Real-Time Imaging **7**, 519–527 (2001)
doi:10.1006/rtim.2001.0265, available online at http://www.idealibrary.com on IDEAL®

Fast Algorithm for Local Statistics Calculation for N-Dimensional Images

Local mean and variance measures are frequently required in multi-dimensional image analysis. These measures are needed when calculating correlation coefficients for local image matching purposes. Other measures such as skewness and autocorrelation are useful for texture analysis. This paper presents a fast algorithm for calculating these local statistics in a window of an *N*-dimensional image. The new algorithm, which is called the plunger method, recursively reduces the dimensions of the input *N*-dimensional image to achieve fast computation. The speed of the algorithm is independent of the window size. Another advantage of the algorithm is that it calculates the local statistics in one pass. Real image tests have been performed.

Changming Sun

CSIRO Mathematical and Information Sciences,
Locked Bag 17, North Ryde, NSW 1670, Australia
E-mail: changming.sun@cmis.csiro.au

Introduction

Local mean estimator provides one of the methods for smoothing an image and is often useful as a general-purpose smoothing algorithm when the exact form of the smoothing point-spread function is not important and when the computational speed is an issue. This application applies a square or rectangular box filter to a one- or two-dimensional image for smoothing purposes. Each output pixel is the mean of the input pixels within the filter box. Local variance calculation in an image is also important. Apart from the application of mean and variance filtering of an image, the obtained local mean and variance can be used for fast calculation of cross correlation or sum of squared differences between two images for image matching or registration purpose. Local skewness and autocorrelation measures can be used for image texture analysis [1, Ch. 9].

McDonnell [2] described several box-filtering procedures for 2D images. The main advantage of box filtering is its speed, which approaches four operations for each output pixel and is independent of box size. Sun [3–5] extends the idea of box filtering for fast calculation of normalized cross correlations for 2D images for the purpose of stereo matching and image motion estimation. Luciano da Fontoura Costa describes a numerical approach to the expedite calculation of vector fields in two-dimensional spaces and how it has allowed the effective application of Gauss' law in image analysis and computer vision [6].

3D images, especially in the medical area such as MRI, CT, PET, and ultrasound, are becoming more readily available. We might also like to treat a sequence of 2D images as a 3D image volume for spatiotemporal analysis [7]. Other types of 3D data include seismic data

1077-2014/01/060519+09 $35.00/0

volume [8], confocal microscopic images [9], spectroscopy images in medical and remote sensing applications. Imaging scanners now exist that can generate 4D images. A 4D image is a time sequence of 3D volumetric images. Much work has been done in the world of 4D imaging. Kriete *et al.* outlined 4D microscopy resources and visualization techniques in [10], while ANALYZE [11] and VIDA [12] are medical image analysis packages that can be used for visualization and display of 4D images. In addition, studies on image enhancement using 4D mathematical morphology and 4D morphological filters [13–15] have shown their utility. Higgins *et al.* [16] describe a procedure for performing semi-automatic image segmentation and analysis upon a 4D cardiac image. Niessen *et al.* [17] introduced a general framework for spatiotemporal analysis of (D + 1)-dimensional datasets. Rohr studied the extraction of 3D point landmark using 3D differential operators [18]. Most of the 4D images are currently from medical applications.

In this paper we present fast algorithms for local statistics calculation for N-dimensional images. We will extend McDonnell's 2D box filtering method of calculating the local mean to calculating local variance and local skewness during the same pass on 2D images. We will also extend the 2D method to general N-dimensional images for fast calculation of local statistics. We call the new algorithm the "plunger technique".

The rest of the paper is organized as follows: The next section reviews the 2D box filtering techniques and extends it for the calculation of local variance, skewness, covariance, auto-correlation and cross-correlation. The fast algorithm which is extended for N-dimensional image is then described. The section following then shows the experimental results obtained using our "plunger" method. The final section gives the concluding remarks.

Fast Algorithms for 2D Images

Local mean calculation is a straightforward technique for image smoothing. Given an N-dimensional image $f(\mathbf{x})$, where $\mathbf{x}$ is an N-dimensional variable $\mathbf{x} = (x_1, x_2, \quad x_N)$, and the size of the image is $X_1 X_2 \quad X_N$. The procedure is to generate a smoothed image $f(\mathbf{x})$ whose gray level at every point $\mathbf{x}$ is obtained by averaging the gray values of the pixels contained in a predefined neighbourhood of $\mathbf{x}$. The size of this neighbourhood or window size is $(2W_1 + 1)\ (2W_2+1) \quad (2W_N+1)$, where W_i is a natural number. The local mean is obtained by:

$$f(\mathbf{x}) = \frac{1}{T} \sum_{m,n,s,\cdots,p \in S} f_{mns \quad p} \tag{1}$$

where S is the set of coordinates of points in the neighbourhood of $\mathbf{x}$ including itself, and T is the total number of points in the neighbourhood. $m, n, s, \cdots, p$ are the indices for the pixels in the neighbourhood, and $f_{mns \quad p}$ is the intensity of f at position $m, n, s, \cdots, p$.

Local variance of an image can be obtained by using the following equation:

$$f(\mathbf{x}) = \frac{1}{T} \sum_{m,n,s,\quad,p \in S} (f_{mns \quad p} - f)^2 \tag{2}$$

In the following subsections, we will first review the 2D box filtering technique for fast calculation of local mean of an image, and then extend the method for fast calculation of local variance, skewness, and other measures at the same time when obtaining the local mean.

Box filtering: Review

A brief review of the 2D box filtering technique is given as follows (a detailed description can be found in [2]). Let f_{mn} be the intensity value of an $X_1 \times X_2$ sized 2D image f at position (m, n), while f is to be box filtered into f, i.e. obtaining the mean of the original image within a local box. Then for a $(2W_1+1) \times (2W_2+1)$ box filter:

$$f_{ij} = \frac{1}{T} \sum_{m=i-W_1}^{i+W_1} \sum_{n=j-W_2}^{j+W_2} f_{mn}, \tag{3}$$

where the denominator T is a constant $(2W_1+1)\ (2W_2+1)$.

The procedures of box filtering can be described as follows (referring to Figure 1). Each output row of f is calculated using the window ABCD in f. The size of this window equals $X_2(2W_1+1)$. A buffer $BUFM_1[X_2]$ is maintained for this window. Each element of $BUFM_1$ is the sum of the pixels in the corresponding column of the window. That is:

$$BUFM_1[j] = \sum_{m=i-W_1}^{i+W_1} f_{mj}, \quad j = 0, \quad , X_2 - 1. \tag{4}$$

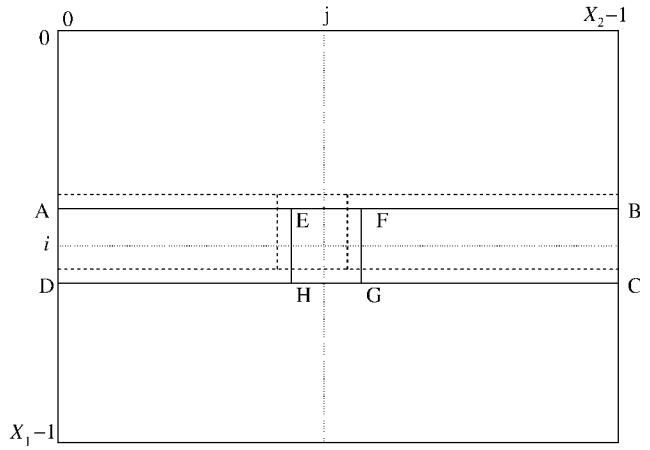

Figure 1. The windowing process of the box filter operation in the 2D case. EFGH indicates a local window or box [2].

After each row of f has been calculated, the window moves down one row, and $BUFM_1$ is updated by adding the new row and subtracting the old one as indicated in Figure 1.

Each f_{ij} is calculated using the box EFGH. A value $SUMM$ is stored for each box position given by:

$$SUMM = \sum_{n=j-W_2}^{j+W_2} BUFM_1[n]. \tag{5}$$

As the box moves horizontally, $SUMM$ is updated by adding in a new $BUFM_1$ value from the right and subtracting out a value from the left. Thus:

$$f_{ij} = \frac{SUMM}{T}. \tag{6}$$

When the first row of f is calculated, $BUFM_1$ must be initialized explicitly. Similar initialization must be performed for the first value of each row of $SUMM$.

Extension to fast calculation of variance

In the 2D case, the following equation (Eqn 7) may be obtained by a rearrangement of Eqn (2). It can be seen that the pixel variance within the box can also be obtained during the same pass as that which calculates the mean. This is achieved by accumulating the square of the intensity values while accumulating original pixel values for mean calculation. The variance of pixels within the box (f_{ij}) is calculated using the last row of Eqn (7).

$$\begin{aligned} f_{ij} &= \frac{1}{T} \sum_{m=i-W_1}^{i+W_1} \sum_{n=j-W_2}^{j+W_2} (f_{mn} - f_{ij})^2 \\ &= \frac{1}{T} \sum_{m=i-W_1}^{i+W_1} \sum_{n=j-W_2}^{j+W_2} (f_{mn}^2 - 2 \times f_{mn} \times f_{ij} + f_{ij}^2) \\ &= \frac{1}{T} \sum_{m=i-W_1}^{i+W_1} \sum_{n=j-W_2}^{j+W_2} f_{mn}^2 - f_{ij}^2 \\ &= S - f_{ij}^2 \end{aligned} \tag{7}$$

where

$$S = \frac{1}{T} \sum_{m=i-W_1}^{i+W_1} \sum_{n=j-W_2}^{j+W_2} f_{mn}^2. \tag{8}$$

The S term in the above equation can be obtained by using a method similar to that which calculates the local mean. A similar buffer $BUFV_1$ is maintained for the first term of the above equation. Each element of buffer $BUFV_1$ contains the sums of the square of pixel values in the corresponding column of the window.

$$BUFV_1[j] = \sum_{m=i-W_1}^{i+W_1} f_{mj}^2, \quad j = 0, \quad , X_2 - 1. \tag{9}$$

After each row of f has been calculated, the window moves down one row, and $BUFV_1$ is updated by adding the new row and subtracting the old one as indicated in Figure 1 at the same time as one updating the buffer $BUFM_1$.

Each f_{ij} is calculated using the same box EFGH. A value $SUMV$ is stored for each box position given by:

$$SUMV = \sum_{n=j-W_2}^{j+W_2} BUFV_1[n]. \tag{10}$$

As the box moves horizontally, $SUMV$ is updated by adding in a new $BUFV_1$ value from the right and subtracting out the value from the left. Thus:

$$f_{ij} = \frac{SUMV}{T} - f_{ij}^2. \tag{11}$$

When the first row of f is calculated, $BUFV_1$ must be initialized explicitly. Similar initialization must be performed for the first value of each row of $SUMV$.

Therefore we have a fast way to obtain the local mean and variance of the input 2D images.

Extension to fast calculation of skewness, autocorrelation, covariance and cross-correlation

The last two subsections describe fast algorithms for obtaining the two most commonly used local statistical measures: mean and variance. This subsection will give fast algorithms for calculating the skewness, covariance, auto-correlation, cross-correlation and higher order central moments within a local window. The covariance and cross-correlation calculations will need two input images.

The formula for the skewness measure within a 2D window is the following:

$$\begin{aligned} f_{ij} &= \frac{1}{T}\sum_{m=i-W_1}^{i+W_1}\sum_{n=j-W_2}^{j+W_2}(f_{mn}-f_{ij})^3 \\ &= \frac{1}{T}\sum_{m=i-W_1}^{i+W_1}\sum_{n=j-W_2}^{j+W_2} f_{mn}^3 - 3Sf_{ij} + 2f_{ij}^3 \end{aligned} \tag{12}$$

where f_{ij} and S are defined in Eqns. (3) and (8). Higher order central moments such as

$$\frac{1}{T}\sum_{m=i-W_1}^{i+W_1}\sum_{n=j-W_2}^{j+W_2}(f_{mn}-f_{ij})^k$$

can also be calculated efficiently. The first term in Eqn (12) can be obtained by using a method similar to that which calculates the local variance. A similar buffer $BUFS_1$ is maintained for the first term of the above equation. Each element of buffer $BUFS_1$ contains the sums of the cube of pixels in the corresponding column of the window.

$$BUFS_1[j] = \sum_{m=i-W_1}^{i+W_1} f_{mj}^3, \quad j = 0, \quad , X_2 - 1. \tag{13}$$

After each row of f has been calculated, the window moves down one row, and $BUFS_1$ is updated by adding the new row and subtracting the old one as indicated in Figure 1 at the same time as one updating the buffer $BUFV_1$. Each f_{ij} is calculated using the same box EFGH.

The autocorrelation of local windows of an image can also be obtained using similar algorithms. If we notionally make a copy of the input image and put it on top of the original input image with a 2D shift (k, l), then the autocorrelation function

$$f_{ij}^{'kl} = \frac{1}{T}\sum_{m=i-W_1}^{i=W_1}\sum_{n=j-W_2}^{j+W_2} f_{mn}f_{m+k,n+l} \tag{14}$$

can be efficiently obtained by accumulating $f_{mn}f_{m+k,n+l}$ instead of f_{mn} for mean, f_{mn}^2 for variance and f_{mn}^3 for skewness calculation.

If there are two images f_{ij} and g_{ij} and local covariance is to be calculated, then the following equation can be used. (k, l are the relative image shifts):

$$c_{ij}^{'kl} = \frac{1}{T}\sum_{m=i-W_1}^{i+W_1}\sum_{n=j-W_2}^{j+W_2} f_{mn}g_{m+k,n+l} \tag{15}$$

All the measures described above are obtained from the image pixel values within a local window with equal weighting. If a non-equal weighting scheme such as the Gaussian smoothing function is necessary, more expensive computation will be needed. For the equal weighting scheme such as those for mean and variance calculation, only two operations are necessary for each pixel on each dimension of the input image. For a relatively fast Gaussian smoothing algorithm, it will need six operations for each pixel on each dimension of the input image [19]. The non-linear min/max operations, which are the fundamental mathematical morphology operations, need three operations for each pixel on each dimension of the input image [20].

Fast Algorithm for nD Images

We will now extend the 2D box filtering technique described in the previous section into N-dimensional cases. As the operation process of our new method is similar to the operation of a coffee plunger, we have named it the ''plunger technique''.

3D images

In the 2D case, the major operation unit is a 2D moving window (or sliding window e.g. window ABCD as shown in Figure 1) which contains the sum of pixels in the corresponding column of the window. In the 3D case, we use a moving volume as the operation unit. The size of this volume is $X_1X_2(2W_3+1)$, where $(2W_3+1)$ is the size of the local window in the X_3 or Z direction. When this volume or 3D box moves from one position to the next, it absorbs one slice of the image from one

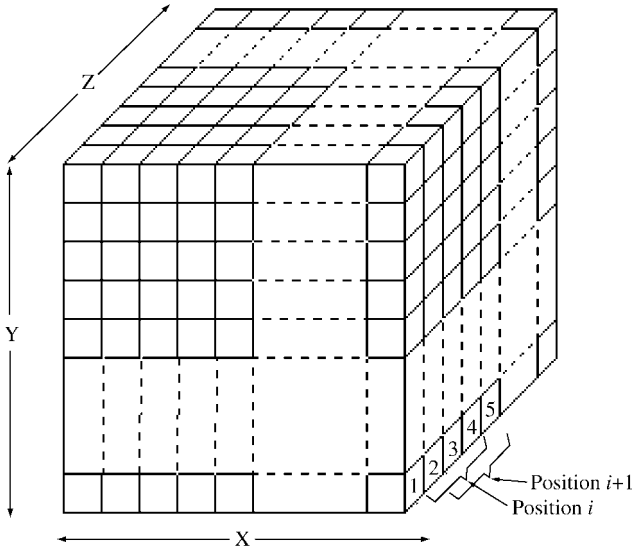

Figure 2. The windowing process of the box filter operation for 3D images [2].

side of the volume and releases another slice of the image from the other side. The image values within this volume are accumulated into one 2D array.

If we imagine that the filter in a coffee plunger has certain thickness, this filter holds a certain amount of coffee. When we push the plunger down, some coffee comes into the filter from the bottom side. At the same time some coffee will be pushed out of the filter. We will use a similar process to filter 3D and multiple dimensional images. The moving volume acts like the filter in the coffee plunger. This volume moves in the X_3 or Z direction and collects/releases the pixel values along the way (see Figure 2). The situation shown in the figure is a local volume with $W_3 = 1$ at position i which contains three image planes: 2, 3, 4. When the local volume moves from position i to position $i+1$, the image plane number 5 moves into the volume, while image plane number 2 moves out of the volume.

Similar to the 1D buffers used in the 2D image situation, three 2D arrays or buffers $BUFM_2$, $BUFV_2$ and $BUFS_2$ need to be created ($BUFM_2$ for mean calculation, $BUFV_2$ for variance calculation and $BUFS_2$ for skewness calculation). If we do not need to obtain the variance or skewness measures, the temporal array $BUFV_2$ or $BUFS_2$ is not necessary. These arrays store the summation of pixel values, squares of pixel values and cubes of pixel values along the X_3 axis:

$$BUFM_2[i, j] = \sum_{s=k-W_3}^{k+W_3} f_{ijs} \tag{16}$$

$$BUFV_2[i, j] = \sum_{s=k-W_3}^{k+W_3} f_{ijs}^2 \tag{17}$$

$$BUFS_2[i, j] = \sum_{s=k-W_3}^{k+W_3} f_{ijs}^3 \tag{18}$$

At each position (k) of the moving volume, we then have three 2D arrays that contain the summation of pixel values, the squares of pixel values and the cubes of pixel values along the X_3 direction within the volume. We can then use these three 2D arrays as input to calculate 3D local mean, variance and skewness using the method described previously. Because buffer $BUFV_2$ ($BUFS_2$) has already performed the square (cube) of the pixel values, when calling the 2D functions, the $BUFV_1$ ($BUFS_1$) buffer does not need to perform square (cube) operations for the original pixel values. $BUFV_1$ ($BUFS_1$) just needs to accumulate the values in buffer $BUFV_2$ ($BUFS_2$).

4D images

A 4D image is a sequence of 3D volume images taken at different times (see Figure 3). We can also extend the procedure for 3D images to work for 4D images for the fast calculation of local mean, variance and skewness in one pass.

In the 3D case, the major operation unit is a 3D moving volume which contains the sum of pixels, sum of square of pixel values and sum of cube of pixel values in the X_3 axis. In the 4D case, we use a local 4D moving sequence or box as the operation unit. The size of this sequence is $X_1X_2X_3$ ($2W_4+1$), and ($2W_4+1$) is the size of the local "window" in the X_4 direction. When this 4D box moves from one position to the next, it absorbs one volume of the image from one time position of the sequence and releases another volume of the image from the other time position. The image values within this 4D box are accumulated into three 3D arrays. We will use similar process to filter 4D images as the process for 3D cases (see Figure 2).

Similar to the buffer used in the 3D situation which is a 2D buffer, three 3D arrays or buffers $BUFM_3$, $BUFV_3$, and $BUFS_3$ need to be created for 4D images. These

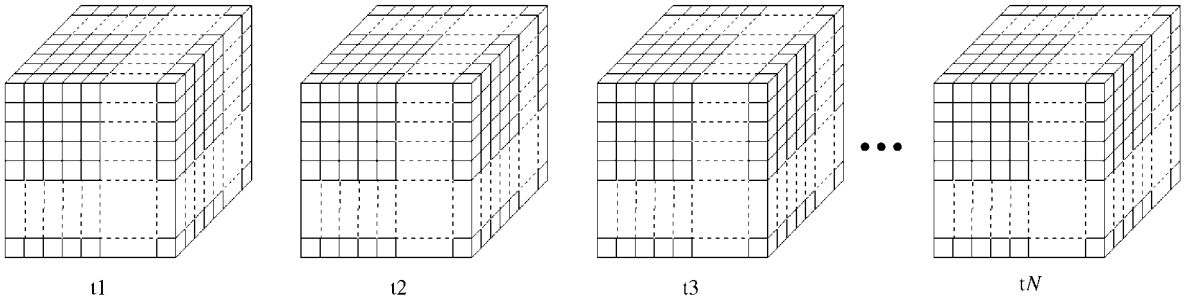

Figure 3. The windowing process of the box filter operation for 4D images.

arrays store the summation of pixel values, squares of pixel values and cubes of pixel values along the X_4 axis:

$$BUFM_3[i,j,k] = \sum_{t=l-W_4}^{l+W_4} f_{ijkt} \tag{19}$$

$$BUFV_3[i,j,k] = \sum_{t=l-W_4}^{l+W_4} f_{ijkt}^2 \tag{20}$$

$$BUFS_3[i,j,k] = \sum_{t=l-W_4}^{l+W_4} f_{ijkt}^3 \tag{21}$$

At each position of the moving volume (l), we then have three 3D arrays that contain the summation of pixel values, the square of pixel values and cube of pixel values along the X_4 direction within the sequence. We can then use these three 3D arrays as input(s) to calculate 4D local mean, variance and skewness using the method described for 3D images.

nD images

Again we can extend the plunger technique to higher dimensions. For N-dimensional images, we can create three $(N-1)$-dimensional arrays or images $BUFM_{N-1}$, $BUFV_{N-1}$ and $BUFS_{N-1}$ as given below which contain the sums of pixel values, sums of the square of pixel values, and sums of cube of pixel values. The process can be recursively carried out so that in the end we have 2D arrays, and we can use the fast method in the 2D case for the calculation of local mean, variance, skewness and other statistical measures.

$$BUFM_{N-1}[i,j,k,\ \ ] = \sum_{p=r-W_N}^{r+W_N} f_{ijk\ \ p} \tag{22}$$

$$BUFV_{N-1}[i,j,k,\ \ ] = \sum_{p=r-W_N}^{r+W_N} f_{ijk\ \ p}^2 \tag{23}$$

$$BUFS_{N-1}[i,j,k,\ \ ] = \sum_{p=r-W_N}^{r+W_N} f_{ijk\ \ p}^3 \tag{24}$$

Algorithm complexity

Assume we have an equal sided (size X for each dimension) N-dimensional image and we would like to smooth the image with an equal sided (size W for each dimension) box using simple direct averaging (mean calculation). For each position of the local window, one needs to add all the pixel values within the window. The size of the local window is W^N. The number of possible local window positions are X^N if we ignore the boundary effect. The direct implementation would take on the order of $W^N X^N$ additions.

For the proposed "plunger technique" for N-dimensional images, the computation involved only has one addition and one subtraction for each pixel in each dimension. So we will have $2N$ additions(subtractions) for each pixel for all the N dimensions. Therefore, we have a total of $2NX^N$ additions for the "plunger technique" which is independent of the local window size ignoring the effect when processing the pixels values at image boundaries. Both of the direct and the plunger techniques need to have X^N number of divisions (one division for each pixel in the image).

For the calculation of variance and skewness, multiplications need to be carried out. The multiplications only happen when we reduce the dimensionality the first time using the plunger technique. For an N-dimensional image, the multiplications (square of pixel value, cube of

pixel value) are necessary only when we reduce the image dimension from N to $N-1$. In all the later stages of dimensionality reduction, there will be no multiplications. The above discussion ignores the processing of the pixels on the input image boundaries.

Steps for the fast algorithm

Our proposed algorithm for fast calculation of local statistics for an N-dimensional image is:

1. Given an N-dimensional image, and the box sizes $(2W_1+1)$, $(2W_2+1)$, , $(2W_N+1)$
2. Maintain two (or three) buffers $BUFM_{N-1}$ and $BUFV_{N-1}$ (or $BUFS_{N-1}$) which contain the sum of pixel values and the sum of square of pixel values (or cube of pixel values)
3. Recursively reduce the image dimension until the image dimension is 2, i.e. reduced to 2D images/arrays
   (a) Generate an $(N-1)$-dimensional image/plane from the original N-dimensional image using the "plunger" technique described.
   (b) Further reduce the dimension of the $(N-1)$-dimensional image until the dimension is 2.
4. Then, use the 2D box filtering method for fast calculation of the local mean and variance (or skewness) on an N-dimensional image.

Experiment Results

In order to remove the boundary effect, pixel padding can be performed at image boundaries. The padded pixels can take the mirrored pixel values along the image boundaries. When the image is one-dimensional, the box size along the insignificant dimension is set to 1. The sizes of each side of the N-dimensional box need to be odd to prevent pixel position shift of the resultant image, but their sizes along each of the dimensional axes do not have to be the same. That is, the $W_i s$ do not have to be equal.

When using our fast mean estimator, the execution time is approximately proportional to the number of pixels in the image to be smoothed and is largely independent of the smoothing filter size. This makes the routine particularly suitable for applying heavy smoothing to an image.

We have tested different type of images with different dimensions. Table 1 shows the comparison of our new plunger algorithm and a direct implementation of the mean and variance calculation algorithm for 1D and 2D images. The first two columns are the image names and image sizes. The third column is the window sizes used. There is no image boundary padding for the timing shown in this table. "User time 1" is the time spent when using our new plunger method for calculating the local mean and variance. "User time 2" is the time spent by a direct implementation of local mean and variance calculation. The time spent by our new plunger method is almost constant while the time spent by the method of direct implementation is almost linear increasing with the size of the local window. The decrease in "User time 1" with increases of the local window size is because of the image boundary effect. That is, the pixels along the image boundary were not calculated.

Table 2 gives the running times of our algorithm on several images with and without image boundary padding. The first column of the table gives the names

Table 1. Running times of the fast plunger algorithm and the direct implementation for local mean and variance calculation on different types of images. The timing shown in this table is when the algorithms do not have boundary padding. The timing under column "User time 1" is from our fast plunger method; while the timing under column "User time 2" is from the algorithm using direct implementation. Times are expressed in seconds.

Image name	Image size	Window sizes	User time 1 (s)		User time 2 (s)	
			Mean	Variance	Mean	Variance
line (1D)	256×1	3×1	0.0002	0.0005	0.0003	0.0007
		5×1	0.0002	0.0005	0.0004	0.0008
		7×1	0.0002	0.0005	0.0004	0.0009
		9×1	0.0002	0.0005	0.0005	0.0011
trui (2D)	256×256	3×3	0.0540	0.1051	0.1169	0.2943
		5×3	0.0518	0.1050	0.1463	0.4338
		7×5	0.0477	0.1047	0.2831	0.9237
		9×7	0.0476	0.1010	0.4564	1.5889

Table 2. Running times of our fast plunger algorithm on different types of images. The timing under column "User time A" is when the algorithm does not have boundary padding; while the timing under column "User time B" is when the algorithm does have boundary padding. The percentage of increase of computation time when using image padding for local mean and variance are given in columns indicated by %**m** and %**v** respectively. Times are expressed in seconds.

Image name	Image size	Window sizes	User time A (s)		User time B (s)			
			Mean	Variance	Mean	% **m**	Variance	% **v**
line (1D)	256×1	3×1	0.0002	0.0005	0.0005	60	0.0006	16
		5×1	0.0002	0.0005	0.0005	60	0.0007	28
trui (2D)	256×256	3×3	0.0540	0.1051	0.0720	25	0.1427	26
		5×3	0.0518	0.1050	0.0756	31	0.1477	28
mr3d (3D)	$85 \times 85 \times 82$	$3 \times 3 \times 3$	0.4251	0.9193	0.6428	33	1.3351	31
		$7 \times 5 \times 3$	0.4133	0.9050	0.6625	37	1.4215	36
98c (4D)	$64 \times 64 \times 26 \times 8$	$3 \times 3 \times 3 \times 1$	1.0043	2.2155	1.5092	33	3.0263	26
		$5 \times 5 \times 3 \times 3$	0.8359	1.7157	1.7012	50	3.3057	48
s1p (4D)	$128 \times 128 \times 15 \times 22$	$3 \times 3 \times 3 \times 3$	6.6376	13.8945	10.4750	36	20.6662	32
		$7 \times 5 \times 3 \times 3$	6.6174	13.7001	10.6435	37	22.1278	38

of images and the image dimensionality. The second column shows the image sizes, and the third column shows the sizes of local windows used. The remaining columns are the running times for our new algorithm. The time measurements were obtained by running the program hundreds of times and taking the average. "User time A" indicates the time spent on the image without image boundary padding. "User time B" gives the time of the algorithm with image boundary padding. The increase of program running time for "User time B" compared with "User time A" results from two factors: one is the process of performing space allocation and data copying for generating a larger image; the second is that the program is running on a larger image. The increase of computation time when image padding is used is shown in Table 2. The column indicated by "% m" shows the percentage of time increase for local mean calculation. The first two values in this column are not accurate as the computation time is very short. The column indicated by "% v" shows the percentage of time increase for local variance calculation. Therefore, one should be aware of the cost for processing the pixels near the image borders. It will save some computation time by leaving the boundary pixels unprocessed.

The times spent by the program on one image when using different window sizes are almost the same. That means the speed of the algorithm is almost invariant to the local window size. The slight difference is because of the change of the image size when considering the image boundary effect. For example, for image `mr3d`, when the sizes of local window change from $3 \times 3 \times 3$ to $7 \times 5 \times 3$, "User time A" reduced from 0.4251 s to 0.4133 s for the mean calculation. This is due to the reduced effort from not processing the boundary pixels of the input image.

The computer used was a relatively old 85 MHz Sun SPARCserver1000 running Solaris 2.5. The typical running time for the algorithm on a 256×256 2D image is in the order of several dozens of milliseconds.

Conclusions

We have developed a fast algorithm for calculating local statistics for N-dimensional images. The fast calculation was realized by using the plunger technique which recursively reduces the dimensionality of the input N-dimensional image. The time spent in the stage which obtains the local statistics is almost invariant to the local window size. The local mean, variance and skewness calculation for the N-dimensional image can be performed in one single pass. The typical running time for a 256×256 2D image on an 85 MHz computer is in the order of several dozens of milliseconds.

Acknowledgments

The author thanks the anonymous referees for their comments and suggestions. We would like to acknowledge Professor David Feng of the University of Sydney for providing one 4D image and giving comments on the manuscript. We thank Brian Hutton of the Medical Physics Department of Westmead Hospital for providing another 4D image. We are also grateful for Weidong

Cai and Leanne Bischof in helping obtain some other images. The help from Ryan Lagerstrom in reading the manuscript is also acknowledged.

References

1. Jain A.K. (1989) *Fundamentals of Digital Image Processing*. Pentice-Hall, Englewood Cliffs, New Jersey.
2. McDonnell M.J. (1981) Box-filtering techniques. *Computer Graphics and Image Processing* **17**: 65–70.
3. Sun, C. (1997) A fast stereo matching method. In *Digital Image Computing: Techniques and Applications*, Massey University, Auckland, New Zealand, December 10–12 1997, pp. 95–100.
4. Sun, C. (1999) Fast optical flow using cross correlation and shortest-path techniques. In *Digital Image Computing: Techniques and Applications*, Perth, Australia, 7–8 December 1999, pp. 143–148.
5. Sun, C. (1999) Multi-resolution stereo matching using maximum-surface techniques. In *Digital Image Computing: Techniques and Applications*, Perth, Australia, 7–8 December 1999, pp. 195–200.
6. da Fontoura Costa, L. (1999) Gauss' law in image processing and analysis via fast numerical calculation of vector fields. *Real-Time Imaging* **5**: 243–251, doi:10.1006/rtim.1998.0136.
7. Wu, G.K. & Reed, T.R. (1999) Image sequence processing using spatiotemporal segmentation. *IEEE Transactions on Circuits and Systems for Video Technology* **9**: 798–807.
8. Doyen, P.M. (1988) Porosity from seismic data: A geostatistical approach. *Geophysics* **53**: 1263–1275.
9. Joshi, S. & Miller, M.I. (1993) Maximum a posteriori estimate with Good's roughness for three-dimensional optical-sectioning microscopy. *Journal of Optical Society of America, Part A* **10**: 1078–1085.
10. Kriete, A., Rohrbach, S. & Schwebel, T. (1992) Data representation and visualization in 4D microscopy. In *Proc. SPIE Conf. Visual. in Biomed. Comp., volume 1808*, pp. 396–409.
11. Robb, R.A. & Barillot, C. (1989) Interactive display and analysis of 3D medical images. *IEEE Transactions on Medical Imaging* **8**: 217–226.
12. Hoffman, E.A., Gnanaprakasam, D., Gupta, K.B., Hoford, J.D., Kugelmass, S.D. & Kulawiec, R.S. (1992) VIDA: An environment for multidimensional image display and analysis. In *SPIE Conf. Biomedical Image Processing and Three-Dimensional Microscopy, volume 1660*, pp. 1–18.
13. Lee J.H. (1992) *4D digital topology and mathematical morphology*. Master's thesis, The Pennsylvania State University, University Park, PA.
14. Higgins, W.E. & Lee, J.H. (1993) 4D morphological processing of cardiac image sequences. In *15th Annual Int. Conf. IEEE Engin. Medicine & Biology Soc.*, San Diego, CA, 28–31 October 1993, pp. 136–137.
15. Wang, A.J. (1995) *Cue-based analysis of 4D cardiac images*. Master's thesis, The Pennsylvania State University, University Park, PA.
16. Higgins, W.E., Wang, A.J. & Reinhardt, J.M. (1996) Semi-automatic 4D analysis of cardiac image sequences. In *SPIE Medical Imaging 1996: Physiology and Function from Multidimensional Images, volume 2709*, Newport Beach, CA, 11–13 Feb. 1996, pp. 359–372.
17. Niessen, W.J., Duncan, J.S., Florack, L.M.J., ter Haar Romeny, B.M. & Viergever M.A. (1995) Spatiotemporal operators and optic flow. In T.S. Huang and D.N. Metaxas (eds), *Physics-Based Modeling in Computer Vision*, pp. 78–84. IEEE Computer Society Press.
18. Rohr, K. (1997) On 3D differential operators for detecting point landmarks. *Image and Vision Computing* **15**: 219–233.
19. Vliet, L.J.V., Young, I.T. & Verbeek P.W. (1998) Recursive Gaussian derivative filters. In *Proceedings of International Conference on Pattern Recognition, volume I*, Brisbane, Australia, 16–20 August 1998. IEEE Computer, Society, pp. 509–514.
20. van Herk, M. (1992) A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. *Pattern Recognition Letters* **13**: 517–521.