Computation of the Euler Number of a Hexagonal Binary Image Using the Perimeter

Jack Weinstein

Department of Mathematics, University of Haifa, Israel

jack-weinstein@hotmail.com

***Abstract* – Two versions of a perimeter-based algorithm for the computation of the Euler number of a hexagonal binary image are devised. These new versions are compared experimentally to the original (definition based) algorithm. One of the new versions is shown to outperform the other two methods.**

***Keywords* — Euler number; hexagonal binary digital image; perimeter; semi-perimeter; algorithm performance**

I. INTRODUCTION

Since Golay [1], researchers have made various attempts to represent digital images by using a hexagonal structure. The hexagonal grid has important advantages: small quantization error, equidistant neighbours, consistent connectivity, high symmetry, good angular resolution. However, it also leads to some difficulties: a lack of practical hardware for capturing and displaying hexagonal images, and a lack of simple storage and addressing schemes. For details, see [2].

A hexagonal binary digital image can be defined as a subset of an array of 0/1 integer values. In more detail, it can be defined as a function $P: Z^2 \supset Hex \to \{0,1\}$, where Z^2 denotes the integer grid in the plane and $Hex = \{(i,j) \in Z^2 \mid i+j \text{ even}\}$. We choose a coordinate system in Z^2 such that the first axis points downward (the *row* axis) and the second axis points to the right (the *column* axis). As usual, an element $(i,j) \in Z^2$ can be regarded as a point (placed at row i and column j), or as a (usually regular) hexagon placed with its centre at array position (i,j); such an element is called a (hexagonal) *pixel*. If $P(i,j)=0$, the pixel (i,j) is called a *background* point; otherwise, if $P(i,j)=1$, the pixel (i,j) is called a *foreground* point. We assume the number of 1-pixels to be finite; we can therefore restrict each image to a digital rectangle. We also assume that the two bottom rows, as well as the leftmost and rightmost columns, completely belong to the background. We distinguish between array coordinates and the usual cartesian coordinates.

There are many ways to orient the pixels of the image, but only two of them are frequently used: (1) the H (horizontal) orientation, where each pixel has two horizontal sides, and (2) the V (vertical) orientation, where each pixel has two vertical sides. We always choose the pixels as having unit length sides.

In the first case (the H orientation), the pixels are placed at cartesian coordinates (x,y), where $x = \left(\sqrt{3}/2\right)\cdot i$, $y = \left(3/2\right)\cdot j$, and $i+j$ is even. Some pixels sit on even rows $i=2a$ and on even columns $j=2b$; their coordinates are $x = \left(\sqrt{3}\right)\cdot a$, $y = 3\cdot b$; the other pixels sit on odd rows $i=2a+1$ and odd columns $j=2b+1$; their coordinates are $x = \left(\sqrt{3}/2\right) + \left(\sqrt{3}\right)\cdot a$, $y = \left(3/2\right) + 3\cdot b$.

In the second case (the V orientation), the pixels are placed at cartesian coordinates (x,y), where $x = \left(3/2\right)\cdot i$, $y = \left(\sqrt{3}/2\right)\cdot j$, and $i+j$ is even. Some pixels sit on even rows $i=2a$ and on even columns

$j=2b$; their coordinates are $x=3\cdot a$, $y=\left(\sqrt{3}\right)\cdot b$; the other pixels sit on odd rows $i=2a+1$ and odd columns $j=2b+1$; their coordinates are $x=\left(3/2\right)+3\cdot a$, $y=\left(\sqrt{3}/2\right)+\left(\sqrt{3}\right)\cdot b$.

A more detailed description can be found in [3] and [4].

II. Basic Definitions

We start with the H orientation. Given a hexagonal binary digital image P, the set of all pixels and their adjacencies define a graph $Himgr(P)$, the H *image graph* of P:

its vertices are single pixels , represented as $[x]$;

its edges are vertical pairs , and tilted pairs  and , represented as $\begin{bmatrix}x\\x\end{bmatrix}$, $\begin{bmatrix}x & x\end{bmatrix}$, $\begin{bmatrix}x & \\ & x\end{bmatrix}$, respectively;

its faces are triples of pixels , , represented as $\begin{bmatrix}x & x\\ x & \end{bmatrix}$, $\begin{bmatrix}x & \\ x & x\end{bmatrix}$, respectively.

Each X denotes either of the values 0 or 1.

The subgraph $Hfggr(P)$ of $Himgr(P)$ that consists of all pixels of type $[1]$, all edges of type $\begin{bmatrix}1\\1\end{bmatrix}$, $\begin{bmatrix}1 & 1\end{bmatrix}$, $\begin{bmatrix}1 & \\ & 1\end{bmatrix}$, and all faces of type $\begin{bmatrix}1 & 1\\ 1 & \end{bmatrix}$, $\begin{bmatrix}1 & \\ 1 & 1\end{bmatrix}$, will be called the H *foreground graph* of P.

In the case of V orientation, the set of all pixels and their adjacencies define a graph $Vimgr(P)$, the V *image graph* of P:

its vertices are single pixels , represented as $[x]$;

its edges are horizontal pairs , and tilted pairs  and , represented as $[x\ \ x]$, $\begin{bmatrix}x & \\ & x\end{bmatrix}$, $\begin{bmatrix}x & \\ x & \end{bmatrix}$, respectively;

its faces are triples of pixels , , represented as $\begin{bmatrix}x & x & \\ x & & x\end{bmatrix}$, $\begin{bmatrix}x & & x\\ & x & \end{bmatrix}$, respectively.

Here also, each X denotes either of the values 0 or 1.

The subgraph $Vfggr(P)$ of $Vimgr(P)$ that consists of all pixels of type $[1]$, all edges of type $[1\ \ 1]$, $\begin{bmatrix}1 & \\ & 1\end{bmatrix}$, $\begin{bmatrix}1 & \\ 1 & \end{bmatrix}$, and all faces of type $\begin{bmatrix}1 & 1 & \\ & & 1\end{bmatrix}$, $\begin{bmatrix}1 & & \\ & 1 & 1\end{bmatrix}$, will be called the V *foreground graph* of P.

In the following we will denote by $\#\left(P^{(H)};\pi\right)$ the number of occurrences of the local pattern π in the graph $Himgr(P)$; we will denote by $\#\left(P^{(V)};\pi\right)$ the number of occurrences of the local pattern π in the graph $Vimgr(P)$.

The Euler number (Euler characteristic in mathematics) is a very important topological invariant in the study of binary images. Many methods have been devised for its computation. We only mention here the paper [8], were a method was given that also works on the hexagonal grid.

The H oriented hexagonal Euler number is defined as

$$\chi\left(P^{(H)}\right)=v(P^{(H)})-e_0(P^{(H)})-e_1(P^{(H)})-e_2(P^{(H)})+f_1(P^{(H)})+f_2(P^{(H)}) \qquad (1H)$$

where

$v\left(P^{(H)}\right)=\#\left(P^{(H)};[1]\right)$ (the number of vertices in $Hfggr(P)$)

$e_0(P^{(H)})=\#\left(P^{(H)};\begin{bmatrix}1\\1\end{bmatrix}\right)$ (the number of vertical edges in $Hfggr(P)$)

$e_1(P^{(H)})=\#\left(P^{(H)};\begin{bmatrix}&1\\1&\end{bmatrix}\right)$ (the number of right tilted edges in $Hfggr(P)$)

$e_2(P^{(H)})=\#\left(P^{(H)};\begin{bmatrix}1&\\&1\end{bmatrix}\right)$ (the number of left tilted edges in $Hfggr(P)$)

$f_1\left(P^{(H)}\right)=\#\left(P^{(H)};\begin{bmatrix}&1\\1&\\&1\end{bmatrix}\right)$ (the number of left faces in $Hfggr(P)$)

$f_2\left(P^{(H)}\right)=\#\left(P^{(H)};\begin{bmatrix}1&\\&1\\1&\end{bmatrix}\right)$ (the number of right faces in $Hfggr(P)$)

Similarly, the V oriented hexagonal Euler number is defined as

$$\chi\left(P^{(V)}\right)=v(P^{(V)})-e_0(P^{(V)})-e_1(P^{(V)})-e_2(P^{(V)})+f_1(P^{(V)})+f_2(P^{(V)}) \qquad (1V)$$

where

$v\left(P^{(V)}\right)=\#\left(P^{(V)};[1]\right)$ (the number of vertices in $Vfggr(P)$)

$e_0(P^{(V)})=\#\left(P^{(V)};[1\ 1]\right)$ (the number of horizontal edges in $Vfggr(P)$)

$e_1(P^{(V)})=\#\left(P^{(V)};\begin{bmatrix}1&\\&1\end{bmatrix}\right)$ (the number of downward tilted edges in $Vfggr(P)$)

$e_2(P^{(V)})=\#\left(P^{(V)};\begin{bmatrix}&1\\1&\end{bmatrix}\right)$ (the number of upward tilted edges in $Vfggr(P)$)

$f_1\left(P^{(V)}\right)=\#\left(P^{(V)};\begin{bmatrix}1&1\\&1\end{bmatrix}\right)$ (the number of top faces in $Vfggr(P)$)

$f_2\left(P^{(V)}\right)=\#\left(P^{(V)};\begin{bmatrix}1&&1\\&1&\end{bmatrix}\right)$ (the number of bottom faces in $Vfggr(P)$)

See also the references [5], p.186, and [6], pp.79-82.

III. The Perimeter

Given a binary digital image P, the (total) perimeter of its foreground is defined as the length of the boundary. For the H oriented version

$$per\left(P^{(H)}\right)=nww(P^{(H)})+sww(P^{(H)})+south(P^{(H)})+see(P^{(H)})+nee(P^{(H)})+north(P^{(H)}) \qquad (2H)$$

where

$nww(P^{(H)})=\#\left(P^{(H)};\begin{bmatrix}0&1\end{bmatrix}\right)$ (the number of edges that cross the north-west-west boundary)

$sww(P^{(H)})=\#\left(P^{(H)};\begin{bmatrix}0&1\end{bmatrix}\right)$ (the number of edges that cross the south-west-west boundary)

$south(P^{(H)}) = \#\left(P^{(H)};\begin{bmatrix}1\\0\end{bmatrix}\right)$ (the number of edges that cross the southern boundary)

$see(P^{(H)}) = \#\left(P^{(H)};\begin{bmatrix}1 & 0\end{bmatrix}\right)$ (the number of edges that cross the south-east-east boundary)

$nee(P^{(H)}) = \#\left(P^{(H)};\begin{bmatrix}0\\1\end{bmatrix}\right)$ (the number of edges that cross the north-east-east boundary)

$north(P^{(H)}) = \#\left(P^{(H)};\begin{bmatrix}0\\1\end{bmatrix}\right)$ (the number of edges that cross the northern boundary)

For the V oriented version

$$per\left(P^{(V)}\right) = west(P^{(V)}) + ssw(P^{(V)}) + sse(P^{(V)}) + east(P^{(V)}) + nne(P^{(V)}) + nnw(P^{(V)}) \qquad (2V)$$

where

$west(P^{(V)}) = \#\left(P^{(V)};[0\ 1]\right)$ (the number of edges that cross the western boundary)

$ssw(P^{(V)}) = \#\left(P^{(V)};\begin{bmatrix}0 & 1\end{bmatrix}\right)$ (the number of edges that cross the south-south-west boundary)

$sse(P^{(V)}) = \#\left(P^{(V)};\begin{bmatrix}1 & 0\end{bmatrix}\right)$ (the number of edges that cross the south-south-east boundary)

$east(P^{(V)}) = \#\left(P^{(V)};[1\ 0]\right)$ (the number of edges that cross the eastern boundary)

$nne(P^{(V)}) = \#\left(P^{(V)};\begin{bmatrix}1 & 0\end{bmatrix}\right)$ (the number of edges that cross the north-north-east boundary)

$nnw(P^{(V)}) = \#\left(P^{(V)};\begin{bmatrix}0 & 1\end{bmatrix}\right)$ (the number of edges that cross the north-north-west boundary)

The boundary of the image (foreground) consists of connected components (boundaries of objects and boundaries of holes). Our convention is that the boundary of an object is traversed in counterclockwise order; the boundary of a hole is traversed in clockwise order. Each boundary component changes direction at each unit step (this is a feature of the hexagonal grid).

A concept, similar to that of perimeter, was defined in [7]; it is called the *contact perimeter*, but it turns out to be the number of edges in the image foreground:

- in the H oriented version, $cper\left(P^{(H)}\right) = e_0(P^{(H)}) + e_1(P^{(H)}) + e_2(P^{(H)})$;
- in the V oriented version, $cper\left(P^{(V)}\right) = e_0(P^{(V)}) + e_1(P^{(V)}) + e_2(P^{(V)})$.

The two "perimeters" are related by the following formula (in the two versions), as shown in [7] (see also [8], [10], [11])

$$2cper(P^{(H)}) + per\left(P^{(H)}\right) = 6v(P^{(H)}) \qquad (3H)$$

$$2cper(P^{(V)}) + per\left(P^{(V)}\right) = 6v(P^{(V)}) \qquad (3V)$$

We provide here a shorter proof. Each foreground pixel p has six neighbors which define six edges in the foreground or on the boundary. In all, there are $6v(P^{(H)})$ edges counted in the H version, or $6v(P^{(V)})$ edges counted in the V version. Each foreground edge appears twice, each boundary edge only once. All foreground edges and all boundary edges are covered.

From formulas (1*H*) and (3*H*) we get

$$\begin{aligned}\chi\left(P^{(H)}\right) &= v(P^{(H)}) - \left(e_0(P^{(H)}) + e_1(P^{(H)}) + e_2(P^{(H)})\right) + f_1(P^{(H)}) + f_2(P^{(H)}) \\ &= v(P^{(H)}) - \left(3v(P^{(H)}) - per(P^{(H)})/2\right) + f_1(P^{(H)}) + f_2(P^{(H)}) \\ &= -2v(P^{(H)}) + per(P^{(H)})/2 + f_1(P^{(H)}) + f_2(P^{(H)})\end{aligned} \tag{4H}$$

or

$$\begin{aligned}\chi\left(P^{(H)}\right) &= -2v(P^{(H)}) \\ &+ \left(nww(P^{(H)}) + sww(P^{(H)}) + south(P^{(H)}) + see(P^{(H)}) + nee(P^{(H)}) + north(P^{(H)})\right)/2 \\ &+ f_1(P^{(H)}) + f_2(P^{(H)})\end{aligned} \tag{5H}$$

Similarly, from formulas (1*V*) and (3*V*) we get

$$\begin{aligned}\chi\left(P^{(V)}\right) &= v(P^{(V)}) - \left(e_0(P^{(V)}) + e_1(P^{(V)}) + e_2(P^{(V)})\right) + f_1(P^{(V)}) + f_2(P^{(V)}) \\ &= v(P^{(V)}) - \left(3v(P^{(V)}) - per(P^{(V)})/2\right) + f_1(P^{(V)}) + f_2(P^{(V)}) \\ &= -2v(P^{(V)}) + per(P^{(V)})/2 + f_1(P^{(V)}) + f_2(P^{(V)})\end{aligned} \tag{4V}$$

or

$$\begin{aligned}\chi\left(P^{(V)}\right) &= -2v(P^{(V)}) \\ &+ \left(west(P^{(V)}) + ssw(P^{(V)}) + sse(P^{(V)}) + east(P^{(V)}) + nne(P^{(V)}) + nnw(P^{(V)})\right)/2 \\ &+ f_1(P^{(V)}) + f_2(P^{(V)})\end{aligned} \tag{5V}$$

A similar result for binary images on the square grid was derived in [10] and [11] (see also [12]).

In the next section we modify formulas (5*H*) and (5*V*) in order to achieve an improved perimeter-based algorithm for the computation of the Euler number.

IV. The Semi-Perimeter

We trivially define the semi-perimeter of the foreground as $semiper(P^{(H)}) = per(P^{(H)})/2$ and $semiper(P^{(V)}) = per(P^{(V)})/2$. We first prove some simple relations for the semi-perimeter.

In the *H* version, we show that

$$nee(P^{(H)}) = sww(P^{(H)});\ north(P^{(H)}) = south(P^{(H)});\ nww(P^{(H)}) = see(P^{(H)}).$$

Proof: To simplify the notations, we denote

$$a = north(P^{(H)}),\ b = nww(P^{(H)}),\ c = sww(P^{(H)}),\ d = south(P^{(H)}),\ e = see(P^{(H)}),\ f = nee(P^{(H)}).$$

Take a vertical sweepline. On each column position, this line cuts the foreground on a number of 0, 1, or more segments. Each of these segments starts on a northern boundary side and ends on a southern boundary side. Each boundary side is cut once by the sweepline. As a result, $a = d$. Similarly, using a left tilted sweepline, we get $b = e$; using a right tilted sweepline, we get $c = f$.

In the *V* version, we get

$$nnw(P^{(V)}) = sse(P^{(V)})\,;\ west(P^{(V)}) = east(P^{(V)})\,;\ ssw(P^{(V)}) = nne(P^{(V)})\,.$$

A similar proof applies here too.

Therefore,

$$semiper\left(P^{(H)}\right) = sww(P^{(H)}) + south(P^{(H)}) + see(P^{(H)}) \tag{6H}$$

and

$$semiper\left(P^{(V)}\right) = sse(P^{(V)}) + east(P^{(V)}) + nne(P^{(V)}) \tag{6V}$$

These formulas allow us to devise a third algorithm, whose implementation will be compared to the previous two. The formulas we get are

$$\chi\left(P^{(H)}\right) = -2v(P^{(H)}) + semiper(P^{(H)}) + f_1(P^{(H)}) + f_2(P^{(H)}) \tag{7H}$$

$$\chi\left(P^{(V)}\right) = -2v(P^{(V)}) + semiper(P^{(V)}) + f_1(P^{(V)}) + f_2(P^{(V)}) \tag{7V}$$

or

$$\chi\left(P^{(H)}\right) = -2v(P^{(H)}) + sww(P^{(H)}) + south(P^{(H)}) + see(P^{(H)}) + f_1(P^{(H)}) + f_2(P^{(H)}) \tag{8H}$$

$$\chi\left(P^{(V)}\right) = -2v(P^{(V)}) + sse(P^{(V)}) + east(P^{(V)}) + nne(P^{(V)}) + f_1(P^{(V)}) + f_2(P^{(V)}) \tag{8V}$$

A similar result for binary images on the square grid was derived in [12].

V. The Algorithms

The three algorithms are implemented by three functions written in the C programming language. We do this for the *H* orientation version only; the *V* orientation version is expected to return similar results. The function **euler_hexa_def** is based on formula (1*H*); the function **euler_hexa_perim** on formula (5*H*); the function **euler_hexa_sperim** on formula (8*H*).

Algorithm euler_hexa_def:

```
int euler_hexa_def (int nr, int nc, u_char (*pict)[nc])
{
        int v = 0, e = 0, f = 0;  int par = 0;
        for (int i = 0; i < nr – 2; i++, par = 1 - par)
        for (int j = 2 - par; j < nc – 1; j+=2)
                if (pict [i][j])
                {       v++;
                        if (pict [i+2][j])
                        {       e++;
                                if (pict [i+1][j-1])    {e++;  f++;   }
                                if (pict [i+1][j+1])    {e++;  f++;   }
                        }
                        else
```

```
                {
                        if (pict [i+1][j-1])    e++;
                        if (pict [i+1][j+1])    e++;
                }
        }
    return v – e + f;
}
```

Algorithm euler_hexa_perim:

```
int euler_hexa_perim (int nr, int nc, u_char (*pict)[nc])
{
    int v = 0, p = 0, f = 0; int par = 0;
    for (int i = 0; i < nr – 2; i++, par = 1 - par)
    for (int j = 2 - par; j < nc – 1; j+=2)
        if (pict [i][j])
        {   v++;
            if (pict [i+2][j])
            {
                if (pict [i+1][j-1])    f++;
                else                    p++;
                if (pict [i+1][j+1])    f++;
                else                    p++;
            }
            else
            {   p++;
                if (!pict [i+1][j-1])   p++;
                if (!pict [i+1][j+1])   p++;
            }
        }
        else
        {   if (pict [i+2][j])      p++;
            if (pict [i+1][j-1])    p++;
            If (pict [i+1][j+1])    p++;
        }
    return -2*v + p / 2 + f;
}
```

Algorithm euler_hexa_sperim:

```
int euler_hexa_perim (int nr, int nc, u_char (*pict)[nc])
{
    int v = 0, sp = 0, f = 0; int par = 0;
    for (int i = 0; i < nr – 2; i++, par = 1 - par)
    for (int j = 2 - par; j < nc – 1; j+=2)
        if (pict [i][j])
```

```
        {       v++;
                if (pict [i+2][j])
                {
                        if (pict [i+1][j-1])    f++;
                        else                    sp++;
                        if (pict [i+1][j+1])    f++;
                        else                    sp++;
                }
                else
                {       sp++;
                        if (!pict [i+1][j-1])   sp++;
                        if (!pict [i+1][j+1])   sp++;
                }
        }
    return -2*v + sp + f;
}
```

VI. Experimental Results

Randomly generated artificial binary images, some of size 64X64 and some of size 128X64, were input to each of the three algorithm versions. The following two tables show the average number *comp* of comparison operations and the average number *incr* of increment operations (increments of indices are *not* counted).

Images of Size 64X64

version	comp	incr
def	20445	19864
perim	31752	14877
semi-perim	20445	12377

Images of Size 128X128

version	comp	incr
def	37480	33726
perim	64008	24909
semi-perim	37480	20855

VII. Conclusions

As we can see, the results are very similar to those obtained in [12] for binary images on the square grid. The perimeter based algorithm version *euler_hexa_perim* is less efficient than the definition based version *euler_hexa_def*. However, the semi-perimeter based version *euler_hexa_sperim* is the most efficient of all three.

REFERENCES

[1] M.J.E. Golay, “Hexagonal parallel pattern transformations,” IEEE Trans. Comput. 18(8):733-740 (1969).

[2] K. M. Jeevan, V. K. Padmaja, A. B. Anne Gowda, S. Krishnakumar, “Performance analysis of image processing in hexagonal grid based on image compression, denoising and steganography,” Int. J. Engin. Adv. Techn. (IJEAT), 9(1S6):26-31 (2019).

[3] N. I. Rummelt, J. N. Wilson, "Array set addressing: enabling technology for the efficient processing of hexagonally sampled imagery", J. Electron. Imag. 20(2):1-12 (2011).

[4] X. Li, “Storage and addressing scheme for practical hexagonal image processing,” J. Electron. Imag. 22(1):1-3 (2013).

[5] J. Serra, Image Analysis and Mathematical Morphology, Academic Press, London (1982).

[6] B. K. P. Horn, Robot Vision, The MIT Press, Cambridge, Mass. and McGraw-Hill, NY (1986).

[7] E. Bribiesca, "Measuring 2-D shape compactness using the contact perimeter," *Comput. Math. Appl*. 33(11):1-9 (1997).

[8] E. Bribiesca, "Computation of the Euler number using the contact perimeter," *Comput. Math.Appl*. 60(5): 1364-1373 (2010).

[9] J.H. Sossa-Azuela, E. B. Cuevas- Jiménez, D.Zaldivar-Navarro, "Computation of the Euler number of a binary image composed of hexagonal cells," J. Appl. Res. Techn. 8(3):340-351 (2010).

[10] J.H. Sossa-Azuela, E. B. Cuevas- Jiménez, D.Zaldivar-Navarro, "Alternative way to compute the Euler number of a binary image," J. Appl. Res. Techn. 9(3):335-340 (2011).

[11] J. H. Sossa Azuela, E. Rubio-Espino, R. Santiago-Montero, A. López, A. Peña Ayala, E. V. Cuevas Jiménez, "Alternative formulations to compute the binary shape Euler number," IET Comput. Vis. 8(3):171-181 (2014).

[12] J. Weinstein, "An improved algorithm for the computation of the Euler number of a binary image using the perimeter," Int. J. Inform. Comput. Sci. (IJICS) 7(8):1-6 (2020). DOI:16.10089.IJICS.2020.V7I8.18.3395