the arc. Since an arc between two key pixels may or may not be approximated by a single quadratic Bezier–Bernstein polynomial, to ensure good approximation and encoding, we restrict the minimum and the maximum number of pixels on an arc. For a 64×64 image these numbers are taken 12 and 30 respectively, while for a 256×256 image these numbers are assumed to be 20 and 40, respectively. In other words, for a 64×64 image the length of every arc is restricted to lie between 12 and 30 while for a 256×256 image, the length of every arc is assumed to lie between 20 and 40. To find out the number of bits required to encode Δx and Δy, we consider a few steps from 1-d quadratic B–B polynomial. Position co-ordinates of the point on the arc at $t = 1/2$ are,

$$\begin{aligned} x_a &= (1-t)^2 x_o + 2t(1-t)x_1 + t^2 x_2 \\ &= \frac{x_o}{4} + \frac{x_1}{2} + \frac{x_2}{4}, \\ y_a &= (1-t)^2 y_o + 2t(1-t)y_1 + t^2 y_2 \\ &= \frac{y_o}{4} + \frac{y_1}{2} + \frac{y_2}{4}. \end{aligned} \tag{19}$$

Here, (x_o, y_o) and (x_2, y_2) are respectively the start and end pixels of an arc, and at these two points, tangents to the reconstructed arc have their point of intersection at (x_1, y_1). Since, we are using relative co-ordinates, (x_o, y_o) is always the origin of the running frame of axes and hence, we take $x_o = 0$ and $y_o = 0$. Therefore, Eq. (19) reduces to

$$\begin{aligned} x_a &= \frac{x_1}{2} + \frac{x_2}{4}, \\ y_a &= \frac{y_1}{2} + \frac{y_2}{4}. \end{aligned} \tag{20}$$

The mid point of the base of the arc is given by $(x_m = x_2/2,\ y_m = y_2/2)$. The difference thus becomes,

$$\begin{aligned} \Delta x &= x_a - x_m \\ &= \frac{x_1}{2} + \frac{x_2}{4} - x_m, \\ \Delta y &= y_a - y_m \\ &= \frac{y_1}{2} + \frac{y_2}{4} - y_m. \end{aligned}$$

Since an arc between any two key pixels remains always confined within a right triangle with its base as the line joining the two key pixels, the point of intersection of tangents at two ends of the arc also remains within this right triangle. Therefore, x_1, can take on its position anywhere between 0 and x_2, and y_1 between 0 and y_2 with respect to the running

axes of co-ordinates. Thus, we get three different cases as given below.

Case I: $x_1 = 0$

$$\Delta x = \frac{x_2}{4} - x_m = -\frac{x_2}{4}, \quad \text{since } x_m = x_2/2.$$

Case II: $x_1 = x_2$

$$\Delta x = \frac{x_1}{2} + \frac{x_2}{4} - x_m = \frac{x_2}{4}.$$

Case III: $x_1 = x_m$

$$\Delta x = \frac{x_m}{2} + \frac{x_2}{4} - x_m = 0.$$

Thus, we see that $|\Delta x|$ has its maximum equal to $x_2/4$ while its minimum equals zero. For odd x_2, we take $\lceil x_2/2 \rceil$ or $\lfloor x_2/2 \rfloor$ depending on whether x_1 is greater or less than x_m so that their difference remains small. Same is the case for y_2. Therefore, the number of bits required to encode Δx and Δy can be dynamically decided based on x_2 and y_2 respectively (end pixel of the arc). For a 64×64 image, the maximum number of pixels on an arc, we have assumed, is 30. Hence, its base is always less than 30. So the end pixel can always be encoded by 5 bits. Therefore, $\Delta x < 30/4$ which is 7.5. Similarly, $\Delta y < 7.5$. In the discrete case, we consider $\Delta x \leqslant 8$ and $\Delta y \leqslant 8$. Thus, we get the following bit requirements for an arc in

64×64 image	256×256 image
identity (line or arc): 1 bit.	identity (line or arc): 1 bit.
x_d: 5 bits;	x_d: 6 bits;
y_d: 5 bits;	y_d: 6 bits;
quadrant information: 2 bits;	quadrant information: 2 bits;
Δx: $\log_2\lceil x_2/4 \rceil$ bits;	Δx: $\log_2\lceil x_2/4 \rceil$ bits;
Δy: $\log_2\lceil y_2/4 \rceil$ bits;	Δy: $\log_2\lceil y_2/4 \rceil$; bits
sign for Δx: 1 bit;	sign for Δx: 1 bit;
sign for Δy: 1 bit;	sign for Δy: 1 bit;

Note that the number of bits used to encode of Δx and Δy varies with the number of pixels on arcs. Thus, for a 256×256 image, we need 25 bits for an arc of length 33 to 40 pixels and 23 bits for an arc of length less than or equal to 32. Number of types of arcs of 33 to 40 pixels is $40-33+1=8$ and of 20 to 32 pixels is $32-20+1=13$. Total number of bits for these types of arcs is $8*25+13*23=499$ and the total number of pixels on these types of arcs is $4*73+13*26=630$. Assuming arcs of all possible lengths are equally probable, the average bit per contour pixel on arc in a 256×256 image is $499/630 = 0.79$ bits/pixel.

For a 64×64 image, an arc of length 17 to 30 pixels needs 21 bits while 19 bits are needed for a arc of length less than or equal to 16. This gives an average of 0.97 bits/contour pixel on arc. Number of types of arcs less than or equal to 16 is $16-12+1=5$ and that greater than 16 is $30-17+1=14$.

For a line segment, we set the minimum and maximum number of pixels to 4 and 8, respectively for both 64×64 and 256×256 images. Chosen length for a line segment is small enough to maintain high accuracy of the curvature of contour lines. Here, we encode straightaway the absolute difference (x_d, y_d) between the start and end points of the line segment. Thus, we need the following bits for images of two different sizes.

64×64 image	256×256 image
identity (line or arc): 1 bit.	identity (line or arc): 1 bit.
x_d: 3 bits;	x_d: 3 bits;
y_d: 3 bits;	y_d: 3 bits;
quadrant information: 2 bits;	quadrant information: 2 bits;

This gives a total of 9 bits, i.e., a maximum of (9/4) or 2.25 bits/pixel and a minimum of (9/8) or 1.125 bits/pixel. One can also find the number of bits for line segments of all possible lengths. Here, the number of types of line segments of different lengths is $8-4+1=5$. Total number of pixels for these types of line segments is $4+5+6+\cdots+8 = 5/2(8+4)=30$. Considering all such types of line segments are equally probable, we have an average of $5*9/30$ bits or 1.5 bits for a contour pixel on line segments.

2.3.3. Starting pixels

For a 64×64 image, we consider 12 bits and for a 256×256 image 16 bits per starting pixel. Therefore, the number of bits for contour pixels can be computed using the following equations.

$$\gamma_{64\times64} = N_{bp} + 12N_{sp} + 0.97N_{ca} + 1.5N_{cl}, \tag{21}$$

$$\gamma_{256\times256} = N_{bp} + 16N_{sp} + 0.79N_{ca} + 1.5N_{cl}, \tag{22}$$

where N_{sp} is the number of starting pixels on contours. Number of contour pixels on arc and line segments are represented respectively by N_{ca} and N_{cl}.

2.3.4. Encoding of primary and adjacent contours using stretching of discrete circular arcs: Approach 2

In approach 2, a contour arc between two consecutive key pixels is approximated by stretching a local discrete circular arc. The stretching is done by affine transformation. In the first order approximation of a contour arc passing through two consecutive key pixels, we consider the affine image of one quadrant discrete circular arc whose center is at the origin, say O, defined by the point of intersection of horizontal and vertical lines through the two key pixels. The axes of reference so obtained is, therefore, local to the contour arc. The line joining the two pixels is the base of the contour arc. Consider the right triangle in which the contour arc is confined. The base of the contour arc is the hypotenuse of the right triangle. The sides of the right triangle may be rotated slowly towards the contour arc as long as all the pixels on the arc remain on the same side of these lines. These lines may be taken as the approximate tangents to the contour arc at the two consecutive positions of key pixels. We call this

triangle the characteristic triangle of the contour arc. When the sides of the right triangle turn out as the approximate tangents to the contour arc, we say the tangent configuration is known but when these lines are rotated, we say the tangent configuration is unknown because the information of their point of intersection needs to be stored. Next, we find out the characteristic triangle for one quadrant discrete circular arc. This characteristic triangle is then mapped to the characteristic triangle of the contour arc through an affine transformation. To carry out the second-order approximation, the point of maximum error is detected in the first order approximated arc. Based on this point (pixel) we get two different sets of two characteristic triangles; one for the first order approximated arc and other for the original arc. Affine transformation is then used on the set of characteristic triangles of the first order approximated arcs to map the to the sets of characteristic triangles for the original arc. For details, one can see Ref. [5].

The average number of bits for the contour pixels on arc segments in this case can be computed in the same way as described for *Approach 1*. From Ref. [5], we provide the expression for the number of bits for the arc segments for different type of approximation. We use the absolute co-ordinates for the starting point and relative co-ordinates for others (e.g., the end points of line and arc segments). The encoding of a segment, therefore, is based on the difference of co-ordinates of the end point (x_e, y_e) from its previous point (x_p, y_p) along with their respective algebraic signs. Given the previous point for a segment, we first encode the larger absolute difference between $\Delta x(=x_e - x_p)$ and $\Delta y(=y_e - y_p)$ and then the smaller one. A single bit, we take, to indicate which one is greater. For this, the larger absolute difference between Δx and Δy for all segments are first computed and this value is encoded by $log_2 M$ bits (assuming the size of the image is $M \times M$, i.e., $log_2 M$ bits convey the information of $\max\{\Delta x_i, \Delta y_i\}$) where i runs over all segments. Let $k = \max\{\Delta x_i, \Delta y_i\}$). Then we can always encode the larger difference of Δx and Δy for a segment by $\log_2 k$ bits and the smaller one by $\log_2(\Delta x + 1)$ bits or $\log_2(\Delta y + 1)$ bits depending on $\Delta x \geqslant \Delta y$ or $\Delta x < \Delta y$, respectively. The sign of Δx and Δy requires 1 bit each. For an arc, the order of approximation and configuration of tangents (known or unknown) require 1 bit each while the point (x_m, y_m) of maximum displacement from the base of arc requires $\log_2(\Delta x + 1)$ and $(\Delta y + 1)$ bits because this point can always be represented with respect to the origin of the discrete circular arc. So, we summarize the bit requirements as

identity of segment : 1 bit,
order of approximation : 1 bit,
tangent configuration : 1 bit,
Δx (for end point of the arc) : $log_2 k$ bits or $log_2(\Delta y+1)$ bits depending on $\Delta x \geqslant \Delta y$ or $\Delta x < \Delta y$,
Δy (for end point of the arc) : $\log_2(\Delta x+1)$ bits or $\log_2 k$ depending on $\Delta y < \Delta x$ or $\Delta y \geqslant \Delta x$,
to indicate larger/smaller value : 1 bit (between Δx and Δy),
sign information of Δx and Δy : 2 bits,
point of maximum displacement (x_m and y_m): $\log_2(\Delta x+1) + \log_2(\Delta y + 1)$ bits,

Therefore, for the first-order approximation the number of bits for known tangent configuration : $6 + N_{bep}$ bits,
unknown tangent configuration : $6 + N_{bep} + \log_2(\Delta x + 1) + \Delta(y + 1)$ bits,
where

$$N_{bep} = \log_2 k + \log_2(\Delta x + 1) \text{ when } \Delta x \geqslant \Delta y$$
$$= \log_2(\Delta y + 1) + \log_2 k \text{ when } \Delta x < \Delta y,$$

For the second-order approximation, the number of bits for known tangent configuration is $6 + N_{bep} + \log_2(\Delta x + 1) + \log_2(\Delta y + 1)$ bits, unknown tangent configuration is $6 + N_{bep} + 2\log_2(\Delta x + 1) + 2\log_2(\Delta y + 1)$;

To find an average number of bits per contour pixel, we consider $\log_2 k$ number of bits for both $\Delta x + 1$ and $\Delta y + 1$. Note that this is worst possible case. Since for a 64×64 image the maximum arc length is 30, $\log_2 k = 5$. Therefore, in the first-order approximation to encode an arc we need for known tangent configuration $5 + 5 + 5$ or 15 bits (note that instead of $6 + N_{bep}$ we consider $5 + N_{bep}$ because we no more need the single bit to check which absolute difference is larger) irrespective of its length and for unknown tangent configuration we need 15+8 or 23 bits when $\Delta x + 1 \leqslant 16$ and $\Delta y + 1 \leqslant 16$, and 15 + 10 or 25 bits when the arc length is greater than 16. Note that decoding Δx and Δy for the end point of the arc, one can easily find out how many bits are required (8 or 10) to decode the point of maximum displacement. Similarly, in the second-order approximation for known tangent configuration 23 bits are required when the arc length is less than or equal to 16 and 25 bits when the arc length is greater than 16. For unknown tangent configuration we require 31 bits and 35 bits respectively for the two different conditions of arc lengths.

Thus, for a 64×64 image, in the first-order approximation with known tangent configuration, the average number of bits for a contour pixel on arc segments $= 15 * 19/19 * 21 = 0.71$, while for unknown tangent configuration, this requirement is $(23 * 5 + 25 * 14)/19 * 21 = 1.16$.

In the second-order approximation with known tangent configuration, the average number of bits for a contour pixel on arc segments = 1.16 and with unknown tangent configuration, the average number of bits $= (31 * 5 + 35 * 14)/19 * 21 = 1.61$.

Similarly, for a 256×256 image, in the first-order approximation with known tangent configuration, the average number of bits for a contour pixel on arc segments $= 17 * 21/21 * 30 = 0.56$ and with unknown tangent configuration, the average number of bits $= (27 * 13 + 29 * 8)/21 * 30 = 0.92$

In the second-order approximation for known tangent configuration, the average number of bits for a contour pixel on arc segments = 0.92 and for unknown tangent configuration,

the average number of bits becomes $(37*13+41*8)/21*30=1.28$.

For line segments, we get exactly the same number of bits as in *approach* 1. Therefore, we get,

$$\gamma_{64\times64} = N_{bp} + 12N_{sp} + 0.71N_{ca10} + 1.16N_{ca11} + 1.16N_{ca20} + 1.61N_{ca21} + 1.5N_{cl} \quad (23)$$

and for a 256 × 256 image, number of bits required for contour pixels,

$$\gamma_{256\times256} = N_{bp} + 16N_{sp} + 0.56N_{ca10} + 0.92N_{ca11} + 0.92N_{na20} + 1.28N_{ca21} + 1.5N_{cl}, \quad (24)$$

where, N_{ca10} is the number of contour pixels on arc segments undergoing first order approximation with known tangent configuration,

N_{ca11} the number of contour pixels on arc segments undergoing first-order approximation with unknown tangent configuration,

N_{ca20} the number of contour pixels on arc segments undergoing second-order approximation with known tangent configuration, and

N_{ca20} the number of contour pixels on arc segments undergoing second-order approximation with known tangent configuration.

One can notice that we have used an average estimate for the number of bits in contour coding, instead of computing actual number of bits, for the computation of compression ratio. However, for encoding images, we have assigned the bits as described in our methods. Providing the estimate, instead of actual number of bits, is judicious because sometimes we get high compression and sometimes less due to the nature of the coding schemes. Neither the high compression nor the low compression does really reflect the actual merit of a scheme.

3. Quantitative assessment for reconstructed images

In order to check the quality of the reconstructed images, most of the authors compute the mean squared error (MSE); though it is clear that MSE does not always reflect the quality of visual images. A reconstructed image with low MSE may, psychovisually appear to be distorted compared to another one with high MSE. For this reason, many authors have felt the need of some other measures for the image quality assessment. Since the mechanism of understanding image quality is not yet fully known, it is very hard to devise a perfectly complete quantitative measure for quality judgment. But one can always consider a measure that depends on some important attributes (depending on local and global properties) present in the input image. We have, therefore, proposed in our investigation, a fidelity vector F_v whose components are indices of different measures. Here, in addition to MSE and PSNR, we use image correlation, homogeneity, contrast and fractal dimension to assess the quality of the reconstructed image.

We classify the quality assessment indices into two categories : (say) x and y. The classification is based on mathematical and physical features. The indices based on mathematical features take care of accuracy in approximation while the indices based on physical features take care of the preservation of physical features present in the reconstructed image. In x, we compute indices taking into account both the images (input and reconstructed) together. MSE, PSNR are in this category. Image correlation between the input and reconstructed images is also included in the category of x. In y, we compute various indices, each characterizing a different image attribute such as homogeneity, contrast and fractal dimension for the two images separately. The above indices are all concerned with pixel intensities of the image.

A good quality reconstructed image should preserve all these components in the fidelity vector of the input image. Thus, the closeness between two such fidelity vectors for the input and reconstructed images indicates the closeness between them.

Different components of the fidelity vector F_v are given below.

MSE: The mean squared error

$$MSE = \frac{Total\ squared\ error}{Number\ of\ data\ points}. \quad (25)$$

PSNR: The normal procedure to evaluate the image quality is to compute the peak signal to noise ratio (PSNR) value of the original as well as of the reconstructed image. PSNR value is defined as

$$PSNR(dB) = 10\log_{10}\frac{(L-1)^2}{MSE}. \quad (26)$$

Correlation: The coefficient of correlation ρ_{xy} for two sets of data $X=\{x_1,x_2,\ldots,x_N\}$ and $Y=\{y_1,y_2,\ldots,y_N\}$ is given by

$$\rho_{xy} = \frac{1/N\sum_{i=1}^{N} x_i y_i - \bar{x}\bar{y}}{\sqrt{1/N\sum_{i=1}^{N} x_i^2 - \bar{x}^2}\sqrt{1/N\sum_{i=1}^{N} y_i^2 - \bar{y}^2}}, \quad (27)$$

where $\bar{x}=1/N\sum_{i=1}^{N} x_i$ and $\bar{y}=1/N\sum_{i=1}^{N} y_i$. The correlation coefficient ρ_{xy} takes on values from $+1$ to -1, depending on the type and extent of correlation between the sets of data. We use correlation measure between the input and reconstructed image. This provides a measure of nearness of two images.

Homogeneity index: As a measure of homogeneity we compute an homogeneity index. This index simply calculates the second-order entropy because it provides local information about the behavior of pixel intensity change. The gray level values in an image are not independent of each other. One can consider the sequences of pixels to incorporate the dependency of pixel intensities in estimating the entropy. In order to compute the entropy of an image, the following theorem due to Shannon [14,18] can be stated.

Theorem. *Let $p(s_i)$ be the probability of a sequence s_i of gray levels of length l, where a sequence s_i of length l is defined as a permutation of l gray levels. Let us define*

$$H^{(l)} = -\frac{1}{l}\sum_i p(s_i)\log_2 p(s_i), \tag{28}$$

where the summation is taken over all gray level sequences of length l. Then $H^{(l)}$ is a monotonic decreasing function of l and $H^{(l)}_{\lim\, l\to\infty} = H$, the entropy of the image. For different values of l, we get different orders of entropy.

Case 1 : $l=1$, i.e., sequence of length one. If $l=1$, we get

$$H^{(1)} = -\sum_{i=0}^{L-1} p_i \log_2 p_i,$$

where p_i is the probability of occurrence of the gray level i. Such an entropy is a function of the histogram only and it may be called "global entropy" of the image. Therefore, different images with identical histogram would have the same $H^{(1)}$ value, irrespective of their content.

Case 2: $l=2$, i.e., sequence of length two. Hence,

$$H^{(2)} = -\frac{1}{2}\sum_i p(s_i)\log_2 p(s_i)$$

$$= -\frac{1}{2}\sum_i\sum_j p_{ij}\log_2 p_{ij}, \tag{29}$$

where s_i is a sequence of length two and p_{ij} is the probability of occurrence of the gray levels i and j. Therefore, $H^{(2)}$ can be obtained from the co-occurrence matrix. $H^{(2)}$ takes into account the spatial distribution of gray levels. Therefore, two images with identical histogram but different spatial distributions will result in different entropy, $H^{(i)}$ values. $H^{(i)}$, $i \geqslant 2$, may be called local entropy. Since the second-order entropy reflects the local behavior of image, it is expected that for a homogeneous region/patch, this measure should be low.

Contrast measure: Image quality index (IQI) from Eq. (14) is used as a measure of contrast.

$$IQI = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N}|\Delta B_{ij}|/B}{MN - \sum\sum h_{ij}}.$$

Texture measure: To compare the texturedness of the reconstructed image with the original image, we examine the fractal dimension (FD) of the reconstructed as well as of the original images. In general, fractal dimension provides a measure of irregularities and, therefore, it can be used very effectively as one of the means to compare the texture quality of two images, provided one of them is obtained after some operation on the other. This is because two images having the same fractal dimension does not necessarily mean that they have the same surface irregularities. In our case, the change in fractal dimension of the reconstructed image from that of the original image indicates the extent of damage in texture of the input image due to approximation. The concept of self-similarity can be used to estimate the fractal dimension. A bounded set A in Euclidian n-space is self-similar if A is the union of N_r distinct (non-overlapping) copies of itself scaled up or down by a ratio r. The fractal dimension D of A is given by the relation [19] $1 = N_r r^D$,i.e.,

$$D = \frac{\log(N_r)}{\log(1/r)}. \tag{30}$$

There exist several approaches to estimate the FD of an image. We have used Ref. [20] to compute the fractal dimension.

Thus, we get the fidelity vector,

$$F_v = [MSE, PSNR, \rho_{xy}, H^{(1)}, H^{(2)}, IQI, FD]^{\mathrm{T}} \tag{31}$$

4. Results and discussion

In the sub-image based lossy image compression (SLIC) algorithm, sub-images obtained through segmentation have been used for gray encoding while their contour maps are encoded after removing redundancy. For each sub-image, the order of the approximating Bezier–Bernstein polynomial is computed. We have followed the IQI based approach for order computation because of psychovisual reasons. For the Lincoln image, local correction is not needed for the residual surface of any region in any sub-image, while for both Lena and Girl images local corrections are required. Girl image is found to have local correction for 18 patches while the Lena image requires local correction for 31 patches. For the Lincoln image, we have obtained 8 sub-images corresponding to seven thresholds. Orders of the polynomials for these sub-images, computed by the IQI based approach, were found to be 2, 2, 2, 2, 2, 0, 2, 2 respectively. Determination of the orders of polynomials using the classical method requires a search for δ in Eq. (12) from the data set, corresponding to an ε which is twice the error of approximation (in fact, for graylevel images we require a 2-d version of the Eq. (12) and hence a search for δ_1 and δ_2 is required). We have seen that the orders computed by the classical approach for the sub-images of Lincoln, are more or less the same to those computed by the IQI based approach. However, this order, may sometimes, be higher than that computed by the IQI based approach. This is because of the hard constraint of ε on δ in Eq. (12).

After removing contour redundancy, knots or key pixels were detected from the contours, and segments between two key pixels were approximated by line or arc segments. A line or arc segment greater than the pre-assumed length was suitably broken up and was approximated accordingly. For the reconstruction of coded images, we have followed two different ways using the same polynomial order for each of the sub-images. The main reason behind these two reconstructions is to examine how different they are from each

Table 1
Bit requirements: contours encoded by *Approach* 1 and *Approach* 2

Image	β_{gr}		$\gamma_{64\times64}$	Total no. of bits		C.R	C.R
	Recons. 1 Eq. (3)	Recons. 2 Eq. (8)	Eqs. (21) and (23)	Recons. 1	Recons. 2	1	2
Lincoln	784	336	1537.56	2321.56	1873.56	8.82	10.93
Lincoln	784	336	1629.98	2413.78	1965.98	8.48	10.41

other as well as from the original input. Reconstructions are based on:

(1) the estimated (in the weighted least-square sense) control points for each sub-image resulting in a Bezier–Bernstein surface;
(2) equally spaced points on the estimated Bezier–Bernstein surface obtained in (1).

Contour encoding in the two different cases of reconstruction of image remains the same. Only the gray values in sub-images are encoded using the above two different ways. Each control point (coefficient), in the first case, has been encoded by 12 bits whereas in the second case, equally spaced gray points are coded using the gray level information of the image. The number of gray points (pixels) are exactly equal to the number of control points. Assuming these points to lie on a Bezier–Bernstein surface patch, we have solved $(p+1)*(p+1)$ equations to get $(p+1)*(p+1)$ control points of the surface. The Bezier–Bernstein surfaces in two cases, are not exactly identical but they are very close.

The experiments have been performed using a Silicon Graphics Indy workstation running IRIX 5.3. The workstation has MIPS RS4600, 96 MB memory and speed 132 MHz. The JPEG algorithm used is of version 6a (7 Feb. 96). All the images in our experiments have been printed by a HP LaserJet printer 5P with a resolution of 600 dpi.

4.1. Result of SLIC algorithm for 64 × 64 images

Table 1 shows the number of bits and the compression ratio required to encode the 5-bit Lincoln image when contours are encoded by *Approach* 1 (1-d B–B polynomial) and by *Approach* 2 (stretched discrete circular arcs). Since this image does not have any texture blocks, the number of bits are mainly due to gray level and contour encoding. Number of bits, β_{gr} for gray level encoding can be computed using Eqs. (3) and (8) for reconstruction 1 and reconstruction 2, while the number of bits, $\gamma_{64\times64}$ for contour encoding can be computed using equation either (21) or (23) depending on *Approach* 1 or 2. Lincoln image was found to have 442 contour pixels on line segments and 348 contour pixels on arc segments. 9 status words, each 9 bits long provided region adjacency information during decoding of Lincoln image, and the number of Starting pixels was found 38. So, the overhead due to contour encoding is $9*9+12*38$ bits or 537 bits. For gray encoding, overhead due to order of approximation from equation (3) is $\beta = 8*2 = 16$ bits (since, $\beta_c = 0$). Number of coefficients for approximation of Lincoln image is 64. Thus, we get the total bit requirements and compression ratio as shown in Table 1.

From Table 1, it is seen that for reconstruction 2, the gain in compression ratio is higher than that for reconstruction 1 roughly by 25 percent. Also, between two approaches for contour encoding, compression ratio do not change appreciably which shows both the approaches for contour coding are almost equally efficient. One can notice the total number of bits for contour coding is not an integer. This is because we have computed an average estimate for them instead of actual number of bits. Hence the total number of bits is ,also not an integer. For the quality of reconstructed images, we consider the following tables for different values of the components of the fidelity vectors.

From the evaluation Table 2, it is clear that the coefficient based reconstructions for the two different approaches are very close to each other, though the PSNR value when the contours are encoded and reconstructed by 1-d B–B polynomial is slightly higher. Other components of the fidelity vector are practically the same. This is also true for the reconstructions based on equispaced surface pixels. All the reconstructed images have different values in entropy from that of the input image. This change is due to merging of small regions in the segmentation procedure before encoding of the input image and polynomial approximation in the reconstruction process. Fractal dimension of the reconstructed images differ slightly from that of the input. This is probably due to the reason that contours of the reconstructed images are not as smooth as that of the input. Below in Fig. 4, we present reconstructed Lincoln image along with the input for visual comparison.

4.2. Results of SLIC algorithm for 256 × 256 images

In this section we present the results of 8-bit 256 × 256 images where *Approach* 1 (App. 1) and *Approach* 2 (App. 2) refer to two different approaches for contour encoding. In this case, gray values are encoded (and also regenerated) using equispaced Bezier–Bernstein surface points.

Table 2
Evaluation of reconstructed image

Components of F_v	Lincoln image				
	Input	*Approach* 1		*Approach* 2	
		Recons.1	Recons.2	Recons.1	Recons.2
MSE	0	7.438	7.884	8.061	8.586
PSNR	∞	21.388	21.135	21.038	20.764
ρxy	1.0	0.958	0.958	0.955	0.955
$H^{(1)}$	3.432	2.693	2.646	2.696	2.650
$H^{(2)}$	0.1005	0.144	0.054	0.146	0.057
IQI	6959.242	6973.531	6985.070	6980.781	6991.595
FD	2.577	2.547	2.555	2.538	2.546

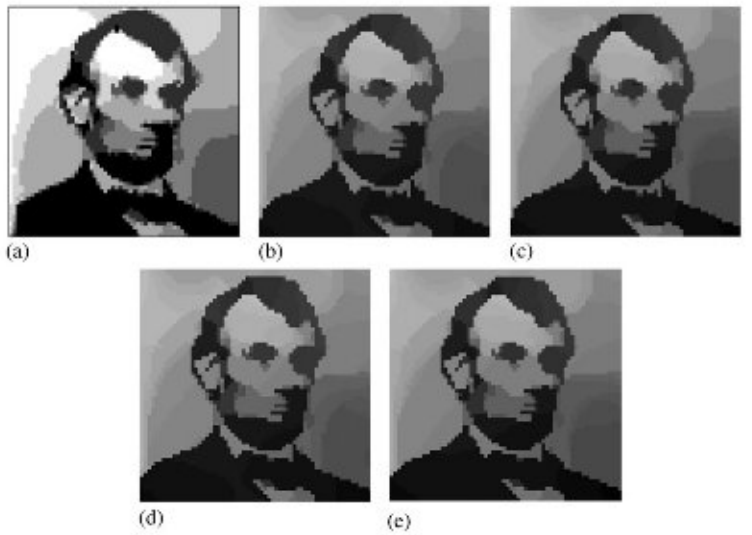

Fig. 4. Reconstruction of Lincoln image with contour encoding based on *Approach* 1 and *Approach* 2 (a)input Lincoln image (b) and (d) reconstruction from coefficients (c) and (e) reconstruction from surface points.

Table 3
Bit requirements

Image	β_{qr}	v	7256×256		Total no. of bits	
	Eq. (8)	Eq. (17)	App. 1 Eq. (22)	App. 2 Eq. (24)	App. 1	App. 2
Lena	2238	26122	19820.02	20018.24	48180.02	48378.24
Girl	1742	20123	17866.14	21131.58	39731.14	39996.58

These two images are complicated than the previous 64×64 images because these images have texture regions in them and the texture blocks as seen from the Tables 3, 4 have taken a considerably large number of bits lowering down the compression ratio. Number of contour pixels on line and arc segments for Lena image are respectively found to be 7398 and 3538 due to approximation by *Approach* 1. For *Approach* 2, 872 pixels were approximated by stretched dc arcs with known tangent configuration, 2521 pixels were approximated with first-order approximation with unknown tangent configuration and 148 pixels were approximated with second order approximation with unknown tangent configuration. Number of bits for starting pixels were found to be 5328 for 333 pixels while 600 bits were required for status

Table 4
Comparison of compression ratio

Image	Compression ratio			
	Approach 1	Approach 2	JPEG result 1	JPEG result 2
Lena	10.88	10.83	8.86	10.92
Girl	13.20	13.11	13.12	13.63

Fig. 5. Reconstruction of Lena image using surface points (a) *Approach* 1 (b) *Approach* 2 (c) JPEG result 1 (d) JPEG result 2.

words. Gray level values altogether needed 111 coefficients for global approximation and 124 coefficients for local corrections. An overhead of 198 bits were required gray level approximation. Fig. 5a, b show the reconstructed Lena images with contour encoded using *Approach* 1 and 2 for the input image as shown in Fig. 1(a). For the Girl image (Fig. 6(a)), number of pixels approximated by line and arc segments due to *Approach* 1 were 6041 and 4016, respectively. 4720 bits were required for the starting pixels while 912 bits were required for the status words. For the local corrections of 18 patches, 72 coefficients or 576 bits were required. When *Approach* 2 was used for encoding, number of pixels approximated by first order approximation with known tangent configuration was 1026 while the number of pixels approximated by first order approximation with unknown tangent configuration was 2876. Number of pixels approximated by second order approximation with unknown tangent configuration was found to be 170. Thus, total number of bits needed for approximation by arc segments became 3438.08 bits. The reconstructed images due to two different approaches for contour encoding are shown n Figs. 6(b) and (c) respectively.

To examine the performance of the proposed algorithm on 256×256 images, we have examined the compression ratio as well as compared the result with that of JPEG algorithm [21]. Note that due to different versions of JPEG algorithm, results may slightly vary. In order to compute the compression ratio by the JPEG algorithm, we have chosen the quality factor in such a way that the PSNR value of the decompressed images remain as close as possible to that of the reconstructed images due to our proposed algorithm. For the Lena image the quality factors are 50 and 30, respectively for the JPEG result 1 and JPEG result 2 (Fig. 5(c) and (d)), while for the Girl image the quality factors are 32 and 30 (Fig. 6(d) and (e)).

In order to evaluate the quality of the reconstructed images, we present below the values of the different indices of the fidelity vector F_v. To compare the performance of our method we have used the JPEG algorithm. Table 5 shows that *Approach* 1 is better than the JPEG result 1 because it has lower MSE and higher PSNR values, and same is the case with the *Approach* 2. The correlation values are comparable for all the four images which mean all the four images are almost alike. The index FD for texture measure is the same for all of them which means texture in all the four images are maintained in the same way (on the average basis). Betterment is also seen in Table 6.

4.3. *Effect of the increase of spatial resolution on compression and quality*

For the 8 bit Lena image, some of the researchers have used an image size of 256×256 while some others have used the size of 512×512. These, two different sizes are widely found in the literature. Due to this variation in size, compression is, also, found to be widely different. To get an idea how the compression and quality are affected by the increase of spatial resolution, we provide some of the results on Lena image from the recent articles.

In Ref. [22] two different compression ratios correspond to two different sizes of the structuring element used in the work. Compression ratio is 31.00 when the structuring element has the size 6×6 and 20.00 when the size is 4×4. PSNR values for the reconstructed images have not been mentioned. From the Table 7, it is found that except in one case [27], the quality (PSNR value) of the reconstructed images due to different methods are almost the same. In some of the articles, the PSNR value is not mentioned (N.M). From the work of Fisher et al. [24] the compression ratio is found to be 3.10 times larger for the size of 512×512, while from the work of Haziti et al., [30] and Lin and Venetsanopoulos [28] we see that an increase of 4.15 times is possible. In our opinion, one can obtain a compression ratio larger by a

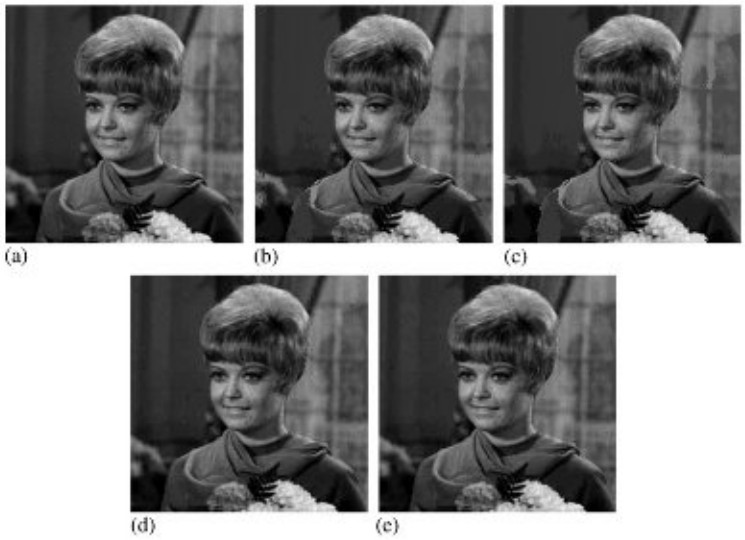

Fig. 6. Reconstruction of Girl image using surface points (a) input (b) *Approach* 1 (c) *Approach* 2 (d) JPEG result 1 (e) JPEG result 2.

Table 5
Evaluation of reconstructed Lena images

Components of F_Σ	Lena image				
	Input	App. 1	App. 2	JPEG result 1	JPEG result 2
MSE	0	96.953	131.737	129.638	142.574
PSNR	∞	28.265	26.677	27.003	26.590
ρ_{xy}	1.0	0.977	0.968	0.992	0.989
$H^{(1)}$	2.529	1.909	1.894	2.639	2.528
$H^{(2)}$	0.0021	0.0009	0.0006	0.00007	0.00046
IQI	27.616	35.700	37.097	30.197	30.372
FD	2.619	2.620	2.611	2.624	2.593

Table 6
Evaluation of reconstructed Girl images

Components of F_Σ	Girl image				
	Input	App. 1	App. 2	JPEG result 1	JPEG result 2
MSE	0	52.848	52.848	37.236	38.497
PSNR	∞	31.457	30.900	32.421	32.27
ρxy	1.0	0.987	0.985	0.991	0.991
$H^{(1)}$	1.956	1.591	1.583	2.268	2.264
$H^{(2)}$	0.043	0.018	0.004	0.144	0.197
IQI	85.952	115.263	112.572	90.619	89.553
FD	2.607	2.577	2.576	2.531	2.529

factor between 3.5 and 4.0 simply by increasing the size of an image from 256×256 to 512×512. Thus, it is expected that our developed method will provide a compression ratio in the range 38.0–43.52 for the Lena and 46.2–52.8 for Girl image respectively.

5. Conclusions

We have developed a segmentation based lossy image compression algorithm, SLIC. The algorithm, SLIC, uses a new segmentation scheme, suitable for image compres-

Table 7
Some results on Lena image due to increase in spatial resolution

Image size	Article	Principle of coding	Compression ratio	PSNR in db
512 × 512	[23]	Vector quantization	12.30	29.95
256 × 256	[22]	Sketch based	5.30	N.M
256 × 256	[24]	Fractal	11.85	30.58
512 × 512	[24]	Fractal	36.78	30.71
512 × 512	[25]	Fractal	40.00	30.20
512 × 512	[22]	Segmentation using	31.00	N.M
		Morphology	20.00	N.M
512 × 512	[26]	Block prediction	30.76	32.78
512 × 512	[27]	Region based fractal	41.00	26.56
512 × 512	[28]	Fractal	44.00	29.10
512 × 512	[29]	Fractal	44.44	29.10
256 × 256	[30]	Fractal	10.60	30.72
512 × 512	[31]	Wavelet based fractal	65.60	29.90

sion. The segmentation scheme provides a number of similar gray regions corresponding to each threshold, instead of a single region. Consequently, a global surface fit (high possibility due to similar gray regions) becomes most economical. When the order of a polynomial for approximating a sub-image goes beyond a preassigned positive integer, say q, (may happen due to the physical configuration of regions or large variation on region boundaries) we compute local corrections over the residual surfaces for which the mean squared error with respect to the global surface of order q exceeds a certain limit. Computing the order of the polynomial by the IQI based approach seems to be simple but effective. A remarkable gain in compression ratio is found when encoded in terms of surface points with the quality of reconstructed images almost the same as that found for reconstruction from control points. It is seen that texture regions require the largest number of bits during their encoding (Lena and Girl images). Examination of the quality of reconstructed images through the fidelity vector is an attempt to determine quantitatively the fidelity of images.

References

[1] M. Kunt, M. Benard, R. Leonardi, Recent results in high compression image coding, IEEE Trans. Circuits Systems 34 (1987) 1306–1336.

[2] S. Carlsson, Sketch based coding of gray level images, Signal Process. 15 (1988) 57–83.

[3] L. Shen, R.M. Rangayyan, A segmentation based lossless image coding method for high resolution medical image compression, IEEE Trans. Med. Imaging 16 (1997) 301–307.

[4] S. Biswas, N.R. Pal, On hierarchical segmentation for image compression, Pattern Recognition Lett. 21 (2000) 131–144.

[5] S. Biswas, Contour coding through discrete circular arcs by affine transformation, Pattern Recognition 34 (2001) 63–77.

[6] S. Biswas, S.K. Pal, Approximate coding of digital contours, IEEE Trans. Systems, Man, Cybernet. 18 (1988) 1056–1066.

[7] M. Eden, M. Kocher, On the performance of a contour coding algorithm in the context of image coding part I: contour segment coding, Signal Process. 8 (1985) 381–386.

[8] M. Kocher, R. Leonardi, Adaptive region growing technique using polynomial functions for image approximation, Signal Process. 11 (1986) 47–60.

[9] G.M. Schuster, G. Melinkov, A.K. Katsaggelos, Operationally optimal vertex-based shape coding, IEEE Signal Process. Mag. 15 (1998) 91–108.

[10] M. Eden, M. Unser, R. Leonardi, Polynomial representation of pictures, Signal Process. 10 (1986) 385–393.

[11] T. Pavlidis, Algorithms for Graphics and Image Processing, Springer-Verlag, New York, 1982.

[12] S. Biswas, N.R. Pal, S.K. Pal, Smoothing of digital images using the concept of diffusion process, Pattern Recognition 29 (1996) 497–510.

[13] Nathaniel Macon, Numerical Analysis, Wiley, New York, 1963.

[14] E.H. Hall, Computer Image Processing and Recognition, Academic Press, New York, 1979.

[15] H.O. Peitjen, H. Jurjens, D. Saupe, Chaos and Fractals, Springer-Verlag, New York, 1992.

[16] S. Kamata, N. Niimi, E. Kawaguchi, A gray image compression using Hilbert scan, Proceedings of the ICPR, Vienna, Austria 1996, pp. 905–909.

[17] S. Biswas, S.K. Pal, D. Dutta Majumder, Binary contour coding using Bezier approximation, Pattern Recognition Lett. 8 (1988) 237–249.

[18] C.E. Shannon, W. Weaver, The Mathematical Theory of Communication, University of Illinois Press, Urbana, 1949.

[19] B.B. Mandelbrot, Fractal Geometry of Nature, Freeman Press, SanFrancisco, 1982.

[20] B.B. Choudhuri, N. Sarkar, Texture segmentation using fractal dimension, IEEE Trans. Pattern Anal. Machine Intell. 17 (1995) 72–77.

[21] G.K. Wallace, The JPEG still picture compression standard, Commun. ACM 34 (4) (1991) 30–44.
[22] P. Salembier, Morphological multiscale segmentation for image coding, Signal Process. 38 (1994) 359–386.
[23] B. Ramamurthi, A. Gersho, Classified vector quantization of images, IEEE Trans. Commun. 34 (1986) 1105–1115.
[24] Y. Fisher, E.W. Jacbos, R.D. Boss, Fractal image compression using iterated transforms, in: J.A. Storer (Ed.), Image and Text Compression, Kluwer Academic Publishers, Dordrecht, 1992, pp. 35–61.
[25] H. Lin, A.N. Venetsanopoulos, Incorporating nonlinear contractive functions into the fractal coding, Proceedings of the IEEE International Workshop on Intelligent Signal Processing and Communication Systems, Seoul, Korea, 1994, pp. 169–172.
[26] R. Rinaldo, G. Calvango, Image coding by block prediction of multi-resolution subimages, IEEE Trans. Image Process. 4 (1995) 909–920.
[27] L. Thomas, F. Deravi, Region-based fractal image compression using heuristic search, IEEE Trans Image Process. 4 (1995) 832–838.
[28] H. Lin, A.N. Venetsanopoulos, A pyramid algorithm for fast fractal image compression, Proceedings of the IEEE International Conference on Image Processing (ICIP'95), Washington, DC, USA, 1995, pp. 596–599.
[29] N.R. Thao, A hybrid fractal-DCT coding scheme for image compression, Proceedings of the IEEE International Conference on Image Processing (ICIP'96), Lausanne, Switzerland, 1996, pp. 169–172.
[30] M.E. Haziti, H. Cherifi, D. Aboutajdine, Complexity reduction in fractal image compression, Proceedings of the IASTED International Conference on Signal and Image Processing (SIP'97), New Orleans, USA, 1997, pp. 245–250.
[31] G. Davis, A wavelet-based analysis of fractals image compression, IEEE Trans. Image Process. 7 (1998) 141–154.

About the Author—SAMBHUNATH BISWAS obtained the M.Sc. degree in Physics from the University of Calcutta in 1973 and Ph.D. (Tech.) in Radio-Physics and Electronics in 2001. He was in electrical industries in the beginning as a Graduate Engineering Trainee and then as a Design and Development Engineer. He was a UNDP fellow at MIT, USA to study Machine Vision during 1988–89. He visited the Australian National University at Canberra in 1995 and Zhejiang University at Hangzhou in China in 2001. He is now a System Analyst at the Machine Intelligence Unit in Indian Statistical Institute, Calcutta. His research interests include image processing, machine vision, computer graphic, and pattern recognition.