Overlapped Rectangle Image Representation and Its Application to Exact Legendre Moments Computation

HUANG Wei  CHEN Chuanbo  SAREM Mudar  ZHENG Yunping

Abstract  Linear quadtree is a popular image representation method due to its convenient imaging procedure. However, the excessive emphasis on the symmetry of segmentation, i.e. dividing repeatedly a square into four equal sub-squares, makes linear quadtree not an optimal representation. In this paper, a no-loss image representation, referred to as Overlapped Rectangle Image Representation (ORIR), is presented to support fast image operations such as Legendre moments computation. The ORIR doesn’t importune the symmetry of segmentation, and it is capable of representing, by using an identical rectangle, the information of the pixels which are not even adjacent to each other in the sense of 4-neighbor and 8-neighbor. Hence, compared with the linear quadtree, the ORIR significantly reduces the number of rectangles required to represent an image. Based on the ORIR, an algorithm for exact Legendre moments computation is presented. The theoretical analysis and the experimental results show that the ORIR-based algorithm for exact Legendre moments computation is faster than the conventional exact algorithms.

Keywords  image processing; image representation; Legendre moments computation

Introduction

Image representation is a principal research area in image processing, pattern recognition, and robotics. In general, there are three types of image representation: the first is developed to support special devices; the second is for the compression of images; and the third is developed to support special image operations. Linear quadtree is a representative of the third type of image representation[1]. The linear quadtree uses 2-dimension correlation which universally exists in any image in the real world, therefore, it reduces the amount of data required to represent an image. However, the nature of insisting the symmetry on segmentation, i.e., repeatedly dividing a square into four equal sub-squares, still impairs the compactness.

Orthogonal moments, such as Legendre moments and Zernike moments, were first introduced in image processing by Teague[2] and they have been widely used in image analysis and pattern recognition[3,4] due to their near-zero information redundancy and high discriminative power. Although Legendre moments have been successfully applied to many applications such as face recognition[5] and line fitting[6], their use was limited due to the computational complexity. Many researches have been conducted to reduce the computational complexity of Legendre moments[7-11].

Received on July 12, 2008.
Supported by the National High Technology Research and Development Program of China (No. 2006AA042211).
HUANG Wei, School of Computer Science and Technology, Huazhong University of Science and Technology, 1037 Luoyu Road, Wuhan 430074, China.
E-mail: huangw2046@gmail.com
However, while some of them focused on binary images\cite{7,8}, some of them proposed the inexact algorithms for computing Legendre moments\cite{7,9}. Liao and Pawlak performed an error analysis on the set of Legendre moments and proposed to use the Alternative Extended Simpson’s Rule (AESR) to minimize the error\cite{10}. However, the computed moments are still inexact. Yap and Paramesran further analyzed the reason of inaccuracy presented in Legendre moments computation and proposed a method to compute the exact values of the Legendre moments by mathematically integrating the Legendre polynomials over the corresponding intervals of the image pixels\cite{11}. However, their method kept the double summation involved in moments computation untouched, and hence the total computational cost of Legendre moments is still enormous.

In this paper, a no-loss image representation, referred to as Overlapped Rectangle Image Representation (ORIR), is presented to support fast image operations. The ORIR is capable of representing, by using an identical rectangle, the information of the pixels which are not even adjacent to each other in the sense of 4-neighbor and 8-neighbor. So, compared with the linear quadtree, the ORIR has significantly reduced the number of rectangles required to represent an image. Based on the ORIR, a fast algorithm for computing exact Legendre moments of both binary and gray level images is presented. The theoretical analysis and the experimental results in this paper show that the ORIR-based algorithm for Legendre moments computation is as exact as, but faster than, the exact algorithm proposed by Yap and Paramesran\cite{11}.

1 Overlapped rectangle image representation

This section presents the Overlapped Rectangle Image Representation (ORIR) and its encoding algorithm.

Fig.1(a) shows a $4 \times 4$ image where all the pixels have the same color value $a$ except that the color value of the pixels $(1, 1), (2, 1),$ and $(1, 3)$ are $b, c,$ and $c$, respectively. Without loss of generality, we suppose $0 < a < b < c$. Quadtree segmentation illustrated in Fig.1(b) divides repeatedly a square into four equal sub-squares until all pixels in a square have the same color. The corresponding quadtree representation is given in Fig.1(d). Obviously, the linear quadtree requires 13 nodes to represent the original image.

In this paper, a no-loss image representation, referred to as Overlapped Rectangle Image Representation (ORIR), is presented to support fast image operations. The ORIR is capable of representing, by using an identical rectangle, the information of the pixels which are not even adjacent to each other in the sense of 4-neighbor and 8-neighbor. So, compared with the linear quadtree, the ORIR has significantly reduced the number of rectangles required to represent an image. Based on the ORIR, a fast algorithm for computing exact Legendre moments of both binary and gray level images is presented. The theoretical analysis and the experimental results in this paper show that the ORIR-based algorithm for Legendre moments computation is as exact as, but faster than, the exact algorithm proposed by Yap and Paramesran\cite{11}.

![Fig.1 Quadtree representation and overlapped rectangle image representation (ORIR) for a 4×4 image](image)
every rectangle as the basic color value of this rectangle.

The ORIR segmentation of the original image given in Fig.1(a) is illustrated in Fig.1(c). We consider this 4 × 4 image as a rectangle \( r_0 \), whose basic color value is 0 and whose width and height are both 4. Because the color values of some pixels covered with the rectangle \( r_0 \) are not 0, some additional rectangles must be used to identify the colors and the positions of these pixels. In raster scan order, the first pixel, which is covered with \( r_0 \) and whose color is not 0, is the pixel (0, 0). So the start point, i.e. the topmost-left pixel, of the rectangle \( r_1 \) is the pixel (0, 0). According to our simple strategy for selecting the basic color value, the basic color value of \( r_0 \) is \( a \). Recall that \( 0 < a < b < c \), and it is obvious that both the width and the height of the rectangle \( r_1 \) are 4, the same as the width and the height of the rectangle \( r_0 \) by chance. Similarly, because there are some pixels which are covered with the rectangle \( r_1 \) and whose colors are not the same as the basic color value \( a \) of \( r_1 \), two additional rectangles, i.e. \( r_2 \) and \( r_4 \), are used to identify the colors and the positions of these pixels. The start point, the basic color value, the width, and the height of the rectangle \( r_2 \) are the pixel (1, 1), \( b \), 2, and 1, respectively. The start point, the basic color value, the width, and the height of the rectangle \( r_4 \) are the pixel (1, 3), \( c \), 1, and 1, respectively. Finally, because the color value of the pixel (2, 1), which is directly covered with the rectangle \( r_5 \), is \( c \) not \( b \), the fifth rectangle \( r_5 \) is used to identify the color and the position of this pixel. The start point, the basic color value, the width and the height of the rectangle \( r_3 \) are the pixel (2, 1), \( c \), 1, and 1, respectively. All the rectangles, which are obtained by the segmentation mentioned above, can be organized in a tree without fixed outdegree. However, for the merits of binary trees, we organize these rectangles in a child-sibling tree, as illustrated in Fig. 1(c). In Fig. 1(e), an ORIR rectangle is represented by using a 5-tuple, i.e. \((i, j, w, h, c)\) where \(i, j, w, h\), and \(c\) denote the \(x\)-coordinate of the start point, the \(y\)-coordinate of the start point, the width, the height, and the difference between the basic color value of the current rectangle and the basic color value of the rectangle that directly contains the current rectangle, respectively. Moreover, we eliminate the pointers in the ORIR and store the child-sibling tree as a list that corresponds to the preorder traversal of this child-sibling tree, as illustrated in Fig.1(f). In the remainder of this paper, we will refer to this list as ORIR list.

From Fig.1(c), it can be seen that the information of some pixels that are detached, such as the pixel (0, 0) and the pixel (2, 2), is represented by using the identical rectangle \( r_1 \) in the ORIR. Undoubtedly, this characteristic makes the ORIR a highly efficient image representation.

The recursive algorithm, which converts an image into its ORIR list, is given below.

**Input** An image \( G \) and an ORIR rectangle \( r = (i, j, w, h, c) \) where \(i, j, w, h, \) and \(c\) denote the \(x\)-coordinate of the start point, the \(y\)-coordinate of the start point, the width, the height, and the difference between the basic color value of the current rectangle \( r \) and the basic color value of the rectangle that directly contains the current rectangle \( r \), respectively.

**Output** The ORIR list \( Q \) of the \( w \times h \) sub-image \( SG \) that is covered with \( r \).

**Step 1** Initialize \( Q \) as an empty list.

**Step 2** Let \( SG \) a \( w \times h \) sub-image of the image \( G \) where all pixels in \( SG \) are covered with rectangle \( r \).

**Step 3** Scan \( SG \) in raster scan order to find the first pixel \( sp \) which is not covered with any rectangle in \( Q \) and whose color value is more than \( c \). If no \( sp \) is found, go to step 8.

**Step 4** Let \( sp_i, sp_j, sp_w, \) and \( sp_h \) indicate the \(x\)-coordinate, the \(y\)-coordinate, and the color value of the pixel \( sp \) respectively. Let \( sp \) be the topmost-left pixel. Match all rectangles where the color values of all the pixels covered with these rectangles are more than or equal to \( sp_c \) and no pixel in these rectangles are covered with any rectangles in \( Q \).

**Step 5** Among the rectangles obtained by step 4, select one rectangle \( r' \) whose area is the biggest. Let \( w' \) and \( h' \) be the width and the height of \( r' \), respectively. Append \((sp_i, sp_j, w', h', sp_c - c)\) to the rear of \( Q \).

**Step 6** Call recursively ORIR generation algorithm with the parameters \( G \) and \((sp_i, sp_j, w', h', sp_c)\) to get the return ORIR list \( Q' \). Append orderly all elements in \( Q' \) to the rear of \( Q \).

**Step 7** Go back to step 3.
Step 8 Output $Q$.

To convert a $M \times N$ image $G$ into its ORIR list $Q$, we first construct an empty list $Q$ and append the ORIR rectangle $r = (0, 0, M, N, 0)$ to $Q$. Then, we call ORIR generation algorithm with the input $G$ and $r$ to get the returned list $Q'$. Last, we orderly append all the elements in $Q'$ to the rear of $Q$.

2 Exact Legendre moments computation

In this section, we apply the ORIR to the exact Legendre moments computation.

2.1 Legendre moments

The $p + q$ order Legendre moment $\lambda_{pq}$ of a density function $f(x, y)$ is defined as:

$$\lambda_{pq} = \frac{(2p+1)(2q+1)}{4} \int_{-1}^{1} \int_{-1}^{1} P_p(x)P_q(y)f(x,y)dx\,dy$$  \hspace{1cm} (1)

where $p, q = 0, 1, \cdots, \infty$ and the $n$th-order Legendre polynomial $P_n(x)$ is defined as:

$$P_n(x) = \frac{1}{n!} \frac{d^n}{dx^n} (1-x^2)^{\frac{n}{2}}$$  \hspace{1cm} (2)

where $x \in [-1, 1]$, and $[n/2]$ denotes the biggest integer that is less than or equal to $n/2$. The recursive relation of Legendre polynomials is:

$$P_n(x) = \frac{x(2n-1)P_{n-1}(x) - (n-1)P_{n-2}(x)}{n}, \quad n \geq 2$$  \hspace{1cm} (3)

with $P_0(x) = 1$ and $P_1(x) = x$. The integral property of Legendre polynomials is:

$$\int P_n(x)dx = \frac{P_{n+1}(x) - P_{n-1}(x)}{2n+1}$$  \hspace{1cm} (4)

For discrete image $f(i, j)$ of size $M \times N$ where $i = 1, 2, \cdots, M$ and $j = 1, 2, \cdots, N$, an widely used approximate version $^{7-9}$ is defined as:

$$\lambda_{pq} = \frac{(2p+1)(2q+1)}{(M-1)(N-1)} \sum_{i=1}^{M} \sum_{j=1}^{N} P_p(x)P_q(y)f(i,j)$$  \hspace{1cm} (5)

where $(x, y) \in [-1, 1] \times [-1, 1]$ is the image of the coordinates $(i, j)$ of a pixel under a certain mapping. One of the widely used mappings is:

$$\begin{align*}
x_i &= \frac{2i-M-1}{M-1} \\
y_j &= \frac{2j-N-1}{N-1}
\end{align*}$$  \hspace{1cm} (6)

The Legendre moments computed by using Eq.(5) are inexact. A more exact version of Eq.(5) was given by Yap and Paramesran $^{11}$ as follows:

$$\tilde{\lambda}_{pq} = \sum_{i=1}^{M} \sum_{j=1}^{N} f(i, j)Q_p(x_i)Q_q(y_j)$$  \hspace{1cm} (7)

where

$$Q_p(x_i) = \frac{(2p+1)}{2} \int_{-\frac{\Delta x}{2}}^{\frac{\Delta x}{2}} P_p(x)dx$$  \hspace{1cm} (8)

and

$$Q_q(y_j) = \frac{(2q+1)}{2} \int_{-\frac{\Delta y}{2}}^{\frac{\Delta y}{2}} P_q(y)dy$$  \hspace{1cm} (9)

where $\Delta x = x_{i+1} - x_i$ and $\Delta y = y_{j+1} - y_j$. The mapping used by them to transfer $(i, j)$ to $(x_i, y_j)$ is:

$$\begin{align*}
x_i &= -1 + (i - \frac{1}{2}) \Delta x \\
y_j &= -1 + (j - \frac{1}{2}) \Delta y
\end{align*}$$  \hspace{1cm} (10)

where the intervals $\Delta x = \Delta y = 2/M$, and are fixed at a constant value for $M \times N$ images.

Yap and Paramesran $^{11}$ noted that $Q_p(x_i)$ and $Q_q(y_j)$ are independent of images. Hence, they computed, stored them in advance, and retrieved whenever necessary. However, the double summation presented in Eq.(5), which is time-consuming, still restrains their algorithm to be optimal.

In the next subsection, we will replace the double summation in Eq.(5) with a single summation over all the rectangles in the ORIR list to speed up the exact Legendre moments computation.

2.2 ORIR-Based algorithm for exact Legendre moments computation

Before describing the ORIR-based algorithm for exact Legendre moments computation, we first give some relevant definitions.

**Definition 1 Companion Images.** A companion image $B_i$ of the original image $G$ is a binary image where the $B_i$ corresponds to the $i$th rectangle $r_i$ in the ORIR list of $G$. In $B_i$, the color values of all the pixels covered with $r_i$ are 1 and the others are 0.

**Definition 2 Weight of Companion Images.** Suppose $B_i$ is a companion image corresponding to the $i$th rectangle $r_i = (i_x, j_x, w_x, h_x, c_i)$ of the original image $G$. Then, we assign $c_i$, i.e. the difference between the basic color value of $r_i$ and the basic color...
value of the rectangle which directly contains \( r_s \), to \( B_s \) as the weight of \( B_s \).

Now, we can represent an \( M \times N \) image \( G \) by the weighted summation of the density functions of its companion images as follows:

\[
G = f(i,j) = \sum_{i=1}^{K} c_i \times b_i(i,j), i = 1, 2, \ldots, M;
\]

\[ j = 1, 2, \ldots, N \]

(11)

where \( K \) is the number of the companion images of \( G \), \( b_i(i,j) \) is the density function of the \( i \)th companion image \( B_i \), and \( c_i \) is the weight of \( B_i \).

Using Eq.(10), we can map the companion image \( B_i \) into the region \([-1, 1] \times [-1, 1] \). Accordingly, we denote the mapped density function over the region \([-1, 1] \times [-1, 1] \) as \( b'_i(x,y) \) where \( b'_i(x,y) = b_i(i,j) \) for \(( x,y) \in [-1 + (i-1)\Delta x, -1 + i\Delta x] \times [-1 + (j-1)\Delta y, -1 + j\Delta y] \). Then, the mapped density function \( f(x,y) \) of the original density function \( f(x,y) \), whose domain is the region \([-1, 1] \times [-1, 1] \), is:

\[
f(x,y) = \sum_{i=1}^{K} c_i \times b'_i(x,y),
\]

\[(x,y) \in [-1, 1] \times [-1, 1] \]

(12)

Substituting Eq.(12) into Eq.(1) and exchanging the order of the integrals and the summation, we get:

\[
\lambda_{pq}^{(s)} = \frac{\sum_{i=1}^{K} c_i \times \lambda_{pq}^{(s)}}{4}
\]

(13)

where

\[
\lambda_{pq}^{(s)} = (2p+1)(2q+1) \int_{\mathbb{R}^2} P_p(x)P_q(y)dxdy,
\]

\[ \forall(x,y): b'_i(x,y) = 1 \]

(14)

is the \( p+q \) order exact Legendre moment of the \( s \)th companion image \( B_s \). (In fact, \( \lambda_{pq}^{(s)} \) is the fourfold \( p+q \) order exact Legendre moment of the \( s \)th companion image \( B_s \)).

It is practically feasible to compute \( \lambda_{pq}^{(s)} \) by using any algorithm for computing exact Legendre moments of binary images. However, note that there is only a rectangular region in \( B_s \). Then, from Eq. (14), we get:

\[
\lambda_{pq}^{(s)} = ((2p+1)\int_{\mathbb{R}^2} P_p(x)dx)((2q+1)\int_{\mathbb{R}^2} P_q(y)dy)
\]

(15)

where

\[
\begin{align*}
x_s^{(i)} &= -1 + (i_s - 1) \times \Delta x \\
y_s^{(i)} &= -1 + (j_s - 1) \times \Delta y
\end{align*}
\]

(16)

and

\[
\begin{align*}
x_s^{(w)} &= -1 + (i_s + w_s - 1) \times \Delta x \\
y_s^{(w)} &= -1 + (j_s + h_s - 1) \times \Delta y
\end{align*}
\]

(17)

where \( i_s, j_s, w_s, \) and \( h_s \) are, respectively, the \( x \)-coordinate of the start point, the \( y \)-coordinate of the start point, the width, and the height of the ORIR rectangle \( r_s \).

For \( p > 0 \) and \( q > 0 \), from Eq.(4) and Eq.(15), we get:

\[
\lambda_{pq}^{(s)} = \frac{4w_s \times h_s}{M \times N}
\]

(19)

For \( p = 0 \) and \( q > 0 \), from Eq.(15), we get 0+q order Legendre moment of \( B_s \) as follows:

\[
\lambda_{p, q+1}^{(s)} = \frac{2x \times [P_{p+1}(y) - P_{p+1}(y)]^{(s)}_x}{M}
\]

(20)

For \( p > 0 \) and \( q = 0 \), from Eq.(15), we get \( p+0 \) order Legendre moment of \( B_s \) as follows:

\[
\lambda_{p+1, q}^{(s)} = \frac{2x \times [P_{p+1}(x) - P_{p+1}(x)]^{(s)}_y}{N}
\]

(21)

Conclusively, in order to compute the exact Legendre moments up to the order \( p+q \), the ORIR-based algorithm for exact Legendre moments computation comprises three steps. The first is to compute the Legendre moments up to \( p+1 \) order for every \( x = -1, -1+2/M, \ldots, 1 \) and the Legendre polynomials up to \( q+1 \) order for every \( y = -1, -1+2/N, \ldots, 1 \) by using Eq.(3). This step is independent of images and can always be computed in advance, stored, and retrieved whenever necessary. The second is to compute the exact Legendre moments up to order \( p+q \) for every companion image \( B_1, B_2, \ldots, B_K \) by using Eq.(18), Eq.(19), Eq.(20) or Eq.(21). The last step is to compute the weighted summation of the \( p+q \) order exact Legendre moments of all the companion images to obtain the \( p+q \) order exact Legendre moments of the original image \( G \) by using Eq.(13).
2.3 Complexity analysis of the ORIR-based algorithm for exact Legendre moments computation

In this subsection, we analyze the computational complexity of the ORIR-based algorithm for exact Legendre moments computation. In general, there are two stages in exact Legendre moments computation. The first is to generate the moments kernels, i.e. \( \rho_{Qx}(x) \) and \( \rho_{Qy}(y) \) in Yap and Paramesran’s algorithm or \( \rho_{P\text{ }x}(x) \) and \( \rho_{P\text{ }y}(y) \) in the ORIR-based algorithm. The second is to multiply the moments kernels with the image density function and sum the products up to obtain the exact Legendre moments of the original image, i.e. double summations by using Eq.(7) in Yap and Paramesran’s algorithm or the second step and the last step in the ORIR-based algorithm presented in the previous subsection.

Because the moments kernels are independent of images and can hence be computed in advance, the first stage is relatively unimportant for total computational complexity. So we would rather only analyze the computational complexity of the second stage.

Suppose that \( K \) rectangles are required in the ORIR to represent an \( M \times N \) digitized image \( G \). In order to generate up to \( p+q \) order exact Legendre moments of \( G \), computing \( 0+0 \) order exact Legendre moment of a companion image \( B_s \) requires only 4 multiplications by using Eq.(19); computing \( 0+q(q \geq 0) \) order exact Legendre moments of \( B_s \) by using Eq.(20) requires 3 multiplications and 3 additions; computing \( p+0 \) \((p > 0)\) order exact Legendre moments of \( B_s \) by using Eq.(21) requires 3 multiplications and 3 additions; and computing \( p+q(p > 0 \text{ and } q > 0) \) exact order Legendre moment of \( B_s \) using Eq.(18) requires 1 multiplication and 6 additions. So, the total multiplications and the total additions required to compute up to \( p+q \) order exact Legendre moments of all \( K \) companion images are \( K(pq+3p+3q+4) \) and \( 3K(2pq+p+q) \), respectively. In order to compute the weighted summations of these exact Legendre moments of the companion images using Eq.(13), additional \( (K+1)(p+1)(q+1) \) multiplications and \( (K-1)(p+1)(q+1) \) additions are required. So, in order to obtain up to \( p+q \) order exact Legendre moments of the original image \( G \), the total multiplications and the total additions involved in the second stage, are \( K(2pq+4p+4q+5)+(p+1)(q+1) \) and \( K(7pq+4p+4q+1)-(p+1)(q+1) \), respectively.

The comparisons of the operation numbers between the ORIR-based algorithm and Yap and Paramesran’s algorithm\[11\] are given in Table 1.

Because the multiplication is more time-consuming than the addition, we define the acceleration ratio \( \eta \) in terms of the number of multiplications as follows:

\[
\eta = \frac{2MN(p+1)(q+1)}{K(2pq+4p+4q+5)+(p+1)(q+1)}
\]

In general, \( p \) and \( q \) are much less than \( K \) and \( MN \). So, we can get an approximate value \( \hat{\eta} \) of \( \eta \) as follows:

\[
\hat{\eta} = \frac{MN}{K}
\]

Because \( K \) is less than \( MN \) for all images, from Eq.(23), it is clear that the ORIR-based algorithm for exact Legendre moments computation is faster than that proposed by Yap and Paramesran\[11\].

3 Experimental results

The experiments are performed on a personal computer with 1.8G Hz CPU and 512M Bytes main memory. The relevant algorithms are coded using the C++ programming language. Two binary images and two gray level images used in the experiments are shown in Fig.2.

First, we verify the compactness of the ORIR. From Table 2, it can be seen that the number \( N_R \) of the rectangles required by the ORIR to represent an image is much less than the number \( N_{\text{LQT}} \) of the nodes required by the linear quadtree. The ratios of
$N_{LQT}$ to $N_R$ range from 1.31 to 4.45, which show that the ORIR is more compact than the linear quadtree.

![Fig. 2 Four 2^9×2^9 images used for experiments](Image)

Table 2  Comparisons between the number $N_R$ of the rectangles required by the ORIR and the number $N_{LQT}$ of the nodes required by the linear quadtree

| Image     | $N_{LQT}$ | $N_R$ | $N_{LQT} / N_R$ |
|-----------|-----------|-------|-----------------|
| Man       | 3 834     | 861   | 4.45            |
| Sailboat  | 17 988    | 4 711 | 3.82            |
| Lena      | 253 816   | 153 184 | 1.67         |
| Peppers   | 260 980   | 199 413 | 1.31         |

Second, in order to verify that the ORIR-based algorithm gives exact Legendre moments values as claimed, up to $5 + 5$ order Legendre moments values of the gray level image shown in Fig.2(d) are given in Table 3 and Table 4. The Legendre moments values given in Table 3 are computed by the ORIR-based algorithm and those given in Table 4 are computed by the algorithm proposed by Yap and Paramesran. From Table 3 and Table 4, it is easy to say that the ORIR-based algorithm for exact Legendre moments computation is as exact as that proposed by Yap and Paramesran.

Table 3  Up to $5 + 5$ Order Legendre moments values computed by the ORIR-based algorithm for peppers image illustrated in Fig.2(d)

| p = 0 | q = 0 | 1   | 2   | 3   | 4   | 5   |
|-------|-------|-----|-----|-----|-----|-----|
|       |       | 120.044 | 11.214 | 4.406 | 86 | -10.254 | 50 | 23.880 | 60 | -4.052 | 49 |
| 1     | 1.117 | 64 | 32.585 | 80 | -24.778 | 30 | 10.240 | 50 | 2.793 | 51 | -5.485 | 75 |
| 2     | -1.319 | 23 | 8.457 | 33 | 55.200 | 80 | -56.522 | 70 | -44.544 | 00 | 34.339 | 60 |
| 3     | 11.050 | 90 | -18.505 | 30 | -23.303 | 40 | -39.608 | 70 | 60.378 | 80 | 13.978 | 30 |
| 4     | -9.361 | 63 | 21.550 | 60 | -14.710 | 40 | -4.131 | 37 | 12.147 | 40 | -19.001 | 30 |
| 5     | -1.108 | 38 | -13.115 | 10 | 18.500 | 20 | 65.781 | 90 | -28.518 | 10 | -65.441 | 60 |

Table 4  Up to $5 + 5$ Order Legendre moments values computed by yap and Paramesran’s algorithm for peppers image illustrated in Fig.2(d)

| p = 0 | q = 0 | 1   | 2   | 3   | 4   | 5   |
|-------|-------|-----|-----|-----|-----|-----|
|       |       | 120.044 | 11.214 | 4.406 | 86 | -10.254 | 50 | 23.880 | 60 | -4.052 | 49 |
| 1     | 1.117 | 64 | 32.585 | 80 | -24.778 | 30 | 10.240 | 50 | 2.793 | 51 | -5.485 | 75 |
| 2     | -1.319 | 23 | 8.457 | 33 | 55.200 | 80 | -56.522 | 70 | -44.544 | 00 | 34.339 | 60 |
| 3     | 11.050 | 90 | -18.505 | 30 | -23.303 | 40 | -39.608 | 70 | 60.378 | 80 | 13.978 | 30 |
| 4     | -9.361 | 63 | 21.550 | 60 | -14.710 | 40 | -4.131 | 37 | 12.147 | 40 | -19.001 | 30 |
| 5     | -1.108 | 38 | -13.115 | 10 | 18.500 | 20 | 65.781 | 90 | -28.518 | 10 | -65.441 | 60 |

Finally, we give the comparisons of the CPU elapsed time for computing the Legendre moments kernels in Table 5 and for computing the Legendre moments in Table 6. From Table 5, it is clear that, although the ORIR-based algorithm must generate up to 101 order Legendre polynomials for computing up to $100 + 100$ order exact Legendre moments, the total CPU elapsed time for computing these moments kernels by the ORIR-based algorithm is much less than that required by the algorithm proposed by Yap and Paramesran due to the fact that the computations of $P_n(x)$ and $P_n(y)$ are simpler than the computations of $Q_n(x)$ and $Q_n(y)$. Table 6 gives the CPU elapsed time for computing up to $100 + 100$ order exact Legendre moments by both the ORIR-based algorithm and the algorithm proposed by Yap and Paramesran. From Table 6 and Table 2, we can see that the CPU elapsed time $T'_R$ of the ORIR-based
algorithm increases when the number $N_R$ of the rectangles required to represent an image increases. However, the CPU elapsed time $T'R$ of the ORIR-based algorithm is much less than that of the algorithm proposed by Yap and Paramesran, $T'Y$. The ratios of $T'Y$ to $T'R$ range from 1.93 to 462.04. These results show that the ORIR-based algorithm for exact Legendre moments computation is faster than that proposed by Yap and Paramesran.

Table 5  Comparisons between the CPU elapsed time $T'R$ of the ORIR-based algorithm and the CPU elapsed time $T'Y$ of the Yap and Paramesran’s algorithm for Legendre moments Kernels generation

| Image | $T'Y$/ms | $T'R$/ms | $T'Y/T'R$ |
|-------|----------|----------|------------|
| Man   | 30.01    | 2.92     | 10.28      |
| Sailboat | 30.62    | 2.98     | 10.28      |
| Lena  | 31.77    | 3.09     | 10.28      |
| Peppers | 30.98    | 3.02     | 10.26      |

Table 6  Comparisons between the CPU elapsed time $T'R$ of the ORIR-based algorithm and the CPU elapsed time $T'Y$ of Yap and Paramesran’s algorithm for computing up to $100 + 100$ order Legendre moments

| Image | $T'Y$/ms | $T'R$/ms | $T'Y/T'R$ |
|-------|----------|----------|------------|
| Man   | 41902.8  | 90.7     | 462.04     |
| Sailboat | 41684.8  | 504.2    | 82.68      |
| Lena  | 41657.5  | 16596.9  | 2.51       |
| Peppers | 42027.9  | 21760.7  | 1.93       |

4  Conclusion

In this paper, we present a no-loss image representation, referred to as Overlapped Rectangle Image Representation (ORIR). The ORIR is capable of coding the information of some detached pixels by using a single rectangle and therefore significantly reduces the number of rectangles required to represent images compared with the linear quadtree.

Based on the ORIR, a fast algorithm for exact Legendre moments computation is presented as well. Due to the high compactness of the ORIR, the presented algorithm reduces the CPU elapsed time markedly. The experimental results given in this paper show that the ORIR-based algorithm for exact Legendre moments computation is as exact as, but faster than that proposed by Yap and Paramesran.

References

[1] Gargantini I (1982) An effective way to represent quad-trees [J]. Communications of the ACM, 25(12): 905-910.
[2] Teague M R (1980) Image analysis via the general theory of moments [J]. Journal of Optical Society of America, 70(8): 920-930.
[3] Liu Maofu, He Yanxiang, Ye Bin (2007) Image zernike moments shape feature evaluation based on image reconstruction [J]. Geo-spatial Information Science, 10(3): 191-195.
[4] Feng Yucai, Wu X, Liang Junjie (2007) Improving retrieval performance of zernike moment descriptor via weighted partitions and relevance feedback [J]. Journal of Image and Graphics, 12(6): 1086-1090.
[5] Haddadnia J, Faez K, Moallem P (2001) Neural network based face recognition with moment invariants [C]. 2001 International Conference on Image Processing, Thessaloniki, Greece.
[6] Qiidaa H, Radouane L (1999) Robust line fitting in a noisy image by the Method of Moments [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 21(11): 1216-1223.
[7] Shu Huazhong, Luo Limin, Yu Wenxue, et al. (2000) A new fast method for computing Legendre moments [J]. Pattern Recognition, 33(2): 341-348.
[8] Zhou Jindan, Shu Huazhong, Luo Limin, et al. (2002) Two new algorithms for efficient computation of Legendre moments [J]. Pattern Recognition, 35(5): 1143-1152.
[9] Yang Guanyu, Shu Huazhong, Tournoulin C, et al. (2006) Efficient Legendre moment computation for grey level images [J]. Pattern Recognition, 39(1): 74-80.
[10] Liao Xinmeng, Pawlak M (1996) On image analysis by moments [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 18(3): 254-266.
[11] Yap P T, Paramesran R (2005) An efficient method for the computation of Legendre moments [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(12): 1996-2002.