Canny Edge Detection on a Virtual Hexagonal Image Structure

Xiangjian He[1,2,#], Jianmin Li[2,3], Daming Wei[1], Wenjing Jia [2] and Qiang Wu[2]

[1]Biomedical Information Technology Laboratory
University of Aizu
Japan

[2]Centre for Innovation of IT Services and Applications (iNEXT)
University of Technology, Sydney,
Australia

[3] College of Mathematics and Computer Science
Fuzhou University
China

[#]Contact email: sean@it.uts.edu.au

Abstract – **Canny edge detector is the most popular tool for edge detection and has many applications in the areas of image processing, multimedia and computer vision. The Canny algorithm optimizes the edge detection through noise filtering using an optimal function approximated by the first derivative of a Gaussian. It identifies the edge points by computing the gradients of light intensity function based on the fact that the edge points likely appear where the gradient magnitudes are large. Hexagonal structure is an image structure alternative to traditional square image structure. Because all the existing hardware for capturing image and for displaying image are produced based on square structure, an approach that uses linear interpolation is proposed in this paper for conversion between square and hexagonal structures. Gaussian filtering together with gradient computation is performed on the hexagonal structure. The pixel edge strengths on the square structure are then estimated before the thresholds of Canny algorithm are applied to determine the final edge map. The experiments show satisfactory edge detection results on hexagonal structure, compared with the results using Canny algorithm on square structure.**

Keywords — **Edge detection, image interpolation, hexagonal structure, Canny edges, Gaussian filtering**

I. INTRODUCTION

In the past decades, many algorithms have been developed for edge detection. Canny edge algorithm [1] is the most popular edge detector that formulates the task of edge detection as a numerical optimization problem.

The Canny edge detector uses a filter based on the first derivative of a Gaussian to suppress image noise. An edge in an image may point in a variety of directions, so the traditional Canny algorithm uses various operators to detect edges in the noise-filtered image in four directions: horizontal, vertical and two diagonal directions. The use of an edge detection operator returns a value of intensity gradient. The pixels at which the gradient magnitudes are large are more likely to be edges than the pixels with small gradient magnitudes. Because it is impossible in most of cased to specify a threshold of intensity gradient magnitudes for the identification of edges, Canny algorithm uses two thresholds, i.e., high and low thresholds. The Canny algorithm begins by applying the high threshold to mark out the highly likely edges. Then, other likely edges can be traced using the low threshold through the image along the directions estimated.

In the previous years, there have seen some papers presenting Canny edge algorithms on hexagonal image structure where an image is represented by a collection of hexagons of the same size. Hexagonal image structure has numerous advantages such as higher degree of circular symmetry, uniform connectivity, greater angular resolution, and a reduced need of storage and computation in image processing operations (see [2-4]).

The most recent work on Canny edge detection on hexagonal structure appeared in [5]. In [5], a scheme to simulate a hexagonal grid on a regular rectangular grid device was presented. Each of the original square pixels and simulated hexagonal pixels is regarded as a collection of

sub-pixels. The light intensities of all sub-pixels constituting a hexagonal pixel are computed using a bi-linear interpolation technique. On the other hand, the light intensities of all sub-pixels constituting a square pixel are computed using a tri-linear interpolation technique. The experimental results showed the improvement in edge detection in terms of accuracy and efficiency. In [5], image filtering and gradient computation were performed on the hexagonal structure. The identification of edges was carried out at the traditional square structure after the tri-linear interpolation process.

In this paper, we modify the work shown in [5]. We replace the operations of bi-linear and tri-linear interpolations with simple and efficient linear interpolation algorithms. The linear interpolation followed by a thresholding is performed to determine the corresponding edges on the square structure after the edge strengths and directions on the hexagonal structure are obtained.

The rest of this paper is organized as follows. In Section II, we perform the linear interpolation schemes. In Section III, the pure Canny edge algorithm on the hexagonal structure is presented. Experimental results are demonstrated in Section IV. We make the conclusion in Section V.

II. CONVERSION BETWEEN HEXAGONAL AND SQUARE STRUCTURES

Because there has been no hardware available for image display and capture on hexagonal structure, a software approach to the construction of virtual hexagonal structure as shown in [6] is used in this paper. To construct hexagonal pixels, each square pixel was first separated into 7×7 small pixels, called *sub-pixels*. Each virtual hexagonal pixel was formed by 56 sub-pixels as shown in Figure 1. Figure 1 shows a collection of seven hexagonal pixels.

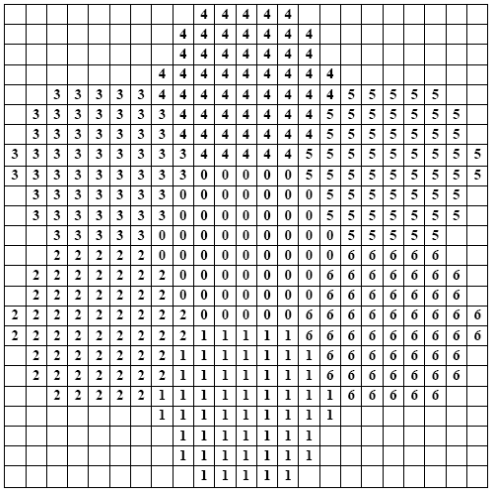

Fig. 1. A cluster of seven hexagonal pixels [6]

Without loss of generality, we assume that the number of rows and number of columns in the original image represented in the square structure are multiples of 8. Different from all previous papers, in this paper, we locate the hexagonal pixels in a new way. The 1st hexagonal pixel is located at exactly the same location of the square pixel at row 0 and column 0. In another word, we locate the central sub-pixel of this 1st hexagonal pixel at the central sub-pixel of the 1st square pixel located at row 0 (i.e., 1st row) and column 0 (i.e., 1st column). After this 1st hexagonal pixel is located, all other hexagonal pixels can then by located accordingly. It is easy to see that all hexagonal pixels are located in the columns that all square pixels are located in. If we use the idea of defining the rows and columns in square structure, we can define a row in the hexagonal structure as a set of hexagonal pixels that are sitting on exactly the same horizontal line, and a column as the set of pixels on the same vertical line. Then every column in the square structure is a column of the hexagonal structure because the distance between any two columns is 7 pixels wide in both structures. Furthermore, it is easy to see that pixels in the first row (i.e., row 0) and every 2nd row after this row are sitting in the columns of 1st, 3rd, 5th and so forth but include no pixels in the columns of 2nd, 4th, 6th and so forth. On the other hand, the pixels in the 2nd row and every 2nd row after this row are sitting in the columns of 2nd, 4th, 6th and so forth but include no pixels in the columns of 1st, 3rd, 5th and so forth. As an illustration, in Figure 2 below, the hexagonal pixel with notation P1 (one above the central pixel) is itself located in one row, pixels P2 and P5 are in the next row, followed by pixel P0 itself in one row, pixels P3 and P5 in one row and pixel P4 itself in another row. In the same figures, we see pixels P5 and P6 in one column, pixel P1, P0 and P4 in another column and pixels P2 and P3 in the last column.

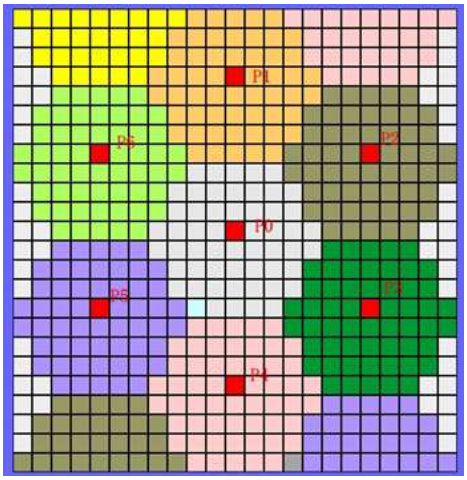

Fig. 2. Reference sub-pixels of virtual hexagonal pixels [5]

To define the size of the virtual hexagonal structure, let us assume that the size of the original square structure is

8M×8N. Then, it is easy to compute that the hexagonal structure has 14M rows and 8N columns (See Figure 3).

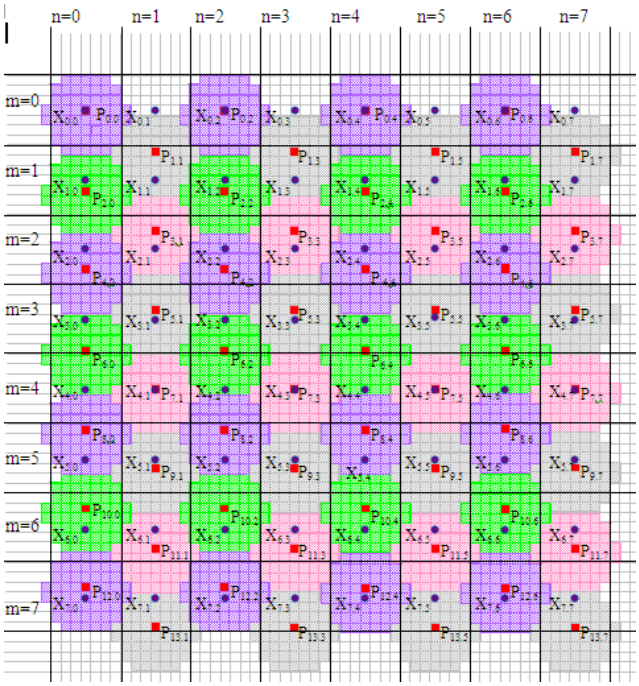

Fig. 3. This figure shows 8x8 square structure and corresponding 14x8 hexagonal structure

2.1. Conversion from Square Structure to Hexagonal Structure

As shown in [5], we recall the central sub-pixel of a given hexagonal pixel is defined to be the sub-pixel that is located at the fourth row and the middle column of the 56 sub-pixels forming the hexagonal pixel. This sub-pixel is also called a *reference sub-pixel* as represented by the sub-pixels P_i (i=0,1,2,…,6) in Figure 2.

For the reference sub-pixel (denoted by X) of a given hexagonal pixel, there exist two square pixels (i.e., the central sub-pixels of the square pixels in the sub-pixel space) denoted by A and B, lying on two consecutive rows and the same column of X, such that point X falls between A and B.

Let us denote the coordinates (the row and column in the sub-pixel space) of A, B and X by (A_x, A_y), (B_x, B_y) and (X_x, X_y) respectively. Let

$$\beta = \frac{|A_y - X_y|}{|A_y - B_y|}. \quad (1)$$

Then, it is easy to derive that

$$X = (1-\beta)A + \beta B. \quad (2)$$

Let f be the image brightness function that maps a pixel (either square pixel or hexagonal pixel) to its light intensity value. Then the intensity value assigned to X using a linear interpolation method is computed as

$$f(X) = (1-\beta)\cdot f(A) + \beta \cdot f(B). \quad (3)$$

The pseudo-code of the above-mentioned linear interpolation is written below.

```
For (int j=0; j<4*N; j++) // for columns
{
        j1=j*2; j2=j*2+1;
        For (int i=0; i<7*M, i++) // for rows
        {
                // interpolation at odd columns
                int Ax1=Bx1=Xx1=j1*7+3
                int Xy1=8*i+3;//central sub-pixel
                int Ay1=(int 8*i/7)*7+3;
                int By1=Ay1+7;
                beta1=(Xy1-Ay1)/7;
                f(Xx1,Xy1)=(1-beta1)*f(Ax1,Ay1)
                        +beta1*f(Bx1, By1);

                //interpolation at even columns
                int Ax2=Bx2=Xx2=j2*7+3
                int Xy2=8*i+4+3;
                int Ay2=(int (8*i+4)/7)*7+3;
                int By2=min(Ay2+7, 8*7*M-4);
                beta2=(Xy2-Ay2)/7;
                f(Xx2,Xy2)=(1-beta2)*f(Ax2,Ay2)
                        +beta2*f(Bx2, By2);
        }
}
```

From the above code, it is easy to see that in the even columns, the intensities of the hexagonal pixels at the last row are identical to the intensities of square pixels at the last row of the same columns.

2.2. Conversion from Hexagonal Structure to Square Structure

Like the conversion from square structure to hexagonal structure, we also perform a simple linear interpolation to convert from hexagonal structure to square structure. Note that for each square pixel (except the pixels in the first row of even columns and the pixels in the last row of odd columns), if we denote its central sub-pixel by *A*, then there exist two hexagonal pixels (i.e., the corresponding central or reference sub-pixels in the sub-pixel space) denoted by *X* and *Y*, lying on two consecutive rows and the same column of *A*, such that point *A* falls between *X* and *Y*.

Similar to the previous subsection, let us denote the coordinates (the row and column in the sub-pixel space) of A, X and Y by (A_x, A_y), (X_x, X_y) and (Y_x, Y_y) respectively. Let

$$\alpha = \frac{|A_y - X_y|}{|X_y - Y_y|}. \quad (4)$$

Then, it is easy to derive that

$$A = (1-\alpha)X + \alpha Y \quad (5)$$

and the intensity value assigned to *A* using a linear interpolation method is computed as

$$f(A) = (1-\alpha)\cdot f(X) + \alpha \cdot f(X). \quad (6)$$

The pseudo-code of the above mentioned linear interpolation is written below.

```
For (int j=0; j<4*N; j++) \\ for columns
{
        j1=j*2; j2=j*2+1;

        // for row 0, do the following;
        int Ax1=Xx1= j1*7+3; //for odd columns
        int Ay1=Xy1=3;
        f(Ax1,Ay1)= f(Xx1,Xy1);

        int Ax2=Yx2=j2*7+3; // for even columns
        int Ay2=3;
        int Yy2=7;
        f(Ax2,Ay2)= f(Yx2,Yy2);

        // for row 8M-1, do the following
        int Ax1=Xx1= j1*7+3; //for odd columns
        int Ay1=7*8*M-4;
        int Xy1=Ay1-1;
        f(Ax1,Ay1)= f(Xx1,Xy1);

        int Ax2=Yx2=j2*7+3; // for even columns
        int Ay2=7*8*M-4;
        int Yy2=Ay2+3;
        f(Ax2,Ay2)= f(Yx2,Yy2);

        // for rows other than rows 0 and 8M-1, do
        // following
        For (int i=1; i<8*M-1, i++)
        {
                // interpolation at odd columns
                int Ax1=Xx1=Yx1=j1*7+3 ;
                int Ay1=7*i+3;
                int Xy1=(int 7*i/8)*8+3;
                int Yy1=Xy1+8;
                alpha1=(Ay1-Xy1)/8;
                f(Ax1,Ay1)=(1-alpha1)*f(Xx1,Xy1)
                        +alpha1*f(Yx1, Yy1);

                //interpolation at even columns
                int Ax2=Xx2=Yx2=j2*7+3 ;
                int Ay2=7*i+3;
                int Xy2=(int 7*i/8)*8+7;
                int Yy2=Xy2+8;
                alpha2=(Ay2-Xy2)/8;
                f(Ax2,Ay2)=(1-alpha2)*f(Xx2,Xy2)
                        +alpha2*f(Yx2, Yy2);
        }
}
```

III. EDGE DETECTION

Similar to the work shown in [5], the edge detection approach goes through three steps: noise filtering using a Gaussian filter, edge detection using Sobel operator and edge refining using thresholds.

3.1. Noise Filtering

Before the edge map of an image is found, it is common that image noise is removed (or suppressed) by applying a filter that blurs or smoothes the image.

One commonly used filter is implemented by convolution of the original image function with a Gaussian kernel as defined in Equation (7) below. Let $f:\Re^2 \to \Re$ be the original brightness function of an image which maps the coordinates of a pixel to a value in light intensity. Let a_0 be the reference pixel. Then, for a given reference pixel a_0, its new intensity value, denoted by $h(a_0)$, is computed by

$$h(a_0) = k^{-1} \sum_{i=0}^{n-1} f(a_i) e^{-\frac{(a_i - a_0)^2}{2\sigma^2}}, \qquad (7)$$

where k is the normalization constant and is defined as

$$k = \sum_{i=0}^{n-1} e^{-\frac{(a_i - a_0)^2}{2\sigma^2}}, \qquad (8)$$

and a_i (i=0, 1, …n-1) are the n neighbouring pixels of a_0.

Considering about 99.5% of energy is found in the central area of "Mexico cap" (the curve of Gaussian function with parameter σ) within the radius of 3σ, in order to increase the computation speed, Equation (7) in this paper is computed only over a small area surrounding each reference pixel and covering the disk with centre at the reference pixel and radius of 3σ. In this paper, σ is set to be 1. Therefore, the convolution window is set to be a 49 pixel block on the virtual hexagonal structure for the weighted average computation using Equations (7) and (8), centred at the reference pixel. Hence, for Equations (7) and (8) above, n = 49. The computation to include the 49 hexagonal pixels for convolution using (7) and (8) on the hexagonal structure is achieved using the intensities of the reference sub-pixels of the hexagonal pixels.

3.2. Edge Detection

In order to implement edge detection on the virtual hexagonal structure, a Sobel operator as defined in [7] is applied in this paper. Using the Sobel operator, edge strength and direction at each hexagonal pixel (i.e., its reference sub-pixel) can be calculated.

3.3. Edge Refining

After the edge detection step shown in Subsection 3.2, all hexagonal pixels have been assigned the intensity values that show the edge hexagonal pixels and their strengths. An edge map on the original square structure can hence be obtained by simply computing the intensity value of every square pixel using the linear interpolation as shown in Subsection 2.2. This edge map shows the square edge pixels and their strengths. We can then follow the remaining steps of Canny's method as shown in [1,8] to obtain the final edge map by using one lower threshold and one higher threshold.

IV. EXPERIMENTAL RESULTS

To study the effect of new edge detection method on the virtual hexagonal structure, and compare with the results obtained in [5] based on complex bilinear and tri-linear image interpolations and the results on the square structure, 8-bit grey level Lena andn Mary images of size 256×256 are chosen as our sample image (see Figure 4).

Fig. 3. Original Lena image (left) and Mary image (right)

Three different edge maps are produced in order to demonstrate the performance in accuracy and efficiency improved by the proposed edge detection method. The first edge map is obtained on square structure. The second one is obtained using the bi-linear and tri-linear interpolation methods as shown in [5] on hexagonal structure. The third one is obtained based on the simple linear interpolation algorithms introduced in this paper. All edge maps are generated with σ=1 and n=49 for Gaussian filtering. The same lower and higher thresholds are used to locate the exact edge points for all edge maps. The higher threshold is 0.125 and the lower threshold is 0.05.

Figure 5(b) shows improved edge maps with clearer edges and less noise points compared with the maps in Figure 5(a). Overall, Figure 5(b) has fewer blurred edge segments than Figure 5(a). This can be seen from the edges at the hair areas. On the other hand, those significant edge points on Figure 5(a) can also be seen in Figure 5(b). This is because an area consisting of 49 hexagonal pixels is bigger than an area consisting of 49 square pixels, i.e., the area involved in convolution for Gaussian filtering on hexagonal structure is bigger than on square structure. Therefore, Gaussian convolution on hexagonal structure suppresses

image noise better while keeping important image information.

(a) Edge maps on square structure

(b) Edge maps based on bi-linear and tri-linear interpolations on hexagonal structure

(c) Edge maps based on simple linear interpolation on hexagonal structure

Fig 4. Edge detection results with Gaussian filtering

Moreover, the edge maps on Figure 5(c) contains more details and more curvature edges than those on Figure (b). Furthermore, the computation for Figure 5(c) is about 2 times faster than Figure 5(b) because it simplifies the computation for interpolation. Table 1 shows the time costs for Canny edge detection on hexagonal structure using method shown in [5] and the method proposed in this paper. Therefore, the edge detection based on the linear interpolation algorithms is more efficient than the work based on bi-linear and tri-linear interpolation algorithms, while the quality of edge results are not reduced.

V. CONCLUSIONS

In this paper, an edge detection method has been presented. The use of simple linear interpolation algorithms combined with the advantages of hexagonal image structure has achieved encouraging and promising edge detection performance. We have shown that the use of linear interpolation results in more efficient edge detection and hexagonal structure leads to more accurate and less noise edge maps compared with the state-of-the-art Canny edge detector.

Table 1. Time costs (in seconds) for Canny edge detection on hexagonal structure

Image	Bi-linear/tri-linear	Linear
Lena	0.411	0.200
Mary	0.391	0.211

REFERENCES

[1] J. F. Canny, *A computational approach to edge detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.8 1986, pp.679-698.

[2] R. Staunton, *The design of hexagonal sampling structures for image digitization and their use with local operators*, Image and Vision Computing, Vol.7, No.3, 1989, pp. 162-166.

[3] B.K.P. Horn, *Robot Vision*. MIT Press, New York, 1986.

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

[5] Xiangjian He, Wenjing Jia and Qiang Wu, *An Approach of Canny Edge Detection with Virtual Hexagonal Image Structure*, Proceedings of 10th International Conference on Control, Automation, Robotics and Vision (ICARCV2008, Tier A), pp.879-882, 2008.

[6] Xiangjian He, Jianmin Li and Tom Hintz, *Comparison of Image Conversions between Square Structure and Hexagonal Structure*, Lecture Notes in Computer Science (ACIVS07), J. Blanc-Talon et al. (Eds.): ACIVS 2007, LNCS 4678, pp. 262–273, 2007.

[7] Xiangjian He, Wenjing Jia, Namho Hur, Qiang Wu, Jinwoong Kim and Tom Hintz, *Bi-lateral Edge Detection on a Virtual Hexagonal Structure*, Lecture Notes in Computer Science (ISVC2006), LCNS, Springer, 2006, Vol.4292, pp.1092-1101.

[8] Qiang Wu, Xiangjian He and Tom Hintz, *Bilateral Filtering Based Edge Detection on Hexagonal Architecture*, Proc. 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing, Philadelphia, PA, USA, Volume II, 2005, pp.713-716.