Edge detection in a hexagonal-image processing framework

Article *in* Image and Vision Computing · December 2001
DOI: 10.1016/S0262-8856(01)00067-1 · Source: DBLP

2 authors:

Lee Middleton
University of Southampton
65 PUBLICATIONS **713** CITATIONS
SEE PROFILE

Jayanthi Sivaswamy
International Institute of Information Technology, Hyderabad
120 PUBLICATIONS **1,380** CITATIONS
SEE PROFILE

ELSEVIER

Image and Vision Computing 19 (2001) 1071–1081

image AND vision COMPUTING

www.elsevier.com/locate/imavis

Edge detection in a hexagonal-image processing framework

Lee Middleton*, Jayanthi Sivaswamy

Department of Electrical and Electronic Engineering, The University of Auckland, Private Bag 92019, Auckland, New Zealand

Received 14 December 1999; received in revised form 2 April 2001; accepted 7 June 2001

Abstract

With processing power of computers and capabilities of graphics devices increasing rapidly, the time is ripe to reconsider using hexagonal sampling for computer vision in earnest. This paper reports on an investigation of edge detection in the context of hexagonally sampled images. It presents a complete framework for processing hexagonally sampled images which addresses four key aspects: conversion of square to hexagonally sampled images, storage, processing, and display of these images. Results from using edge detection on this framework show that (a) the computational requirement for processing a hexagonally sampled image is less than that for square sampled images, and (b) a better qualitative performance which is due to the compact and circular nature of the hexagonal lattice. This last point needs to be exploited in the development of edge detectors for hexagonally sampled images.

Keywords: Hexagonal-image processing; GBT; Edge detection

1. Introduction

As the domains to which computer vision is applied expands, the characterisation of the image increasingly depends on gathering information about the structures contained within an image. These structures are often made up of irregular curves. It is thus of paramount importance to find efficient methodologies to represent and analyse these structures. One such method that is commonly employed is edge detection. However, the success of edge detection is often limited when it comes to curved features. This is primarily due to the algorithm's effectiveness being limited by the quality of the input image. Consequently, high resolution grids are often used in order to represent the data with sufficient fidelity. However, this comes at a cost and that is a resulting decrease in computational performance. Another way to overcome this difficulty is via the use of alternative sampling lattices. Sampling on a hexagonal lattice is a promising solution which has received some attention and been shown to have better efficiency and less aliasing [13]. Hexagonal pixels are also attractive as they exploit the oblique effect in human vision [6]. This makes humans less sensitive to edges in the diagonal rather than horizontal or vertical directions. Hence our objective is to investigate the use of a hexagonally sampled image for edge detection. Specifically, we wish to investigate the merit of using a hexagonal sampling regime for implementing various operations that are called for by different edge detection techniques. We perform this by implementing various edge detection schemes on a framework that we have developed specifically for processing hexagonally sampled images. In this paper, we report our findings and present our processing framework which has been used for other applications as well [14–16]. For convenience, we denote hexagonally sampled images as hexagonal-images and square sampled images as square-images.

We begin with a review of the field of hexagonal-image processing in Section 1 followed by details of our framework for processing hexagonal-images, in Section 3, that was used in our investigation. Details of our investigation on edge detection on hexagonal-images along with a comparison with edge detection using square-images is presented in Section 4. We conclude the paper in Section 5, with a discussion of the computational requirements and overall benefits of edge detection with hexagonal-images.

2. Background

Hexagonal-image processing is not a new idea. Over the past 30 years many researches have examined various aspects of this field. We review the literature in the following, in roughly chronological order.

Theoretical studies of hexagonal sampling can be traced to begin with Peterson [19], who considered it in the context of a possible alternative sampling regime for a

* Corresponding author.
E-mail address: l.middleton@auckland.ac.nz (L. Middleton).

0262-8856/01/$ - see front matter
PII: S0262-8856(01)00067-1

2-D Euclidean space. Petersen concluded that the most efficient sampling schemes were not based on square lattices. The next significant work can be dated to Rosenfeld [20] who studied distance functions on digital pictures using a variety of coordinate systems. Simple morphological operators were then evaluated in these coordinate systems via some simple applications. Finally, Rosenfeld had a suggestion for displaying the hexagonal-images by staggering the individual square pixels like a 'brick wall'. Around the same time, Golay [5] developed parallel computing systems based on hexagonal structures. Later Mersereau [13], motivated by computational savings offered by hexagonal sampling consolidated the earlier work by formulating the hexagonal sampling theorem. This was used to derive the discrete hexagonal Fourier transform. Mersereau also showed that hexagonal sampling produced fewer samples than square sampling for the same data. Serra [21] later consolidated and extended the earlier work particularly as it pertains to mathematical morphology. The theory was as applicable to hexagonal systems as to square ones. Serra evaluated a variety of problems and in each one chose a suitable coordinate system. Generally, a preference for hexagonal lattices has been shown in the work due to their consistent connectivity.

The first fundamental work on storing hexagonal-images was performed by Burt [2] whose motivation was to find efficient storage schemes for an image using tree and pyramid data structures. The 'sept-tree' which was composed of a central hexagonal cell surrounded by six others was advocated as the most useful structure as it was roughly hexagonal in shape. Around the same time, Gibson and Lucas [4] studied the problem of conversion of map data to other forms for graphical information systems and devised a data structure known as the Generalised Balanced Ternary (GBT). The GBT consists of a hierarchy of cells with the cells at each level constructed from those at a previous level using an aggregation rule. For the 2-D case, the aggregation rule is the same as that of Burt's 'sept-tree'. As each level consists of clusters of seven cells, any GBT-based structure can be represented by unique, base-seven indices.

Independent of the work on GBT-based data structures, Hartman and Tanimoto [7] investigated generating hexagonally-based pyramid structures as well, for the purpose of modelling the behaviour of the orientation- and location-specific cells in the primary visual cortex. The structure they devised was based on aggregations of triangular pixels which resulted in an image which was hexagonal in shape. Each pixel in the image was labelled uniquely by a three-tuple of coordinates: the first being the level of the pyramid and the other two being the position within the level. The position was measured on a coordinate system with the x-axis at 0° and the y-axis at 120°. Edge detection was tested using this structure.

The superiority of hexagonal sampling lattice over the square lattice was established at a theoretical level by Whitehouse [26] for cases of high data density. This was done on the basis of a quantitative comparison of the three regular tilings (with triangles, squares, and hexagons) of the plane and their associated lattices. van Roessel [25], re-examined the GBT and improved the conversion between GBT and Cartesian coordinates. The algorithm for conversion form GBT to Cartesian coordinates was very quick but the reverse conversion was slow as interpolation was required to find the points.

Noting that the receptors of the eye are laid out in a hexagonal ensemble Watson and Ahumada [1] proposed a hexagonal orthogonal-oriented pyramid structure for image coding. Drawing from biological precepts they generated a structure that was similar to the pyramid of Burt and the GBT. Groups of 7 px were used as kernels to represent the image. The image coding regime derived from this showed a high degree of efficiency. Again, the coordinate system used was based upon a skewed axis, in this case the y-axis being at 60°.

In the same year, Staunton [22–24] performed some work on a practical, hardware-based system for hexagonal-image processing. First a conventional frame grabber was modified to perform pseudo-hexagonal sampling by offsetting alternate rows by half an element. The resulting data was processed using a pipeline computer system. Simple algorithms were examined such as thresholding and local operators and the processed images were displayed on a computer screen using a 'brick wall' approach.

Wüthrich and Stucki [28], performed an algorithmic comparison of square and hexagonal lattices. Again hexagonal coordinates were represented on a skewed coordinate system, this time the x-axis was rotated by 60° anti-clockwise. Various graphics algorithms were implemented and compared using both square and hexagonal lattices. The comparison was performed using a simulator, which represented a hexagonal pixel as an accumulation of square pixels. The results showed that the hexagonal- and square-based systems compared favourably from an algorithmic point of view.

Based on many years of experience in modelling the limits of human visual perception, Overington [17] proposed a matrix model for early human vision. The sampling step involved a simple reordering of the input image data reducing every eight rows to seven to approximate ideal hexagonal sampling. The data was displayed on an offset 'brick wall'. Low level image operations were tested within this framework and the model appeared consistent with many aspects of the early human visual system.

Her [8,9] was interested in representing hexagonal data and thus focussed on two aspects. The first was re-sampling a square-image to a hexagonal-image. The effectiveness of different sampling kernels were examined in the re-sampling process. Display of the hexagonal-image again used the 'brick wall' approach. Her also proposed a new three-coordinate system to represent hexagonal data and

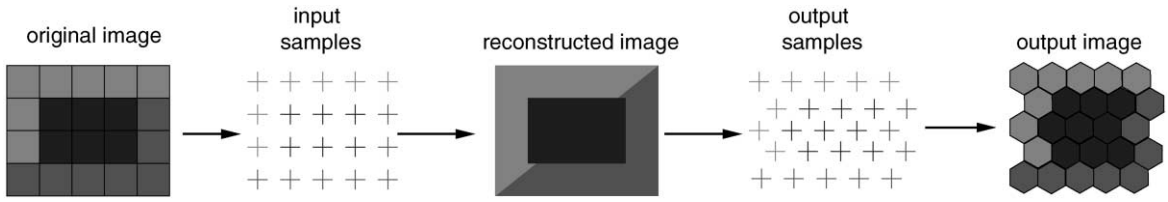


Fig. 1. Image re-sampling on a hexagonal grid.

used a three-tuple to represent the displacement from each of the major axes of the hexagon. A variety of operators and translations were defined on this framework.

Two points can be concluded from the above review. First, the merits of hexagonal sampling in general, and for image processing in particular have been widely acknowledged. However, a solution for a complete practical system for processing hexagonal-images is lacking. Second, edge detection has by and large not been studied in the context of hexagonal sampling. We address these points in the rest of the paper.

3. The hexagonal-image processing framework

In developing a framework for hexagonally sampled images the issue of practicality is particularly important since an abundance of hardware for square sampling is available at the moment and it is unlikely that this situation will change in the near future. Therefore, efforts to switch to a hexagonal sampling scheme have to include finding the means to adapt the existing systems to this scheme. We have identified the following as major issues involved in practical hexagonal-image processing: conversion of a given square-image to a hexagonal-image, storing this image, performing operations upon this image, and finally displaying the hexagonal-image on a normal graphics device using square pixels. We now present the details of our framework for performing hexagonal-image processing which we call the HIP framework.

3.1. Square- to hexagonal-image conversion

Let an image be represented by a real function, $f(x)$, where $x = (x_1, x_2)$ is a spatial variable. To generate a sampled image, $f_s(x)$, one uses an appropriate sampling kernel $h(x)$. For the case of images this process can ideally be written as:

$$f_s(x_1, x_2) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} f(k_1, k_2) h(x_1 - k_1, x_2 - k_2)$$

However, in practice what is commonly available is the sampled image using a square lattice, $f_{ss}(x)$. Given this $f_{ss}(x)$, to generate an image $f_{sh}(x)$ with a hexagonal sampling lattice, a process known as image re-sampling is employed [27]. This process is illustrated in Fig. 1.

The reconstructed image is not calculated explicitly but is implicit within the re-sampling scheme. In the re-sampling process, the main problem is to determine the locations of the input and output samples relative to the reconstructed image. These are the points in the square and hexagonal lattices, respectively.

A lattice or grid is defined as a set of points which are the centres of periodic tilings of the plane. Let $\mathbf{B} = \{b_1, b_2\}$ be a set of basis vectors for the plane. The set $\mathbf{B}$ will define a lattice defined by:

$$L_{\mathbf{B}} = \{n_1 b_1 + n_2 b_2 : n_i \in Z, i = 1, 2\}$$

In the above Z is the set of integer numbers. Different basis vector sets lead to different lattices. For instance:

$$\mathbf{B}_S = \{(1, 0), (0, 1)\}$$

will generate a square lattice. There are many ways to generate a hexagonal lattice but two convenient ones are:

$$\mathbf{B}_{H1} = \left\{(1, 0), \left(\frac{1}{2}, \frac{\sqrt{3}}{2}\right)\right\}$$

$$\mathbf{B}_{H2} = \left\{\left(\frac{\sqrt{3}}{2}, \frac{1}{2}\right), (0, 1)\right\}$$

The sets $\mathbf{B}_{H1}$ and $\mathbf{B}_{H2}$ are related by a simple rotation. For reasons discussed in Section 3.4 we have chosen $\mathbf{B}_{H1}$. In examining the relationship between $\mathbf{B}_{H1}$ and $\mathbf{B}_S$ we note the following. Only b_2 is different which means that the horizontal spacing in the two lattices is the same (as shown in Fig. 2).

In both $\mathbf{B}_{H1}$ and $\mathbf{B}_{H2}$ the individual basis vectors are dependent on each other. This logically will result in some redundancy. For example, in $\mathbf{B}_{H1}$, the second basis vector depends on the first. If an image is re-sampled from an $n \times n$ square-image then the choice of $\mathbf{B}_{H1}$ will result in one of two possibilities: (a) horizontal lattice spacing is fixed, or (b) vertical lattice spacing is fixed. If

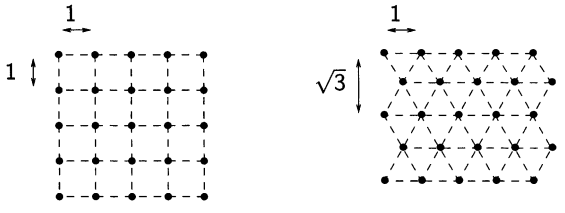
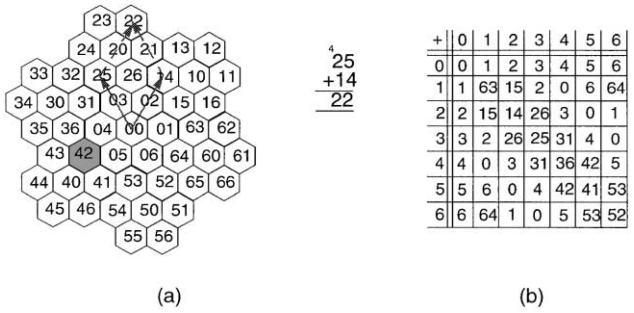


Fig. 2. Relative spacing of sample points in square and hexagonal lattices.



+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	63	15	2	0	6	64
2	2	15	14	26	3	0	1
3	3	2	26	25	31	4	0
4	4	0	3	31	36	42	5
5	5	6	0	4	42	41	53
6	6	64	1	0	5	53	52

Fig. 3. (a) The hexagonal image structure with indices (b) Balanced Ternary addition.

the horizontal lattice spacing is fixed to give n points, it will result in approximately 15% extra vertical lattice points due to its closer packing (due to the second basis vector). If a naive approach to display is used, such as the brick-wall approach, then this will result in an elongated image. Alternatively, if the vertical lattice spacing is fixed so as to only yield n points, it will generate an image that is exactly $n \times n$ in size but will have an inappropriate aspect ratio. This problem can be alleviated by taking advantage of the display device's geometry [9]. The best option is the first of these approaches and take measures to avoid the elongation effect altogether. The issue of displaying the resulting data is discussed in detail in Section 3.4.

The last issue of relevance in re-sampling is the sampling kernel. The choice of sampling kernel is beyond the scope of this paper and hence will not be discussed here. A good review on the choice of the re-sampling kernel for general applications is given in Ref. [27]. The specific kernel that was used in the work discussed here was a bi-linear kernel. This can be defined as:

$$h(x) = \begin{cases} 1 - \|x\|, & \text{if } 0 \le \|x\| < 1 \\ 0, & \text{if } \|x\| \ge 1 \end{cases}$$

where x is as defined earlier and $\| \|$ is the standard Euclidean norm. A review on the choice of kernel for hexagonal lattices is given in Ref. [9]. We have observed that the above bi-linear kernel is adequate for most applications.

3.2. Addressing and storage issues

Once a given square-image is re-sampled to a hexagonal-image, a way to address the pixels in the latter needs to be devised. Symmetry can be exploited for this purpose as follows. Consider a regular hexagonal tiling. For every hexagonal tile there are six other tiles with a shared boundary due to symmetry. This can be alternatively viewed as follows: if we label the central tile as layer 0, we have in layer 1, the next layer, a total of 7 (six neighbouring and the central) tiles. The total number of tiles in layer 2 is equal to the number of tiles in layer 1 plus 42 tiles, due to symmetry. It is straightforward to show that the number of tiles in layer L is 7^L. This comes about because every layer acts as a 'super-tile' for the next layer.

The proposed indexing scheme for hexagonal-images exploits these relationships to identify every pixel with the layer and the position it occupies in the layer, using a base 7 number. The first layer is composed of a central pixel labelled 0 with 6 adjoining pixels numbered 1 through 6 in an anti-clockwise direction. This hexagonally arranged cluster is used as a template to generate the position information for all other layers as shown in Fig. 3a. For example, consider the pixel labelled 42_7. The use of two digits indicates that there are two layers. The first digit (4) indicates that the pixel is positioned in the fourth location of the super-tile in layer 1 and the last digit (2) indicates that it is in the second location of the central tile. An important consequence of this indexing scheme is that all points in the image can be represented by a single coordinate. The proposed indexing system is therefore, a modified form of the GBT [3] system.

The single-index system for pixel addresses has several advantages. It permits a full exploitation of the symmetry in the hexagonal lattice. It also allows the image to be stored using a vector. One coordinate for each pixel also facilitates efficient implementation of certain classes of image processing operators as described in Section 3.3. In comparison with other addressing schemes [7,9] proposed in the literature for hexagonal-images, the single index scheme makes the process of re-sampling the square-image more efficient. This is because converting Cartesian coordinates into indices for the image re-sampling process requires only a single loop. It is possible to use a single index for square-sampled images as well. This is generally done for computational efficiency by reordering the rows or columns into a vector and manipulating the pointer into this vector. In contrast such an extra step is not required for the HIP addressing scheme.

The storage requirements for a hexagonal-image depend on two factors: the image resolution and the number of colour levels that are being used in the image. For instance, a square-image of size $M \times N$ with 24 bit colour will have a size of $3MN$ bytes. Similarly for a hexagonal-image using L layers the total number of pixels is 7^L, and for a 24 bit colour image the required storage space is 3×7^L bytes. For $M = N = 2^m$, the relationship between L and m can also be computed as below. Thus, for any given square-image an appropriate number of layers for the hexagonal image can be chosen.

$$L = m\frac{2\log 2}{\log 7} \approx 0.71m$$

3.3. Image processing operations

Neighbours of a pixel is one of the basic relationships

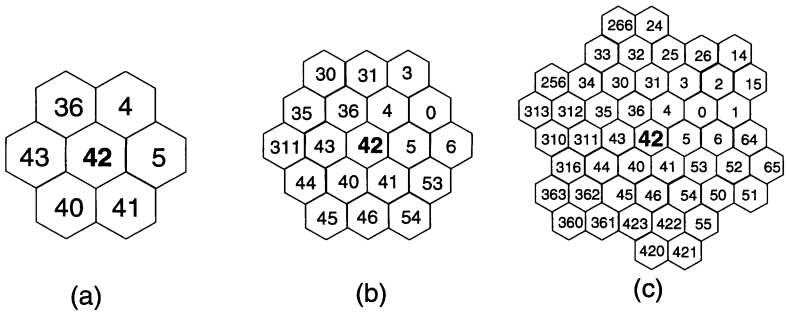


Fig. 4. Neighbourhoods of pixel 42 (a) N_1 (b) N_2 (c) $N_{g(2)}$.

of interest in image processing with the concept of a neighbourhood used in many processing techniques. Two cells in a lattice (pixels) can be regarded as neighbours when they have a common edge or when they have at least one joined corner. For square-images this definition permits two possibilities for smallest neighbours, namely, the 4-neighbourhood and the 8-neighbourhood, with the pixels in the former being at a distance 1 and the latter at a distance $\sqrt{2}$ from the central pixel. For a hexagonal-image however, the only permissible smallest neighbourhood is the 6-neighbourhood with all pixels equidistant (at distance 1) from the central pixel, an important feature. Other neighbourhoods can be defined by increasing the distance from the central cell. These neighbourhoods can be defined by walking, at most, this distance from the central cell.

Thus, in a hexagonal-image each pixel has 6 neighbourhood pixels in the first neighbourhood denoted by N_1. N_1 is a hexagon surrounding the central pixel. The next neighbourhood contains 19 px and is denoted by N_2 (containing the pixels in N_1 plus 12 others). The hexagonal image structure also permits another sort of neighbourhood defined by aggregations of super-tiles in the same way as the data structure was originally defined. This neighbourhood is denoted by $N_{g(i)}$ and the first such neighbourhood, $N_{g(1)}$, is equivalent to N_1. The neighbourhood denoted by $N_{g(i)}$ is much more circular than the ordinary neighbourhood.

Finding the neighbours in a hexagonal-image is a simple exercise, irrespective of which neighbourhood definition is used. It makes use of the addition operation of Balanced Ternaries [10]. The addition of two numbers can be visualised in a fashion similar to the addition of vectors shown in Fig. 3a. This process can be repeated for all possible pairs of numbers and used to generate a table such as the one shown in Fig. 3b. The two-digit values in the table result from the situation when the vectors representing the address add up to go outside the current layer.

Table 1
An algorithm describing how to find neighbours of a point (the ' + ' operation signifies Balanced Ternary addition)

```
neighbours = [ ]
for i = 0 to pow(7,layers) − 1
    neighbours[i] = current point + i
```

The addition procedure is now illustrated with an example:

$$\begin{array}{r} 2 \\ 4\,2 \\ +\,4\,3 \\ \hline 3\,1\,6 \end{array}$$

To add 43 and 42, we first add the 2 and 3 to obtain '26'. The '6' is written down and the '2' is carried. The '2' is then added to the '4' producing '3'. This '3' is then added to the other '4' producing '31', which is also written down. The result is thus '316'. In the implementation, Balanced Ternary addition is achieved using a lookup table. In general, to find the N_1 neighbourhood of an index, x, requires the Balanced Ternary addition of 1 through 6 to be added to x. Extending this to one further layer gives the N_2 neighbourhood which requires the Balanced Ternary addition of 15, 14, 26, 25, 31, 36, 42, 41, 53, 52, 64, 63, and also the same additions as in N_1. The sequences of additions to find the neighbours in both N_2 and $N_{g(2)}$ are easy to find. The actual generation of neighbourhoods can be performed by use of a lookup table or by adding the coefficients at run time. Symmetry can be exploited to aid in this process. For the point 42, Fig. 4 shows N_1, N_2, and $N_{g(2)}$. An algorithmic description of the process is illustrated in Table 1.

Neighbourhood operations are often used in processing using masks. Here, a given pixel is replaced by a weighted sum of its neighbours and itself. As the neighbours for a given point have been calculated, this operation is easy to implement. Neighbourhoods of type $N_{g(i)}$ give a circular mask, something that is difficult to implement in square-images. In square-images, four nested loops are required to perform a complete mask operation on an image but in a hexagonal-image only two nested loops are required. This is a direct result of

Table 2
Reducing the image by a factor of seven

```
level 1 = [ ]
for i = 0 to pow(7,layers) − 1 step 7
  sum = 0
  for j = 0 to 6
    sum = sum + average[j] × image[i + j]
  level 1[i/7] = sum
```

the structure being a convolution ring [11]. A convolution by a mask, M, of size $M \times N$ performed upon a square-image is:

$$I(x_1,x_2)M(x_1,x_2) = \sum_{k_1=\frac{-M}{2}}^{\frac{M}{2}-1} \sum_{k_2=\frac{-N}{2}}^{\frac{N}{2}-1} I(x_1,x_2)M(x_1 - k_1, x_2 - k_2)$$

This is carried out for all values of (x_1, x_2) in the image. The hexagonal-image equivalent is for a mask, M, of size 7^m is:

$$I(x_1)M(x_1) = \sum_{k_1=0}^{7^m-1} I(x_1)M(x_1 - k_1).$$

Again this is carried out for all x_1 in the original image. The ' − ' in the summation is carried out using Balanced Ternary subtraction. As all points in the hexagonal-image are less than 7^L (L is the number of layers) the image boundaries are also trivial to compute. For instance, for an L level image:

$$b(x) = \begin{cases} 1, & (x \cup x+1 \cup \cdots \cup x+6) < 7^L \\ 0, & \text{otherwise} \end{cases}$$

Again the ' + ' is performed using Balanced Ternary addition. The function $b(x)$ will return a 1 when the point is on the images boundary.

Our definition of neighbourhoods in the hexagonal-image also permits easy generation of pyramid structures, which can be useful for multi-resolution edge detection, for a given image. This can be done by starting at the origin and averaging all the pixels in the $N_{g(i)}$ neighbourhoods. Successive layers can be found by increasing i from 0 to L. A simple example of an algorithm to reduce the images order by 7 is shown in Table 2. The average function can be chosen to perform different sorts of averaging. For example it could be defined to be a Gaussian to generate a Gaussian pyramid.

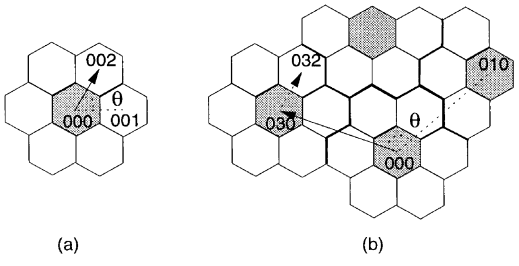


Fig. 5. Conversion from an 32_7 to an (x,y) coordinate.

Table 3
Rotation corresponding to each digit in an index

d_i	Rotation
1	0
2	$\frac{5\pi}{3}$
3	$\frac{4\pi}{3}$
4	π
5	$\frac{2\pi}{3}$
6	$\frac{\pi}{3}$

3.4. Display of hexagonal-images

Ideally, a display with a hexagonal matrix is needed for displaying hexagonal-images. However, conventional display devices use a square matrix, rendering simulation of the hexagonal display a necessity. The simulation involves a two-step process. First, a coordinate conversion is needed to change the index in the hexagonal-image to Cartesian coordinates. Next, hexagonal pixels have to be simulated by a suitable aggregation of square pixels.

The coordinate conversion process is relatively straightforward. It is best illustrated through the use of an example. Fig. 5 shows the conversion of 32_7 to a pair of Cartesian coordinates. A number in the Balanced Ternary system can be written as $d_n d_{n-1} \ldots d_0$. Examination of successive digits (starting from d_0) show an increase in radius. This increase can be seen by examining the sequence $\{1_7, 10_7, 100_7, \ldots\}$. The Cartesian coordinates expressed as a vector for each of these points can be computed and are illustrated below:

$$1_7 \rightarrow R\begin{bmatrix} 1 \\ 0 \end{bmatrix}, 10_7 \rightarrow R\begin{bmatrix} 2 \\ \sqrt{3} \end{bmatrix}, 100_7 \rightarrow R\begin{bmatrix} 1 \\ 4\sqrt{3} \end{bmatrix}$$

R corresponds to the inter-pixel spacing. The specific value of each digit corresponds to a rotation of this vector about the origin. The angle of rotation corresponding to each digit is illustrated in Table 3. A matrix can be formed for these rotations using the standard rotation matrix for Cartesian space. In the case of the digit being equal to zero, this digit introduces no offset.

The procedure is now illustrated by a simple example. For the index 32_7, the 2 after suitable rotation and scaling generates the following coordinates $(1/2, \sqrt{3}/2)$, and the 3 generates $(-5/2, \sqrt{3}/2)$. These offsets are then added together to generate the final Cartesian coordinates, $(-2, \sqrt{3})$.

Once the image has been sampled, each point can then be displayed as a hexagonally shaped pixel on the screen. These pixels are chosen to be both approximately hexagonal

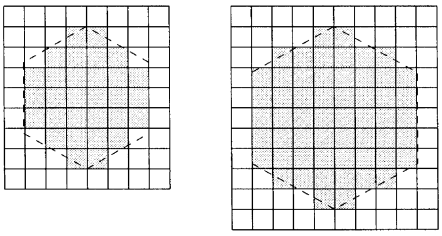


Fig. 6. Two possible choices for the hexagonal hyper-pixel.

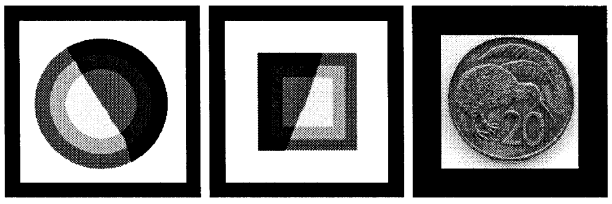
and to tile the plane exactly. A hexagonal pixel generated in this manner is known as a hyper-pixel. The hyper-pixel has been shown to be more pleasing to the eye as it takes into account the oblique effect in human vision [6]. Some possible choices are shown in Fig. 6.

Both cases shown in this figure approximate a hexagonal tile well. For the work presented here, the larger one has been chosen as it approximates the geometry of the hexagon better. A natural consequence of the hyper-pixel approach is that a somewhat lesser screen resolution is achieved with the image.

It can be shown that the screen has an aspect ratio such that the pixels are elongated in the horizontal direction. Thus, the major axis of the hexagonal hyper-pixel should be aligned in the vertical direction. The perimeter of the resulting image is roughly hexagonal in shape. This should not be considered unusual as the standard practice in conventional displays is to routinely clip images to be square in shape.

4. Edge detection

To study the effect of using a hexagonal sampling regime for edge detection the HIP framework was employed. The basic assumption used in most edge detection techniques is that the edges are characterised by large (step) changes in intensity. Hence, at the location of an edge, the first derivative of the intensity function should be a maximum or the second derivative should have a zero-crossing. In real images however, edges are also often marked by subtle changes in intensity which are noted and addressed by more advanced edge detection techniques. The three techniques we have chosen to study are Prewitt, Laplacian of Gaussian and the Canny edge detector. The Prewitt edge operators are first derivative operators while the Laplacian of Gaussian (LoG) is a second derivative operator. In terms of directional sensitivity, the Prewitt operators essentially are maximally sensitive to edges in horizontal and vertical directions while the LoG operator is isotropic. The isotropic nature of the LoG operator is due to the Gaussian smoothing function employed to reduce the noise sensitivity of the second derivative operation. The Canny edge detector is a good example of a near optimal edge detector combining features of the Prewitt and LoG operators.



(a) (b) (c)

Fig. 7. The input images: (a) T1 (b) T2 (c) T3.

Edge detection was performed on three test images in our study. Two of these (T1 and T2) were synthetic while the third (T3) was a real image of a New Zealand coin. All images, of size 256×256 px^2 (see Fig. 7a–c), were chosen as they contain a mixture of curves and lines along with a variation in contrast. All three techniques that were used in our study perform two distinct steps: edge magnitude calculation and thresholding. To compare the results of processing square-images versus hexagonal-images, the threshold was tuned to produce the best qualitative results and then the ratio of edge pixels to the image size was computed.

4.1. Prewitt edge operator

The Prewitt edge detector [18] is a gradient based edge detector. The detector is considered to be poor due to its bad approximation to the gradient operator. Also in square-images, the Prewitt edge mask weights all points the same despite the corner points being further away from the centre. However, the ease of implementation and low computational cost overcome these disadvantages.

Implementation of the edge operator involved computing an equivalent mask for the hexagonal case. This takes advantage of the fact that the vertical direction gradient mask can be approximated by a combination of two masks oriented at 60 and 120° to the horizontal. The horizontal mask for the hexagonal case is equivalent to the square case. The masks are illustrated in Fig. 8. Given that $h_1 = h_2 - h_3$ then only two of the masks are needed in the computation of the gradient images and the magnitude operator needs to be adjusted appropriately.

$$s_1 = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad s_2 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

(a)

$$h_1 = \begin{bmatrix} & 1 & 1 & \\ 0 & & & 0 \\ & -1 & -1 & \end{bmatrix} \quad h_2 = \begin{bmatrix} & 0 & 1 & \\ -1 & & & 1 \\ & -1 & 0 & \end{bmatrix} \quad h_3 = \begin{bmatrix} & -1 & 0 & \\ -1 & & & 1 \\ & 0 & 1 & \end{bmatrix}$$

(b)

Fig. 8. The different masks used in the Prewitt edge detector implementation.

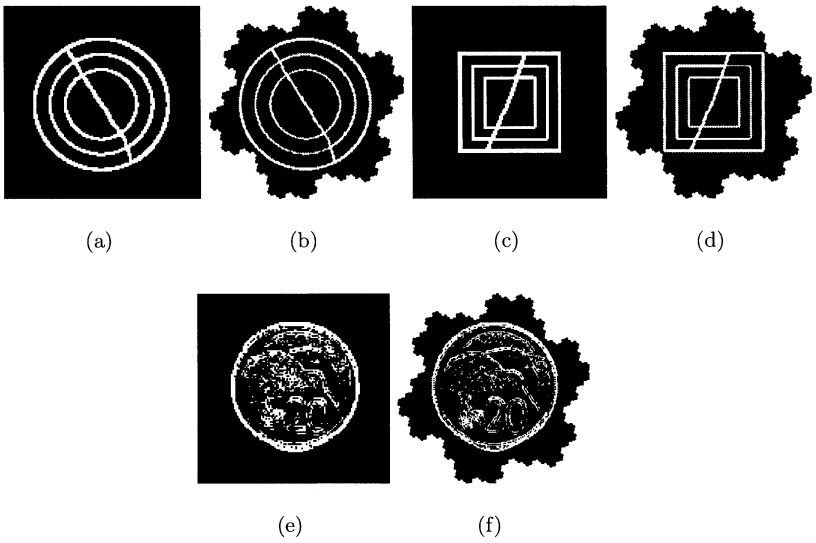


Fig. 9. Results for the Prewitt edge detector: (a) square-image for T1 (b) hexagonal-image for T1 (c) square-image for T2 (d) hexagonal-image for T2 (e) square-image for T3 (f) hexagonal-image for T3.

The results for the Prewitt edge detector are shown in Fig. 9. The results of processing the synthetic images illustrate the superiority of representation of circles in a hexagonal-image by way of the smoothness of the circles in the edge detected image. Furthermore, the diagonal dividing lines are also smoother in the hexagonal-image. Examination of the edge detected image T3, shows that (i) the shape of the coin is much more circular in the hexagonal-image, and (ii) the kiwi, the fern, and the number '20' all appear more clearly in the hexagonal-image. Overall, the appearance of the hexagonal-image is less noisy. This is due to the edges in the square-image being thicker. The ratio of edge pixels to image size for T1 is 11.5% for the hexagonal-image and 11.3% for the square-image. For T2, the ratios are 11.2% for the hexagonal-image and 9.5% for the square-image. In T3 the ratio of edge pixels in the two cases are 13.3% for the hexagonal-image and 11.9% for the square-image.

4.2. Laplacian of Gaussian edge operator

The LoG detector was first proposed by Marr [12]. The detection regime is as mentioned earlier, isotropic and consequently should perform well on hexagonal-images. The masks used in our implementation to approximate the LoG function are made up of 49 px.

The results of edge detection with the LoG operator, using the same test images, are shown in Fig. 10. For the hexagonal-images, the curves are shown with good clarity. In T3, the leaves of the fern are also revealed. The poor performance of edge detection in the square-image is due

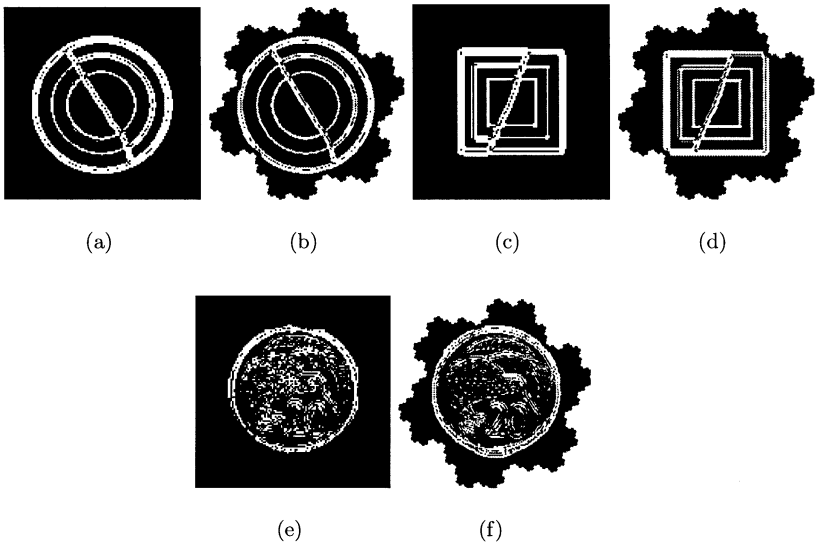


Fig. 10. Results for the Laplacian of Gaussian edge detector with a 7×7 mask: (a) square-image for T1 (b) hexagonal-image for T1 (c) square-image for T2 (d) hexagonal-image for T2 (e) square-image for T3 (f) hexagonal-image for T3.

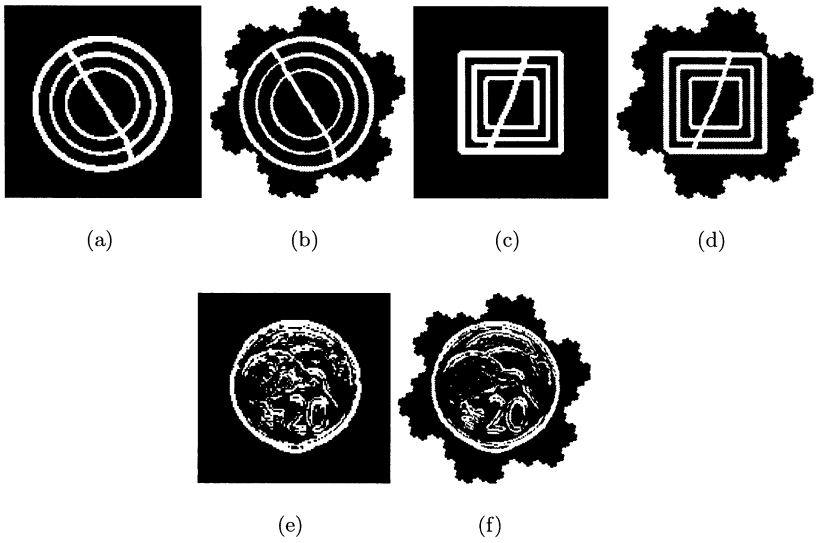


Fig. 11. Results for the Canny edge detector: (a) square-image for T1 (b) hexagonal-image for T1 (c) square-image for T2 (d) hexagonal-image for T2 (e) square-image for T3 (f) hexagonal-image for T3.

to the fact that the mask is a poor fit for the LoG function. To improve the performance of the square-image would require a bigger mask which would also result in an increased computational cost. Referring to T1, the ratio of edge pixels for the hexagonal-image is 19.4 and 14.3% for the square-image. For T2, the ratios are 18.3% for the hexagonal-image and 13.1% for the square-image. T3 gives ratios of 18.0 and 20.1% for the hexagonal- and square-images, respectively.

4.3. Canny edge operator

The Canny edge detector [18] combines two Gaussians to produce the gradient estimates in the horizontal and vertical directions. For the experiment, the masks used are of identical size to those described in Section 4.2.

The results are illustrated in Fig. 11. As Expected, this edge detector shows improved performance over the previous two cases. This is especially dramatic in the case of the square-image. In all cases of test images, the edges appear less noisy due to the maximal suppression step. The similarity in performance in the three images is due to the design of the Canny edge detector. However, it must be noted that the hexagonal-image achieves its results with fewer computations. For T1, the ratio of edge pixels for the hexagonal-image are 18.7 and 15.1% for the square-image. T2 has ratios of 19.9% for the hexagonal-image and 15.3% for the square-image. The ratios for T3 are 18.2% for the hexagonal-image and 14.2% for the square-image.

4.4. Effect of noise

Edge detection under noisy conditions is important for many applications. Hence, the performance of the Prewitt Edge detector was examined. The LoG and Canny detectors were not examined as they have in-built smoothing functions which would bias the results. Gaussian noise of standard deviation 20% was added to the original images before processing as illustrated in Fig. 12a and d.

The results are shown in Fig. 12. The results indicate a degradation in performance over the non-noisy case. However, in relative comparison, the result with the hexagonal-image is superior to the square-image as indicated by fewer parts of the image being incorrectly classified as edges. The level of noise retained in the interior of the circle in the T1 square-image is in marked contrast to the hexagonal-image. For T3, the outline of the kiwi and the number '20' have much clarity in the hexagonal-image as well. In the case of T1, the ratio of edge pixels are 18.9% for the

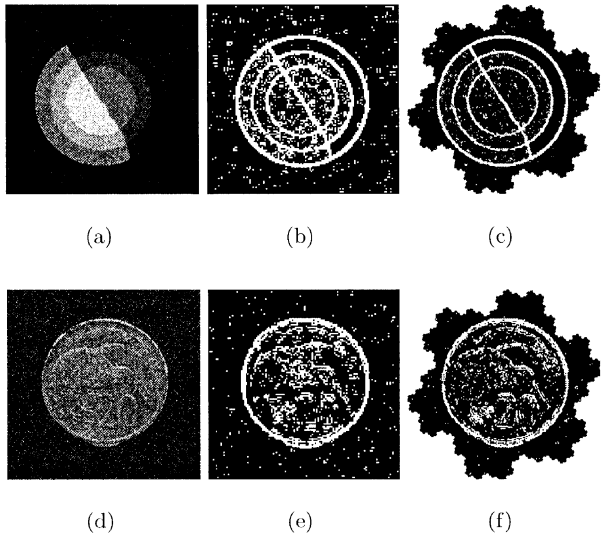


Fig. 12. Results for Prewitt edge detector with noise: (a) T1 with noise (b) square-image for T1 (c) hexagonal-image for T1 (d) T3 with noise (e) square-image for T3 (f) hexagonal-image for T3.

Table 4
Main features of square- and hexagonal-images

Feature	Hexagonal	Square
Indexing	x	(x,y)
Origin	Image centre	Top left
Neighbours	Hexagonal	Square
Image boundary	Hexagonal	Rectangular

hexagonal-image and 15.4% for the square-image. In T3, the ratio of edge pixels for the hexagonal-image are 13.9 and 12.9% for the square-image.

4.5. Summary of results

There are a number of conclusions that can be drawn from the above study. In all cases, the number of edge pixels were roughly the same but the edge detected hexagonal-image appears to be qualitatively better. This stems from the consistent connectivity of the pixels in hexagonal-images which aids edge detection. Pixels without consistent connectivity show up as discontinuities in the contour (in the input image) and breaks in the edge image. This is especially a problem when it comes to finding the edges of curved objects in square-images. Another issue, which has a direct bearing on the computational efficiency, is the mask size. For the Prewitt operator the hexagonal-image achieves similar performance with 13% fewer points in the mask. For the LoG operator, the square mask needs to be much bigger to produce a performance similar to the hexagonal mask. Under noisy conditions, the Prewitt operator is more robust when using hexagonal-images which is a definite advantage. A plausible explanation for this effect is that the masks used to implement the Prewitt operators align along different axes in the square- and hexagonal-images. This serves to partially reject noise in the vertical direction.

5. Concluding remarks

Our objective was to see if using an alternative, namely hexagonal, sampling scheme would improve the performance of the edge detection task. This has been one of the motivations for developing a practical hexagonal-image processing system. The HIP framework we have presented has some distinguishing features compared to the square-image processing framework. These are presented in Table 4. The first three rows indicate advantages of the HIP framework. A single index addressing scheme leads to efficient storage, while the placement of the image origin at the centre simplifies geometric transformations of a given image. Finally, the hexagonal-image allows non-traditional neighbourhoods with consistent boundary connectivity, which is useful for many computer vision applications.

Results of edge detection on hexagonal-images show that an edge map with better fidelity for curved objects is obtained than with square-images. In the case of straight edged objects the edge-maps are of similar quality. The hexagonal-images also reveals fine structures. For noisy images, the performance of edge detection improved with hexagonal-images. Furthermore, using the HIP framework for edge detection has computational advantages due to the use of the novel single index addressing scheme. In particular, convolution operations which are routinely used in edge detection can be implemented with great efficiency as indicated by Table 5.

Based on our work in edge detection, we make the following general observations. Orthogonal masks such as Sobel and Prewitt for square-images essentially compute gradients along the axes of symmetry of a square. Hence, in the development of gradient-based edge detectors for hexagonal-images, it is best to design masks that compute gradients along the three axes of symmetry of a hexagon. Secondly, isotropic edge detectors should perform much better on hexagonal-images since the circular mask functions used in these detectors have a best fit in a hexagonal and not a square lattice. Finally, further improvements in performance can be achieved with optimal edge detectors such as the Canny edge detector if they are redesigned for hexagonal-images. The hexagonal lattice gives rise to a variety of new neighbourhoods which can also be exploited in the design of new detection operators. Furthermore, these neighbourhoods can be used to provide better fit for existing ideal masking functions.

In conclusion, it should be noted that using hexagonal-images rather than square-images for edge detection has several advantages as illustrated in this paper. These are due mainly to the connectivity of the individual hexagonal

Table 5
Comparison of computational requirements for processing square- and hexagonal- images

Process	Hexagonal		Square	
	Additions	Multiplications	Additions	Multiplications
Convolution using N_1[a] mask	48	49	80	81
Finding N_1[a] neighbours	6[b]	–	8	–
Re-sampling	$MN7^L$	$MN7^L$	–	–
Display	–	2×7^l	–	–

[a] For square images a 3×3 neighbourhood approximates N_1.
[b] Using Balanced Ternary addition.

pixels generating more consistent contours. The HIP framework offers an added advantage in using hexagonal-images for edge detection, namely, great computational savings. The two together make a strong case for hexagonal based edge detection and seem to reinforce the point that hexagonal-image processing can be a viable alternative to square-image processing.

References

[1] A.A. Ahumada Jr., A. Poirson, Cone sampling array models, Journal of the Optical Society of America A 4 (8) (1987) 1493–1502.

[2] P.J. Burt, Tree and pyramid structures for coding hexagonally sampled binary images, Computer Graphics and Images Processing 14 (1980) 271–280.

[3] L. Gibson, D. Lucas, Spatial data processing using generalized balanced ternary, Proceedings of PRIP 82, IEEE Computer Society Conference on Pattern Recognition and Image Processing, 1982.

[4] L. Gibson, D. Lucas, Vectorization of raster images using hierarchical methods, Computer Graphics and Image Processing 20 (1982) 82–89.

[5] M.J.E. Golay, Hexagonal Parallel Pattern Transforms, IEEE Transactions on Computers C-18 (8) (1969) 733–740.

[6] R.M. Gray, P.C. Cosman, K.L. Oehler, Digital Images and Human Vision, MIT Press, Cambridge, MA, 1993 Chapter 4, pp. 35–52.

[7] N.P. Hartman, S.L. Tanimoto, A hexagonal pyramid data structure for image processing, IEEE Transactions on Systems, Man, and Cybernetics SMC-14 (2) (1984) 247–256.

[8] I. Her, Geometric transforms on the hexagonal grid, IEEE Transactions on Image Processing 4 (9) (1995) 1213–1222.

[9] I. Her, C.-T. Yuan, Resampling on a pseudohexagonal grid, CVGIP: Graphical Models and Image Processing 56 (4) (1994) 336–347.

[10] D.E. Knuth, The Art of Computer Programming: Seminumerical Algorithms, vol. 2, Addison-Wesley, Reading, MA, 1969.

[11] D. Lucas, A multiplication in *N*-space, IEEE Transactions on Image Processing Proceedings of the American Mathematical Society 74 (1) (1979) 1–8.

[12] D. Marr, Vision, Freeman, San Francisco, CA, 1982.

[13] R.M. Mersereau, The processing of hexagonally sampled two-dimensional signals, Proceedings of the IEEE 67 (6) (1979) 930–949.

[14] L. Middleton, J. Sivaswamy, Edge detection in a hexagonal-image processing framework, Image and Vision Computing New Zealand 1999, D. Pairman, H. North (Eds.), Landcare Research (1999) 217–222.

[15] L. Middleton, J. Sivaswamy, G. Coghill, Saccadic exploration using a hexagonal retina, Proceedings of ISA 2000, International ICSC Congress on Intelligent Systems and Applications, 2000.

[16] L. Middleton, J. Sivaswamy, G. Coghill, Shape extraction in a hexagonal-image processing framework, Proceedings of the 6th International Conference on Control, Automation, Robotics and Vision, ICARV 2000, 2000.

[17] I. Overington, Computer Vision: a Unified, Biologically-inspired Approach, Elsevier, Amsterdam, 1992.

[18] J.R. Parker, Algorithms for Image Processing and Computer Vision, Wiley, Canada, 1996.

[19] D.P. Peterson, D. Middleton, Sampling and reconstruction of wavenumber-limited functions in *N*-dimensional Euclidean spaces, Information and Control 5 (1962) 279–323.

[20] A. Rosenfeld, J.L. Pfaltz, Distance functions on digital pictures, Pattern Recognition 1 (1968) 33–61.

[21] J. Serra, Introduction to mathematical morphology, Computer Vision Graphics, and Image Processing 35 (1986) 283–305.

[22] R.C. Staunton, Hexagonal image sampling: a practical proposition, Proceedings of SPIE 1008 (1989) 23–27.

[23] R.C. Staunton, The design of hexagonal sampling structures for image digitisation and their use with local operators, Image and Vision Computing 7 (3) (1989) 162–166.

[24] R.C. Staunton, N. Storey, A comparison between square and hexagonal sampling methods for pipeline image processing, Proceedings of SPIE 1194 (1989) 142–151.

[25] J.W. van Roessel, Conversion of Cartesian coordinates from and to generalised balanced ternary addresses, Photogrammetric Engineering and Remote Sensing 54 (11) (1988) 1565–1570.

[26] D. Whitehouse, M. Phillips, Sampling in a two-dimensional plane. , Journal of Physics A: Mathematical and General 18 (1985) 2465–2477.

[27] G. Wolberg, Digital Image Warping, IEEE Computer Society Press, Silver Spring, MD, 1990.

[28] C.A. Wüthrich, P. Stucki, An algorithmic comparison between square- and hexagonal-based grids, CVGIP: Graphical Models and Image Processing 53 (4) (1991) 324–339.