Journal of Electronic Imaging

SPIEDigitalLibrary.org/jei

Storage and addressing scheme for practical hexagonal image processing

Xiangguo Li

SPIE

IS&T imaging.org

Storage and addressing scheme for practical hexagonal image processing

Xiangguo Li
Henan University of Technology, College of Information Science and Engineering, Zhengzhou Hi-tech Development Zone, Lianhua Street, Zhengzhou, Henan 450001, China
E-mail: xiangguoli@gmail.com

Abstract. *We present a practical hexagonal storage and addressing scheme, which eliminates the difference between theory and implementation in other addressing methods. This scheme employs a middleware-based address-mapping module that separates the algorithm and specific data addressing; thus any hexagonal algorithm can keep the native and consistent forms as in the coordinate system through theory and implementation. The scheme simplifies the implementation work and preserves all excellent features of the hexagonal lattice. Finally, we discuss the implementation issues and show that it's feasible and can be implemented efficiently.* © *2013 SPIE and IS&T [DOI: 10.1117/1.JEI.22.1.010502]*

1 Introduction

Almost 50 years ago, Petersen and Middleton[1] studied the general sampling problem in two and higher dimensions, and they showed that hexagonal lattice was optimal for circularly band-limited images. Besides 13.4% fewer samples than the rectangular counterpart, Mersereau[2] further demonstrated that hexagonal processing was superior in filter performance and computational efficiency. Furthermore, hexagonal lattice has better geometric properties, such as equidistant neighbors and uniform connectivity,[3] and it can also be widely found in the structure of biological visual sensors, such as compound eyes of insects and retina of human eyes; therefore, hexagonal processing is attractive in computer vision.

Generally, the coordinate systems for hexagonal lattice are not orthogonal, and this causes inconveniences for practical implementations. For instance, given a rectangular shape image, if it is sampled with rectangular lattice, as in the real world, the sampled data can be mapped into an array in memory naturally and can be addressed the same as in the Cartesian coordinates; if the image is sampled with hexagonal lattice, as shown in Fig. 1(a), things become different. Because the coordinates are skewed, if we want to address data as in the skewed coordinates, we may map the parallelogram region into an array;[4,5] obviously it is not efficient in memory usage.

Intuitively, we need to store the image data only,[4,5] i.e., map the rectangular region into an array as shown in Fig. 1(b). Meanwhile, new trouble arises. Consider two pixels located on an odd row and even row, respectively, together with their own six neighbors, it's clear that the two area shapes are quite different in the storage array, which means two sets of filters are needed for the odd rows and the even rows separately.[5,6] Recently, Rummelt et al.[7] proposed a scheme called array set addressing (ASA), in which two arrays were used to store odd rows and even rows separately. Three coordinates were used in the system. Rummelt also developed fundamental operation rules for the ASA system, and its significant feature is that it provides uniform addressing and operations for all data. In addition, there are other addressing methods for hexagonal shape images,[8–10] and they are not the concern of this paper.

Except for the parallelogram region mapping, we observe that current implementations rely on the specific data storage and addressing closely, which results in expression difference from the original forms in the skewed coordinate system, and this also introduces a gap between theory and implementation. Motivated by the description of row-based storage in Ref. 5, we adopt the middleware concept, and use an address-mapping module to separate the algorithm and data addressing. In this scheme, any algorithm uses the coordinates in the skewed hexagonal system, and accesses data from the storage array through the module; therefore, any algorithm can keep the consistent forms through theory and implementation. We also discuss the implementation issues, and show it's simple and feasible for practical use.

2 Proposed Scheme

2.1 *Fundamentals*

We start with a regular hexagonal sampling lattice as shown in Fig. 2, in which four vectors v_{h1}, v_{r1}, v_{h2}, and v_{r2} are defined. Then we can use v_{h1} and v_{h2} to define a sampling matrix:[11]

$$V_h = [v_{h1}|v_{h2}] = \begin{bmatrix} 1 & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} \end{bmatrix}. \tag{1}$$

Obviously, any sampled pixel can be located by a pair of integer values, and we can set up a hexagonal coordinate system (u, v), which is natural and complete for the hexagonal processing.

We can define another sampling matrix using v_{r1} and v_{r2}:

$$V_r = [v_{r1}|v_{r2}] = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{\sqrt{3}}{2} \end{bmatrix}. \tag{2}$$

Because v_{r1} and v_{r2} are orthogonal, we can set up a Cartesian coordinate system (x, y). In this case, the coordinates of any sampled pixel are integers but not arbitrary; thus it is not suitable for hexagonal processing.

Given a sampled pixel in Fig. 2, and let its coordinates be (u_i, v_i) in (u, v), and (x_i, y_i) in (x, y), its position can be given as $V_h[\,u_i \quad v_i\,]'$ and $V_r[\,x_i \quad y_i\,]'$. Since the same pixel corresponds with the same position, the following equality holds:

Paper 12482L received Nov. 19, 2012; revised manuscript received Jan. 14, 2013; accepted for publication Jan. 16, 2013; published online Jan. 31, 2013.

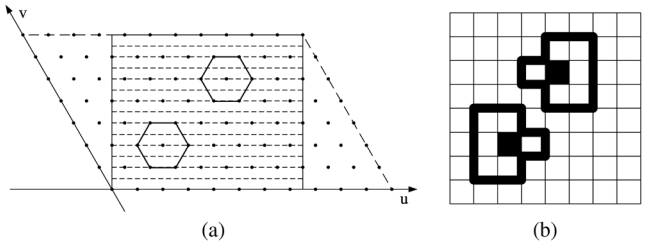


Fig. 1 Illustration of one hexagonally sampled rectangular shape image: (a) pixels in the hexagonal coordinate system, and (b) data in the storage array.

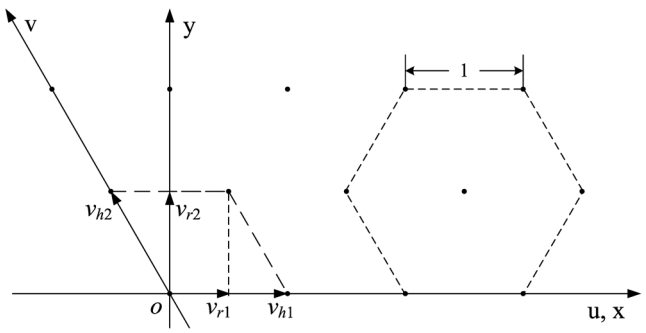


Fig. 2 Illustration of hexagonal lattice and the coordinate system.

$$V_h \begin{bmatrix} u_i \\ v_i \end{bmatrix} = V_r \begin{bmatrix} x_i \\ y_i \end{bmatrix}. \tag{3}$$

Because V_r is nonsingular, we can get

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = V_r^{-1} V_h \begin{bmatrix} u_i \\ v_i \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_i \\ v_i \end{bmatrix}. \tag{4}$$

Similarly, we have the following equation:

$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = V_h^{-1} V_r \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}. \tag{5}$$

That is, the coordinates in one system can be uniquely mapped into the other system and vice versa.

2.2 *Middleware-Based Address Mapping*

We aim to use both the natural hexagonal coordinates and the memory efficient array storage, and set up address mapping from the hexagonal coordinate system to the storage array, where the rectangular addressing serves as a bridge. Let (u_i, v_i), (x_i, y_i), and (r_i, c_i) be the addresses of one pixel in the hexagonal coordinate system, in the Cartesian coordinate system, and in the storage array, respectively; then, we express the mapping process as $(u_i, v_i) \Rightarrow (x_i, y_i) \Rightarrow (r_i, c_i)$. An example is shown in Fig. 3.

2.2.1 *Hexagonal to rectangular*

This conversion is based on Eq. (4), and can be rewritten as

$$\begin{cases} x_i = 2u_i - v_i \\ y_i = v_i \end{cases}. \tag{6}$$

2.2.2 *Rectangular to array*

It's natural to map a block of rectangular data into an array, while the data are staggered in this case. From the coordinates corresponding between Fig. 3(b) and 3(c), we can reach the relation:

$$\begin{cases} r_i = y_i \\ c_i = \lfloor x_i/2 \rfloor \end{cases} \tag{7}$$

with $\lfloor \cdot \rfloor$ the floor operator.

2.2.3 *Comprehensive mapping*

Combining Eqs. (6) and (7), we can get the final mapping formulae:

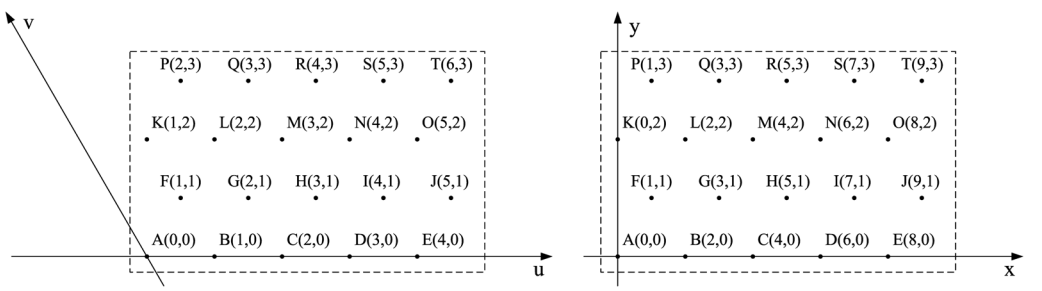
$$\begin{cases} r_i = v_i \\ c_i = \lfloor (2u_i - v_i)/2 \rfloor \end{cases}. \tag{8}$$

Based on the addressing relation, a middleware module can be designed to perform a data-accessing function.

3 Implementation Issues

3.1 *Address Mapping Computation*

Given a rectangular shape image is hexagonally sampled with nR rows and nC columns, and the data are stored in an array ImData[nR][nC]. Suppose a pixel (ui, vi) is stored



P(3,0)	Q(3,1)	R(3,2)	S(3,3)	T(3,4)
K(2,0)	L(2,1)	M(2,2)	N(2,3)	O(2,4)
F(1,0)	G(1,1)	H(1,2)	I(1,3)	J(1,4)
A(0,0)	B(0,1)	C(0,2)	D(0,3)	E(0,4)

(c)

Fig. 3 Example of address mapping: (a) in the skewed hexagonal coordinate system; (b) in the Cartesian coordinate system; and (c) in the storage array.

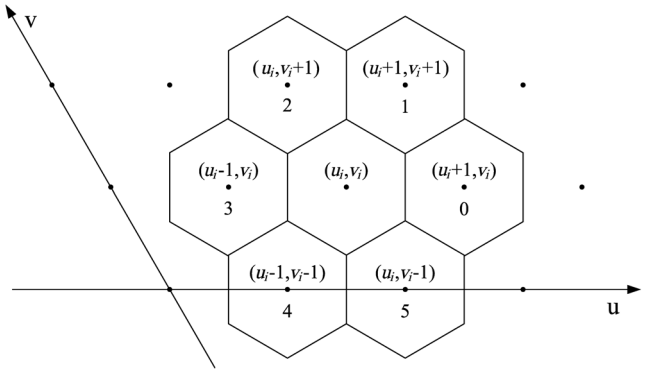


Fig. 4 Illustration of relative position of one pixel and its six neighbors.

at (ri, ci), we can access the pixel (ui, vi) by ImData$[vi]$ $[\lfloor(2 * ui - vi)/2\rfloor]$ from Eq. (8). However, it's not apparent which pixel is accessed in the expression. Instead, we choose to apply a macro substitution

$$\#\text{define FunImData}(ui, vi)$$
$$\text{ImData}[vi][\lfloor(2 * ui - vi)/2\rfloor]$$

to hide the address mapping and data accessing. The form (ui, vi) is much close to the array form $[ri][ci]$, and it's clear and readable.

Further, we can treat $\lfloor(2 * ui - vi)/2\rfloor$ as $(2 * ui - vi)/2$ followed by the floor operation $\lfloor\cdot\rfloor$. Since the multiplying and dividing of 2 can be efficiently executed by shift instructions, we prefer to rewrite $(2 * ui - vi)/2$ as $((ui << 1) - vi) >> 1$. Because the floor operation $\lfloor\cdot\rfloor$ is a byproduct of the integer operations, and the coordinates are indeed integer values, the floor operator $\lfloor\cdot\rfloor$ can be omitted. Therefore, we obtain the efficient expression:

$$\#\text{define FunImData}(ui, vi)$$
$$\text{ImData}[vi][((ui << 1) - vi) >> 1].$$

3.2 *Array Addressing Computation*

In the addressing of two-dimensional (2-D) array ImData$[ri]$ $[ci]$, i.e., $nC * ri + ci$, there is a multiplication instruction. Snyder et al.[4] proposed an alternative approach, and we adopt here. The 2-D array is serialized to a one-dimensional (1-D) array row-by-row: ImData$[nR * nC]$, and another 1-D array is also defined: FirstPtr$[nR]$, which is used to store the address of the start element of each row. Then, the macro is rewritten as

$$\#\text{define FunImData}(ui, vi)$$
$$\text{ImData}[\text{FirstPtr}[vi] + ((ui << 1) - vi) >> 1],$$

in which the multiplication is eliminated through the lookup table.

3.3 *Neighbor and Distance Computation*

Since the coordinates contain position information, we can perform geometry-related operations easily; for example, finding neighbors and computing distance.

As illustrated in Fig. 4, the relative positions of any pixel and its six neighbors are stable. We label the six neighbors as 0 to 5, and define two 1-D offset arrays:[4] $ou = \{1, 1, 0, -1 - 1, 0\}$, and $ov = \{0, 1, 1, 0, -1, -1\}$; then, the coordinates of the k'th neighbor can be given as $(ui + ou[k]$, $vi + ov[k])$.

For any two points $P_i(u_i, v_i)$ and $P_j(u_j, v_j)$, the Euclidean distance can be defined as

$$D(P_i, P_j) = \sqrt{(u_i - u_j)^2 + (v_i - v_j)^2 - (u_i - u_j)(v_i - v_j)}. \quad (9)$$

4 Conclusion

We present a middleware-based storage and addressing scheme for practical hexagonal image processing. The scheme uses the original coordinates in the hexagonal system, and two benefits are kept. One is that we can implement the algorithm naturally as in the theory; the other is that we can perform geometry-related operations easily. The scheme is feasible and efficient.

References

1. D. P. Petersen and D. Middleton, "Sampling and reconstruction of wave-number-limited functions in n-dimensional Euclidean spaces," *Inf. Control* **5**(4), 279–323 (1962).
2. R. M. Mersereau, "The processing of hexagonally sampled two-dimensional signals," *Proc. IEEE* **67**(6), 930–949 (1979).
3. L. Middleton and J. Sivaswamy, *Hexagonal Image Processing: A Practical Approach*, Springer, London (2005).
4. W. E. Snyder, H. Qi, and W. Sander, "Coordinate system for hexagonal pixels," *Proc. SPIE* **3661**, 716–727 (1999).
5. J. Rosenthal, "Filters and filterbanks for hexagonally sampled signals," Ph.D. Thesis, Georgia Institute of Technology (2001).
6. R. C. Staunton, "Hexagonal sampling in image processing," *Adv. Imag. Electron Phys.* **107**, 231–307 (1999).
7. N. I. Rummelt and J. N. Wilson, "Array set addressing: enabling technology for the efficient processing of hexagonally sampled imagery," *J. Electron. Imag.* **20**(2), 023012 (2011).
8. I. Her, "Geometric transformations on the hexagonal grid," *IEEE Trans. Image Process.* **4**(9), 1213–1222 (1995).
9. P. Sheridan, *Spiral Architecture for Machine Vision*, Ph.D. Thesis, Univ. of Technology Sydney (1996).
10. L. Middleton and J. Sivaswamy, "Framework for practical hexagonal-image processing," *J. Electron. Imag.* **11**(1), 104–114 (2002).
11. D. E. Dudgeon and R. M. Mersereau, *Multidimensional Digital Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey (1984).