RECURSIVE, IN-PLACE ALGORITHM FOR THE HEXAGONAL ORTHOGONAL ORIENTED QUADRATURE IMAGE PYRAMID

Andrew B. Watson

Vision Group, NASA Ames Research Center
Moffett Field, CA 94035

ABSTRACT

Pyramid image transforms have proven useful in image coding and pattern recognition. The Hexagonal orthogonal Oriented quadrature image Pyramid (HOP), transforms an image into a set of orthogonal, oriented, odd and even bandpass sub-images. It operates on a hexagonal input lattice, and employs seven kernels, each of which occupies a neighborhood consisting of a point and a hexagon of six nearest neighbors. The kernels consist of one lowpass and six bandpass kernels that are orthogonal, self-similar, and localized in space, spatial frequency, orientation, and phase. The kernels are first applied to the image samples to create the first level of the pyramid, then to the lowpass coefficients to create the next level. The resulting pyramid is a compact, efficient image code. Here we describe a recursive, in-place algorithm for computation of the HOP transform. The transform may be regarded as a depth-first traversal of a tree structure. We show that the algorithm requires a number of operations that is on the order of the number of pixels.

1. INTRODUCTION

The Hexagonal orthogonal Oriented quadrature image Pyramid (HOP) transform converts an image into a set of sub-images that vary in resolution and orientation. It is a pyramid transform in the sense that the size of each sub-image is proportional to resolution[1]. A pyramid transform is analogous to to a subband code [2, 3, 4]. The distinctive features of the HOP transform are that it is a pyramid transform, that it operates on a hexagonal lattice, that it is orthogonal, that the sub-images are bandpass and oriented, that the kernels form quadrature pairs, and that the passbands resemble those of individual neurons in primate visual cortex. More extensive descriptions of the transform are given elsewhere [5, 6, 7, 8]. The purpose of this paper is to describe a recursive, in-place algorithm for computing the HOP transform of a square image.

2. HOP TRANSFORM

The HOP transform is characterized by a set of seven kernels, each with seven coefficients arranged in a hexagon (Fig. 1).

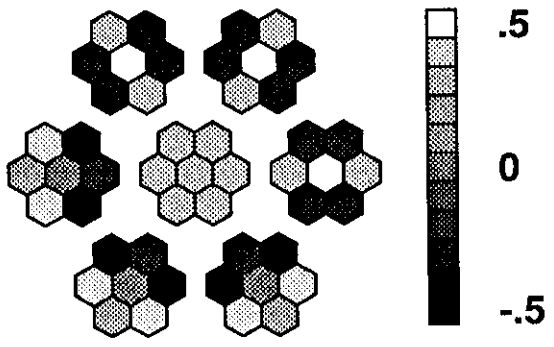

Figure 1. Seven HOP kernels represented by greylevels. The lowpass kernel is at the center, the three even kernels are in the upper right, and the three odd are to the lower left.

The seven kernels are derived in such a way that all are orthonormal, three are even, three are odd, and one is lowpass. Within each hexagonal kernel, the samples are ordered in *neighborhood order* (this order is arbitrary, but we proceed from the center to the right and then counterclockwise). The seven kernels can be represented as the rows of a transformation matrix H, in which the column index corresponds to the neighborhood order.

$$H = \begin{bmatrix} h & h & h & h & h & h & h \\ a & b & b & c & b & b & c \\ a & c & b & b & c & b & b \\ a & b & c & b & b & c & b \\ 0 & -e & e & f & e & -e & -f \\ 0 & -f & -e & e & f & e & -e \\ 0 & -e & -f & -e & e & f & e \end{bmatrix} \quad (1)$$

where

$$h = 1/\sqrt{7} \quad (2)$$
$$a = \sqrt{2}\,h \quad (3)$$
$$f = \sqrt{2}/6 \quad (4)$$
$$b = -(1+h)f \quad (5)$$
$$c = (2-h)f \quad (6)$$
$$e = 2f \quad (7)$$

The HOP transform generates seven sub-images at each resolution. The highest resolution (level 0) sub-images are produced by applying the seven kernels to hexagonal neighborhoods that completely tile the image. The result of each application is a single coefficient that constitutes a pixel of the sub-image. Thus each sub-image is one seventh the size of the original. The sub-images at the next lower resolution (level 1) are produced by applying the seven kernels to the lowpass sub-image from level 0. The remaining levels are created in the same way, by progressive transformation of the lowpass sub-image from the previous level.

3. SQUARE COORDINATES

Although designed on a hexagonal lattice, the HOP transform may be applied to pixels on a square lattice. The square lattice is regarded as a skewed hexagonal lattice. This transforms the hexagonal neighborhood into a *quasihexagonal* neighborhood (Fig. 2).

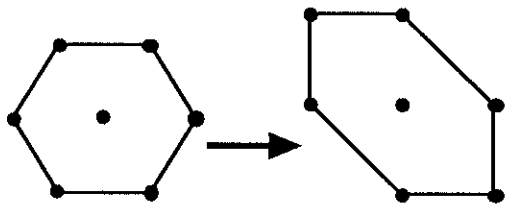

Figure 2. Hexagonal and quasihexagonal neighborhoods.

This purely geometric transformation has no effect on the arithmetic of the HOP transform, but alters the shape of the kernels, and consequently, the passbands. For example, they are no longer invariant with respect to rotation. The other requirement for application to a square image is that the image must be a power of 7 on each side.

4. PIXEL TREE

The pixels in the image may be regarded as the nodes in an inverted tree structure with seven-fold branching, as shown in Fig. 3 (for clarity, only three-fold branching is shown).

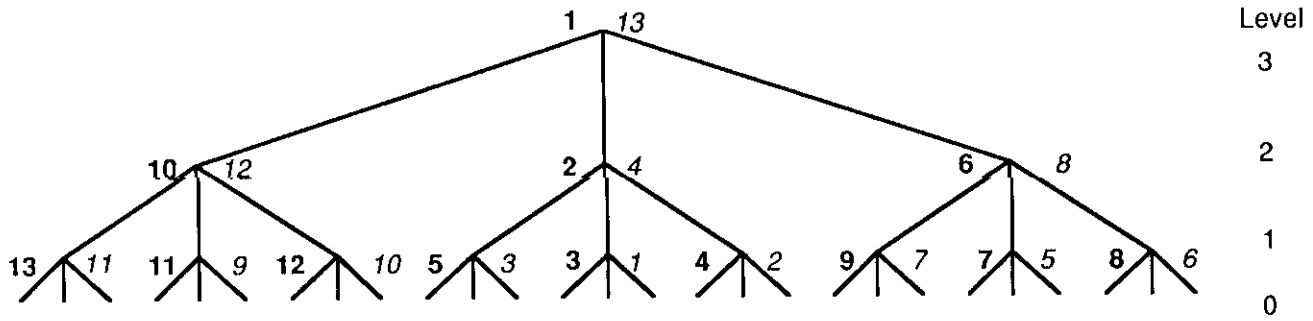

Figure 3. An image represented as a tree structure. The bold numbers give the order in which nodes are visited, the italic numbers give the order of application of local transforms.

The terminal nodes of the tree (level 0) are the image pixels. These pixels are grouped into quasihexagonal neighborhoods that collectively tile the image. The parent node of each neighborhood is the center sample. These parent nodes may also be grouped into quasihexagonal neighborhoods (level 1). (Note that the pixels of level 1 are also nodes of level 0.) The centers of the level 1 neighborhoods form the nodes of level 2, and so on. The root of the tree is the center pixel.

The complete transform is simply a matter of applying the same local transform to each neighborhood at each level of the tree, but doing so in the correct order. This order is given by a depth-first tree traversal. At each node, starting from the root, we chose the next unvisited branch, proceeding in neighborhood order. If there are no remaining untried branches, or if we have reached the terminal nodes, we ascend one level. A neighborhood is transformed whenever maximum depth is reached, or when all the branches of a node have been traversed. This sort of tree traversal is pictured in Fig. 3. The bold numbers indicate the order in which the nodes are visitied, and italic numbers show the order in which local transforms would be computed. This pattern of tree traversal can be achieved efficiently with a recursive process.

5. RECURSION

The recursive process is described by the following pseudocode.

```
main()
{       xc = yc = width/2;
        level = log7(width*width);
        descend();
}

descend()
{       x = xc;
        y = yc;
        level--;
        if( level > 1)
        {       descend();
                for(i=1;i<7;i++) branch(i);
                xc = x;
                yc = y;
        }
        transform();
        level++;
}

branch(node)
{       xc += xo[level][node];
        yc += yo[level][node];
        descend();
}

transform()
{       compute coordinates of samples N in neighborhood;
        transform as N -> HN;
}
```

We begin by setting the current neighborhood center (**xc,yc**) to the center of the image, and the current level to the number of levels. We then call the routine **descend()**. This routine sets a temporary location equal to the current center, descends one level, and checks whether we have reached level 1. If not, **descend()** is called (recursively). This recursive descent continues until **level==1**. At that point, the neighborhood of pixels is transformed. The procedure then ascends one level, and exits the deepest call to **descend()**. The procedure then moves to each remaining sample in the neighborhood at that level. In the routine **branch()**, the current center is set to that sample location, and **descend()** is called again.

The routine **transform()** multiplies the neighborhood of seven samples by each of the seven kernels. If the neighborhood is represented as a column vector N, the transformation can be represented as multiplication by the matrix H

$$N \rightarrow HN \tag{8}$$

6. INVERSE TRANSFORM

Only two small changes to the algorithm are required to compute the inverse transform. The first is that H^{-1}, the inverse of the transformation matrix H, is used in **transform()**. Since H is orthonormal, $H^{-1} = H^T$. The second is that the **transform()** routine is called upon entry to **descend()**, rather than just before exit. This means that each parent neighborhood is transformed before its children.

The use of recursion leads to very compact code, but may require exorbitant stack memory. This is not likely to be a problem here, as the maximum depth of recursion is equal to one less than the number of levels. The number of levels is equal to the log to the base 7 of the number of pixels.

7. ADDRESS CALCULATIONS

Much of the work in the HOP transform is address calculations. These consist of two parts. The first is generation of a vector of offsets that describe the location of the next sample in the neighborhood, relative to the current sample. The second is the calculation of absolute coordinates, based on these offsets and a neighborhood center location. The former may be done once, before the transform begins. The second must be done each time the routine **transform()** is called.

As noted above, conversion from hexagonal to a square lattice transforms the neighborhood shape from hexagonal to quasihexagonal. We represent the coordinates of the samples within the neighborhood by a matrix, R, with two rows and seven columns, wherein each column vector represents the coordinates of one sample relative to the center. At level 0, these coordinates are

$$R_0 = \begin{bmatrix} 0 & 1 & -1 & -1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & -1 & -1 & 0 \end{bmatrix} \tag{9}$$

The order of samples is pictured in Fig. 4.

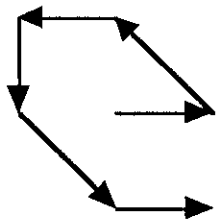

Figure 4. Locations of samples in a neighborhood at level 0. Arrows indicate neighborhood order of samples.

When the image is tiled with neighborhoods like that in Fig. 4, the neighborhood centers form a new lattice. There are in fact two ways to tile the image; we chose the one defined by the sampling matrix

$$S_0 = \begin{bmatrix} 2 & -1 \\ 1 & 3 \end{bmatrix} \tag{10}$$

Each neighborhood center is an integer linear combination of the column vectors of the sampling matrix S_0. This tiling is shown in Fig. 5.

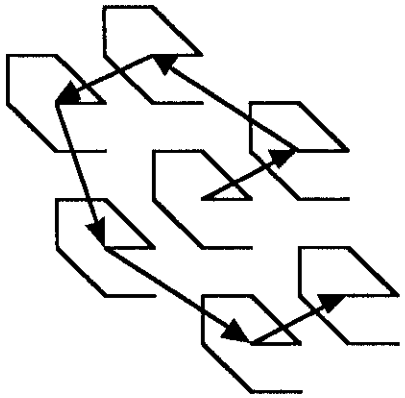

Figure 5. Tiling pattern for quasihexagonal neighborhoods at level 0. The arrows show one neighborhood from level 1, composed of neighborhood centers from level 0.

We also construct a new quasihexagonal neighborhood, shown in Fig. 5, whose relative sample locations (offsets) are given by the column vectors of the matrix S_0R_0. These new neighborhoods also tile the image, and their centers are integer linear combinations of the sampling matrix S_0E, where

$$E = 7S_0^{-1} = \begin{bmatrix} 3 & 1 \\ -1 & 2 \end{bmatrix} \qquad (11)$$

Note that the new sample lattice defined by S_0E is again square, but seven times larger than the lattice at level 0. This process of creating larger and larger quasihexagonal tilings of the image can be repeated until the image accommodates only one neighborhood. The sampling matrices at each level are given by

$$S_{level} = S_0 E S_0 E \cdots (\text{to } \textit{level} \text{ terms}) \qquad (12)$$

The offsets at each level are given by

$$R_{level} = R_0 S_{level} \qquad (13)$$

Since the offsets are used repeatedly, they are best computed once at the start of the transform program, or stored as constants.

When applied to a square image, the computed address will sometimes fall outside the bounds of the image. In this case, the address is "wrapped around" until it falls within the bounds of the square image.

8. COMPLEXITY

The computational complexity of image transforms is of great interest, since current computing equipment operating on typical image sizes is slow relative to the response time of the human eye (.02 sec) or even the interaction speed of the human user (1 sec). As a standard of comparison, the FFT requires a number of operations that is order of NlogN. A remarkable feature of the HOP transform is it requires a number of operations that is order of N. Consider an image with $N=7^n$ pixels. The HOP transform will have n levels. Within each neighborhood 7^2 operations are required. Table 1 shows the number of operations required at each level, as well as the total.

level	neighborhoods	operations
0	7^{n-1}	7^{n+1}
i	7^{n-1-i}	7^{n+1-i}
n-1	1	7^2
total		$\sum_{i=0}^{n-1} 7^{n+1-i}$

Table 1. Operations required for the HOP transform.

As n increases, this total rapidly approaches $7^{n+2}/6$. This corresponds to a constant 8.17 operations per pixel. This is an upper bound, for small images fewer operations are required.

9. ACKNOWLEDGEMENTS

I thank Albert J. Ahumada, Jr. for many useful discussions. This work supported by NASA RTOP 505-67.

10. REFERENCES

1. S. Tanimoto and T. Pavlidis,"A Hierarchical data structure for picture processing," Computer Graphics and Image Processing, **4**, 104-119 (1975).

2. J.W. Woods and S.D. O'Neil,"Subband coding of images," IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP-**34**(5), 1278-1288 (1986).

3. M. Vetterli,"Multi-dimensional sub-band coding: some theory and algorithms," Signal Processing, **6**, 97-112 (1984).

4. E.H. Adelson, E. Simoncelli and R. Hingorani,"Orthogonal pyramid transforms for image coding," Proceedings of the SPIE, **845** (1987).

5. A.B. Watson and A.J. Ahumada Jr., "An orthogonal oriented quadrature hexagonal image pyramid," NASA Technical Memorandum **100054**, (1987).

6. A.B. Watson,"Cortical Algotecture," in *Vision: Coding and Efficiency*, C. B. Blakemore , ed., Cambridge: Cambridge University Press (in press).

7. A.B. Watson and A.J. Ahumada Jr.,"A hexagonal orthogonal oriented pyramid as a model of image representation in visual cortex," IEEE Trans. Biomed. Eng., **36**(1), 97-106 (1989).

8. A.B. Watson,"Receptive fields and visual representations," SPIE Proceedings, **1077**, (1989).