Geometric Transformations of BDD Encoded Image

Watis Leelapatra
Kanchana Kanchanasut *

Chidchanok Lursinsap †

Abstract—**Binary Decision Diagram (BDD) has been shown to be applicable in image coding. A technique presented in this paper extends the functionality of BDD in image coding to cover major geometric transformations of image such as translation, rotations, scaling and shearing. By introducing the concept of transition branch, nonuniform and uniform translation of image by a given displacement could be obtained using only exclusive-or operation. Based on the translation, we show that image rotation, scaling and shearing could be achieved.**

Keywords: BDD, image coding, translation, rotation, scaling, shearing

1 Introduction

Binary decision diagram (BDD), which can be used for representing Boolean function [1], is a rooted acyclic graph that contains nonterminal and terminal nodes. Each nonterminal node of BDD is associated with a Boolean variable, and two terminal nodes are assigned to constant 0 and 1. Each nonterminal node has two outgoing edges, zero-edge and one-edge, that link to its children. Either one of these edges is taken depending on the value of the variable associating with the nonterminal node. The represented Boolean function can be evaluated by traversing through all nodes along the path from the root to the terminal node. The interesting property of the BDD is that it can be made a canonical representation of Boolean function by applying reduction rules presented in [2]. The results from applying reduction rules are that the size of BDD is reduced and is uniquely representing Boolean function. This becomes advantage for image coding when consider a black-and-white image as a Karnaugh map of $\lceil log_2 M \rceil + \lceil log_2 N \rceil$ variables, where M and N are width and height of image, respectively [3], [6]. Detail of representing image using BDD is explained in Section 2.

2 Image Representation

In this paper, a rectangular image of size $W \times H$ pixel is considered as a Karnaugh map of w+h variables where $w = \lceil log_2 W \rceil$, $h = \lceil log_2 H \rceil$ and the variable ordering is: $x_0, x_1, ..., x_{w-1}, y_0, y_1, ..., y_{h-1}$. The h-bit and w-bit Gray codes are used to represent the rows and columns of the Karnaugh map, respectively as shown in Figure 1. We will refer to the rows and columns of the Karnaugh map as the Gray code coordinates throughout this paper.

2.1 Definitions of BDD Components

The definitions of term describing components of a BDD used in our approach for image representation are as follows.

Definition 1 A BDD is a set of nodes **V** and edges **E** that forms an undirected rooted tree. A nonterminal node v_i of BDD is associated with a Boolean variable i. The evaluation of Boolean variable i yields outgoing edge e_i whose attribute is 0 when the variable is evaluated as 0, and is 1 when the variable is evaluated as 1. There is only one terminal node **T** which is assigned to constant 1.

Definition 2 A path of BDD is a set of nodes and edges that appears in an alternating sequence $\{v_1, e_1, v_2, e_2, ..., e_{n-1}, v_n\}$, where $n = w + h$, with restriction that each node and each edge must appear only once.

Definition 3 A branch B_i of BDD is a path whose v_1 is the root of BDD and v_n, where $n = w + h$, is the terminal node. Using our definitions, a branch of BDD shown in Figure 1(b) can be written as $B = \{x_0, e_1, x_1, e_2, ..., e_{w-1}, x_{w-1}, e_w, y_0, e_{w+1}, y_1, ..., e_{w+h-1}, y_{h-1}, e_{w+h}, T\}$.

A branch B of BDD encodes the coordinate (x, y) of a pixel and can be decomposed into two parts, a subbranch B^x which contains information of x coordinate and a subbranch B^y contains information of y coordinate written as $B^x = \{x_0, e_1, x_1, e_2, ..., e_{w-1}, x_{w-1}, e_w, y_0\}$, and $B^y = \{y_0, e_{w+1}, y_1, e_{w+2}, ..., e_{w+h-1}, y_{h-1}, e_{w+h}, T\}$.

2.2 Definitions of BDD Operations

Given branch $B_i = \{v_1^i, e_1^i, ..., e_{n-1}^i, v_n^i\}$ and branch $B_j = \{v_1^j, e_1^j, ..., e_{n-1}^j, v_n^j\}$ where $n = w + h$.

Definition 4 An exclusive-or operation between branch B_i and B_j yields a branch B_k if and only if sequence

*Computer Science and Information Management Department, Asian Institute of Technology, Klong Luang, Pathumthani, Thailand 12120. Email: {watis,kk}@cs.ait.ac.th

†Department of Mathematics, Chulalongkorn University Bangkok, Thailand 10330. Email: chidchanok.l@chula.ac.th

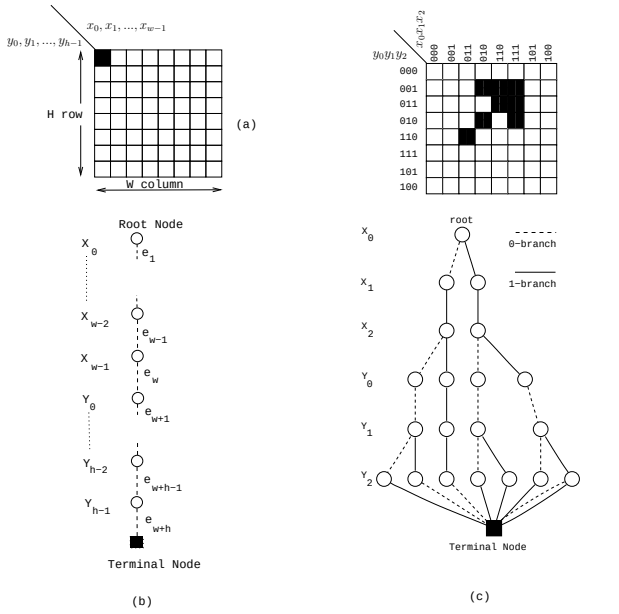

Figure 1: (a) Binary image. (b) BDD representation. (c) Example of a binary image and its BDD representation.

of nodes from both branches are orderly identical, that is $v_1^i = v_1^j, v_2^i = v_2^j, ..., v_n^i = v_n^j$ and the branch B_k is determined by $B_k = \{v_1^i, e_1^i \oplus e_1^j, ..., e_{n-1}^i \oplus e_{n-1}^j, v_n^i\}$

Definition 5 Let p be the number of edges whose attribute is one. The parity of a branch is even if p is even, otherwise the parity is said to be odd.

Definition 6 Left shifting of a branch replaces the attribute of edge e_i by the attribute of edge e_{i+1} and assigns the attribute of e_{n-1} to 0.

Definition 7 Right shifting of a branch replaces the attribute of edge e_{i+1} by the attribute of edge e_i and assigns the attribute of e_1 to 0.

3 Transition Branch

Suppose that B_p is a branch of a BDD representing a pixel p at its original location. Let p' be the pixel p after translation and p' is represented by branch B'_p. The branch B'_p can be obtained by applying exclusive-or between branch B_p and a transition branch Bt.

3.1 Transition Branch of Unit Displacement

A transition branch $Bt = \{x_0, e_1^t, x_1, e_2^t, ..., e_{w-1}^t, x_{w-1}, e_w^t, y_0, e_{w+1}^t, ..., e_{w+h-1}^t, y_{h-1}, e_{w+h}^t, T\}$ can be decomposed into two parts, Bt^x and Bt^y where $Bt^x = \{x_0, e_1^t, x_1, e_2^t, ..., e_{w-1}^t, x_{w-1}, e_w^t, y_0\}$ and $Bt^y = \{y_0, e_{w+1}^t, ..., e_{w+h-1}^t, y_{h-1}, e_{w+h}^t, T\}$.

The transition branch Bt^x is constructed depending on the parity of B^x as follows:

Case 1.1 Parity of B^x is even. Set the attribute of e_w^t of Bt^x to 1. Set the attributes of other edges to 0.

Case 1.2 Parity of B^x is odd. Let e be an edge in B^x and e^t be an edge in Bt^x. Locate the first edge e_{w-j}, $0 \leq j \leq w-1$, such that $e_{w-j} = 1$. Set attribute of e_{w-j-1}^t to 1, and attribute of the remaining edges are set to 0.

Similarly, the transition branch Bt^y is constructed depending on the parity of B^y as follows:

Case 2.1 Parity of B^y is even. Set the attribute of e_{w+h}^t to 1. Set the attributes of other edges to 0.

Case 2.2 Parity of B^y is odd. Let e be an edge in B^y and e^t be an edge in Bt^y. Locate the first edge e_{w+h-k}, $w+1 \leq k \leq w+h-1$ such that $e_{w+h-k} = 1$. Set attribute of $e_{w+h-k-1}^t$ to 1, and attribute of the remaining edges are set to 0.

After Bt^x and Bt^y are constructed, we merge them to form Bt. A transition branch representing negative can be constructed by simply exchanging the case of parity from even to odd and vice versa.

An example of translation is shown in Figure 2. A given displacement of translation is (1,-1). To clarify the steps of the algorithm, we consider these branches individually as shown in Figure 3. The algorithm starts by constructing transition branch Bt_1, Bt_2 and Bt_3 corresponding to B_1, B_2 and B_3, respectively. To construct Bt_1, we consider the parity of B_1^x. Since the parity of B_1^x is odd, we follow the procedure as stated in case 1.1. Bt_1^y is obtained by considering the parity of B_1^y. Since the parity of B_1^y is odd and the displacement is -1, we follow procedures stated in case 2.2. But the parity condition has to be switched. Merging Bt_1^x and Bt_1^y yields Bt_1 as shown in Figure 3(d). We repeat the process to construct Bt_2 and Bt_3. The branches of BDD after the image is translated is obtained by $B'_1 = B_1 \oplus Bt_1$, $B'_2 = B_2 \oplus Bt_2$ and $B'_3 = B_3 \oplus Bt_3$.

3.2 Transition Branch of 2^n Displacement

Given $B^x = \{x_0, e_1, x_1, e_2, ..., e_{w-1}, x_{w-1}, e_w, y_0\}$ and a displacement of translation $d = 2^n, n = 1, 2, 3, ...,$ $\lceil log_2 W \rceil$ where W is the width of the image. The transition branch Bt^x of B^x can be constructed by the following procedures:

step 1 Shift B^x to the right by n positions, then assign to the temporary branch B'^x, that is $B'^x = \{x_0, 0, x_1, ..., 0, x_n, e_1, ..., e_w, y_0\}$.

step 2 Using procedures described in Section 3.1 to determine the transition branch Bt^x of B'^x. Suppose that $Bt^x = \{x_0, e_1^t, x_1, e_2^t, ..., e_{w-1}^t, x_{w-1}, e_w^t, y_0\}$.

step 3 Shift edges of the branch Bt^x from step 2 to

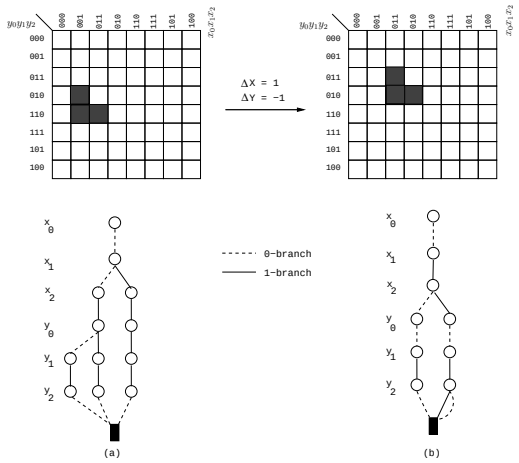

Figure 2: (a) BDD representation of image at original location. (b) BDD after translation.

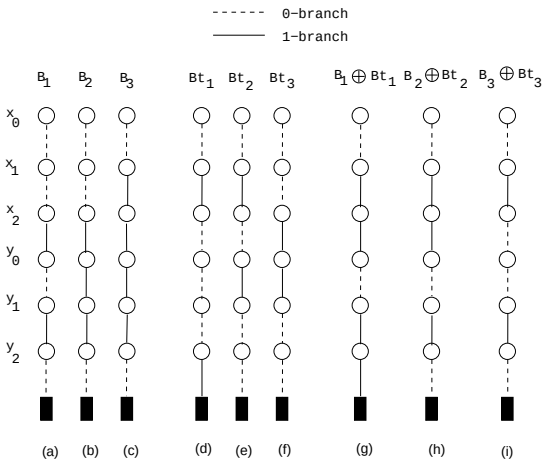

Figure 3: (a), (b) and (c) are branches of the original BDD. (d), (e) and (f) are constructed transition branches. (g), (h) and (i) are results after translation.

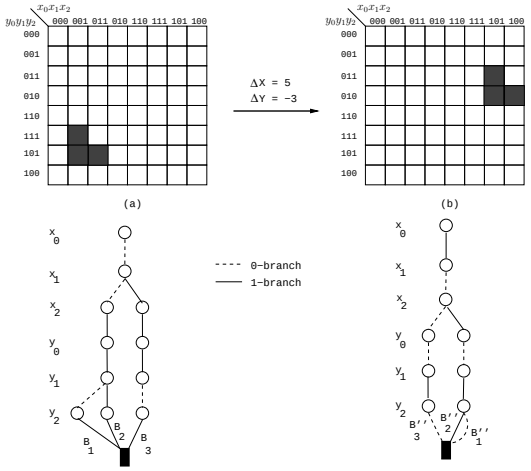

Figure 4: (a) BDD representation of image at original location. (b) BDD and image after translation.

the left one position, then assign to the temporary branch Bt'^x.

step 4 Set the attribute of the rightmost edge of Bt'^x to 1. The content of Bt'^x becomes $Bt'^x = \{x_0, e^t_2, x_1, e^t_3, ..., e^t_w, x_{w-1}, 1, y_0\}$.

step 5 Continue shifting the edges of Bt'^x to the left by n - 1 positions. The content of Bt'^x after shifting becomes $Bt'^x = \{x_0, e^t_{n+1}, x_1, e^t_{n+2}, ..., e^t_w, x_{w-n-1}, 1, 0, ..., 0, y_0\}$.

After the process terminates, the content of Bt'^x becomes the desired transition branch Bt^x. Similarly, Bt^y could be constructed by repeating step 1-5. After Bt^x and Bt^y are obtained, we merge them to form Bt .

3.3 Arbitrary Translation

The general rules of translation by an arbitrary displacement d , $d \leq 2^m - 1$ where m is the number of bits of the Gray code coordinates, are as follows. First, divide the translation of displacement d into k steps such that $d = a_{k-1} \cdot 2^{k-1} + a_{k-2} \cdot 2^{k-2} + ... + a_1 \cdot 2^1 + a_0 \cdot 2^0$ where $a_{k-1}, a_{k-2}, ..., a_1, a_0 \in \{0, 1\}$. Then successively apply the translation of nonzero displacement $a_i \cdot 2^i$ for k times. For example, translation of displacement $d = 13$ can be achieved by dividing translation into $k = 3$ steps since $d = 8 + 4 + 1 = 2^3 + 2^2 + 2^0$ which means we translate the object by 8 pixels first then follow by 4 pixels and then 1 pixel.

We choose the branch B_1 of the BDD from Figure 4 for demonstration as shown in Figure 5. Note that we use the prime ($'$) to indicate the temporary storage used in our algorithm. In Figure 5, the algorithm is applied to the subbranch B^x_1 and B^y_1, separately. The branch B'^x_1 and B'^y_1, as shown in Figure 5(b), are the result from right shifting of B^x_1 by two positions and B^y_1 by one position,

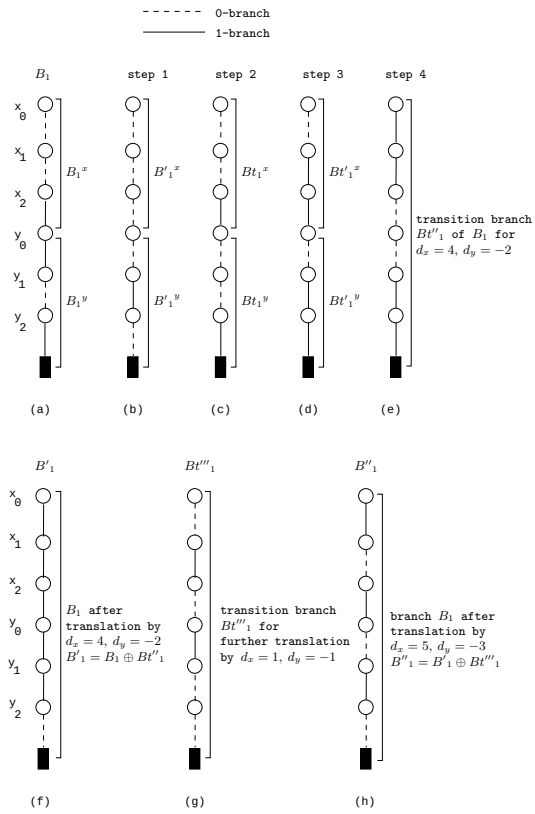

Figure 5: Detail of the branch from Figure 4 when the algorithm is applied.

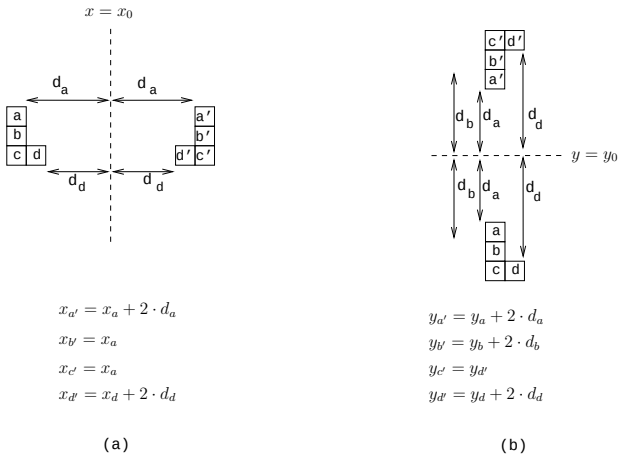

Figure 6: Image flipping. (a) vertical flipping (b) horizontal flipping.

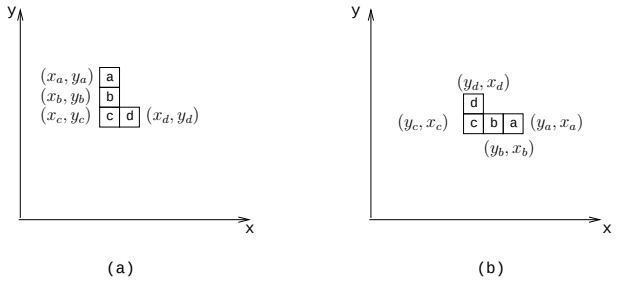

Figure 7: Coordinate Swapping.

respectively. Next, we determine the transition branch Bt_1^x of ${B'_1}^x$ and Bt_1^y of ${B'_1}^y$ using procedure described in Section 3.1. The result is shown in Figure 5(c). By shifting Bt_1^x and Bt_1^y to the left by one position and then setting the attribute of edge e_3 and e_6 to 1, we have ${Bt'_1}^x$ and ${Bt'_1}^y$ as shown in Figure 5(d). Since $n = 2$ for $\triangle x$, we continue shifting ${Bt'_1}^x$ further one position to the left. The algorithm stops processing the ${Bt'_1}^y$ because $n = 1$. Therefore, we have the transition branch of B_1 for displacement (4, -2) as shown in Figure 5(e). Next, we exclusive-or the branch B_1 to the transition branch we obtained. The result is branch B_1 after translation by (4,-2) as shown in Figure 5(f). We repeat the algorithm again for the successive translation by (1, -1). The algorithm starts with the branch B'_1, which is B_1 after translation by (-4, 2). The transition branch Bt'''_1 of B'_1 is shown in Figure 5(g). The branch ${B''}_1$, which is the complete translation of B_1, can be obtained from exclusive-or the transition branch Bt'''_1 to the branch B'_1 as shown in Figure 5(h).

4 Orthogonal Image Rotation

4.1 Image Flipping

Definition 8 Image flipping along an axis $x = x_0$, $x_0 > 0$ is an operation that moves every pixel i of the image from (x_i, y_i) to the new coordinate (x'_i, y'_i) defined by $x'_i = x_i + 2 \cdot d_i$, $d_i = x_o - x_i$ and $y'_i = y_i$.

Definition 9 Image flipping along an axis $y = y_0$, $y_0 > 0$ is an operation that moves every pixel i of the image from (x_i, y_i) to the new coordinate (x'_i, y'_j) defined by $y'_i = y_i + 2 \cdot d_i$, $d_i = y_0 - y_i$ and $x'_i = x_i$.

Flipping operation is equivalent to translation of each pixel by $2 \cdot d_i$. Figure 6 shows the concept of image flipping when considered as nonuniform translation.

4.2 Coordinate Swapping

Definition 10 Coordinate swapping is an operation that moves every pixel i of the image from (x_i, y_i) to the new coordinate (x'_i, y'_j) defined by $x'_i = y_i$ and $y'_i = x_i$.

Coordinate swapping is equivalent to exchanging of edges from subbranch B^x and B^y of a branch B. Figure 7 shows the concept of coordinate swapping.

Image rotation of 90^o can be achieved by coordinate swapping, and then follow by flipping along $y = y_0$ axis. The rotation of 180^o is achieved by flipping along $y = y_0$ axis, and then follow by flipping along $x = x_0$ axis. The rotation of 270^o is achieved by flipping along $y = y_0$ axis, and then follow by coordinate swapping. Figure 8 shows

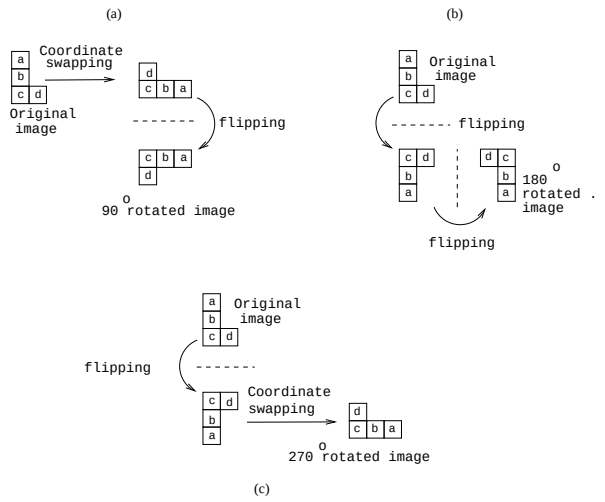

Figure 8: Image rotation. (a) 90^o (b) 180^o and (c) 270^o.

process of rotation.

5 Arbitrary Rotation

Image rotation by an arbitrary angle α, $0 \leq \alpha \leq 90^o$ about the point (x_o, y_o) can be considered as the translation as follows. Let (x_i, y_i) be the coordinate of the pixel i at the original location and (x'_i, y'_i) be the coordinate of the pixel i after rotation. The coordinates x'_i and y'_i are determined by $x'_i = x_i \cdot cos(\alpha) - y_i \cdot sin(\alpha)$ and $y'_i = x_i \cdot sin(\alpha) + y_i \cdot cos(\alpha)$. Then, we calculate the displacement for each pixel individually, that is $\triangle x_i = x'_i - x_i$ and $\triangle y_i = y'_i - y_i$. Given the original location and the displacement, we apply the translation procedures as described in Section 3.2 and 3.3.

6 Image Shearing

Shearing is a transformation of image that every pixel i of the image is moved from the location (x_i, y_i) to the new location (x'_i, y'_i) defined by $x'_i = x_i + a \cdot y_i$ and $y'_i = y_i + b \cdot x_i$. The coefficients a and b are shearing factor which control shape of the image after shearing. If $a = 0$, the result is called shearing along y-axis. If $b = 0$, the result is called shearing along x-axis. After the new location (x'_i, y'_i) is obtained, we calculate the displacement of pixel i. Given the new location and the displacement, the translation algorithm can be applied to perform the image shearing.

7 Image Scaling

Image scaling enlarges or reduces the image by the given scaling factor about the origin. Image scaling can be considered as the translation that moves the pixel i from (x_i, y_i) to the new location (x'_i, y'_i) defined by $x'_i = S_x \cdot x_i$ and $y'_i = S_y \cdot y_i$. S_x and S_y are called scaling factor. After the new location (x'_i, y'_i) is obtained, we calculate the displacement of pixel i. Given the new location and the displacement, the translation algorithm can be applied to perform the image scaling.

8 Computational Requirements

Since the proposed algorithm is based on the construction of transition branch which involves parity checking and branch searching. The best case is when branch searching is not required. In this case, the construction of the transition branch can be done in $O(1)$. If branch searching takes place, the construction of the transition branch can be done in time linearly to the height of the branch or $O(w + h)$. If the BDD contains n branches, then the required time is $O(n(w + h))$. The transition branch for an arbitrary translation displacement requires additional time for edge shifting which is linear to the height of the BDD or $O(w + h)$. The running time also depends on the value of translation displacement. The best case is when the displacement of translation is power of 2 because only one transition branch has to be constructed. In practice, the translation of displacement which is not power of two is done by using the combination of decremental and incremental translation. For example, given $d = 255$. If only the incremental translation is used, then the translation must be divided into 7 steps. The smaller number of steps could be achieved by translating by $d =$ 256 first, and, then by $d =$ -1. For geometric transformations which rely on nonuniform translation, we need to construct transition branches for each branch of the BDD representation of the image independently.

9 Experimental Results

The performance of the proposed algorithm is compared to the well-known standard lossless image coding algorithms such as GIF, JPEG-LS, JBIG and PNG. The performance of algorithms under test are measured in term of execution time when performing transformations of the test images encoded in their native format. The algorithms under test are implemented using C language and compiled with GNU gcc compiler running on 1.8 GHz Pentium4 PC with 256MB of memory under Linux operating system. We use the standard test image *Lena*, which is converted to bi-level 256x256 and 64x64 pixels in our measurement. Figure 9-13 and 14-18 show the transformation performance of 64x64 and 256x256 pixels images, respectively. Sample of images obtained from the experiment are shown in Figure 19-24.

10 Conclusion

We have shown that the geometric transformations of an image represented by a BDD can be expressed using only BDD manipulation process. From the experimental results, the performance of our proposed algorithm is comparable to those standard lossless image coding

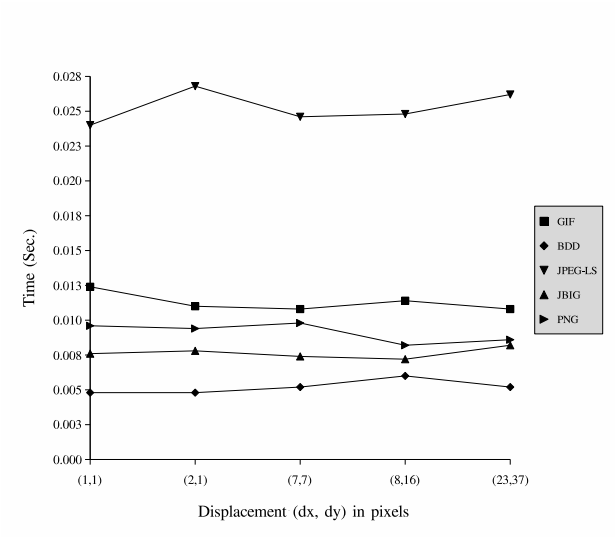

Figure 9: Performance comparison of image translation (64x64 pixels).

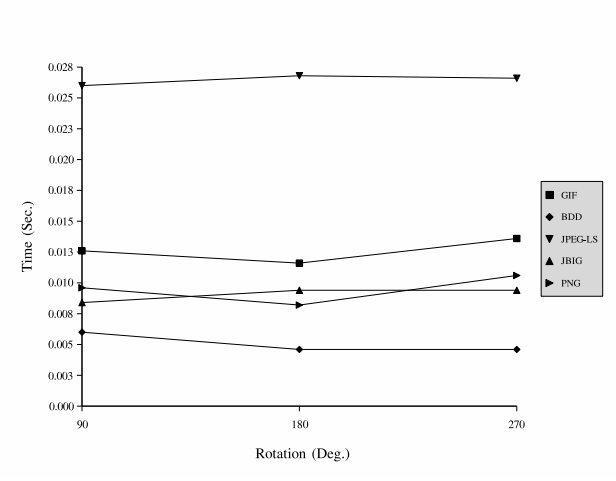

Figure 10: Performance comparison of image orthogonal rotation (64x64 pixels).

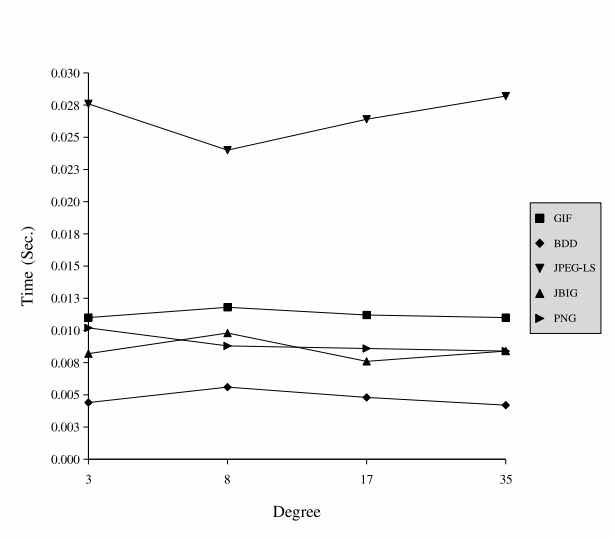

Figure 11: Performance comparison of image arbitrary rotation (64x64 pixels).

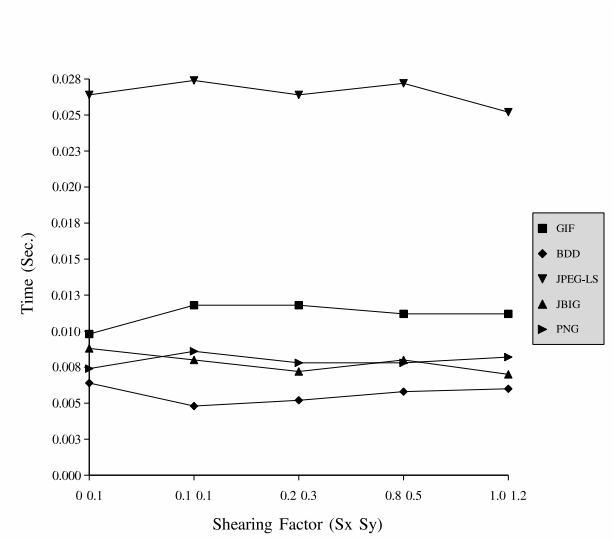

Figure 12: Performance comparison of image shearing (64x64 pixels).

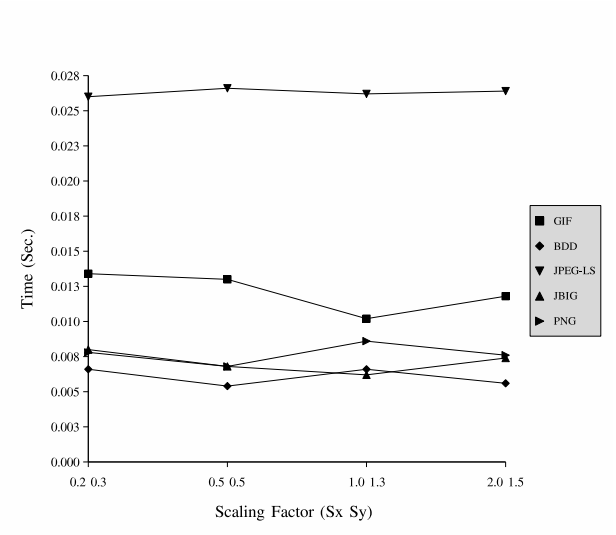

Figure 13: Performance comparison of image scaling (64x64 pixels).

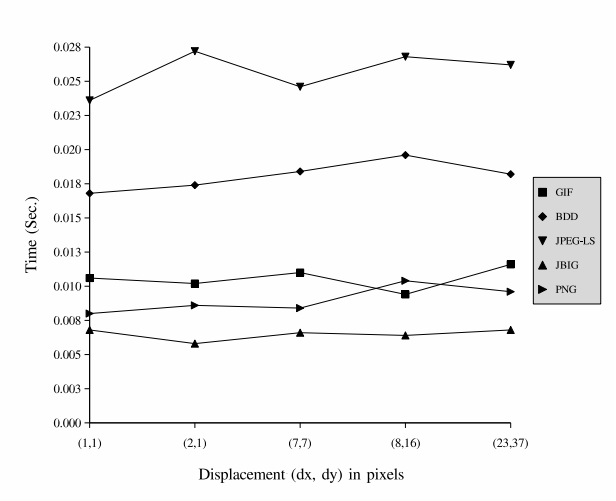

Figure 14: Performance comparison of image translation (256x256 pixels).

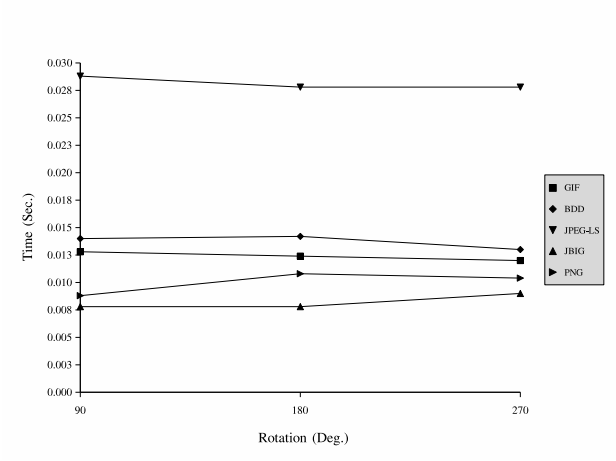

Figure 15: Performance comparison of image orthogonal rotation (256x256 pixels).

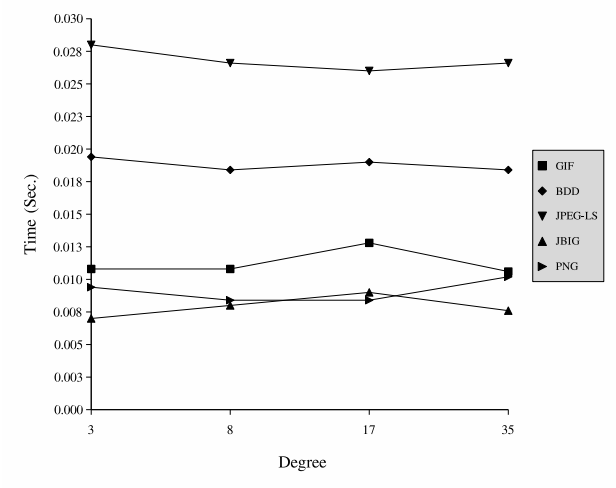

Figure 16: Performance comparison of image arbitrary rotation (256x256 pixels).

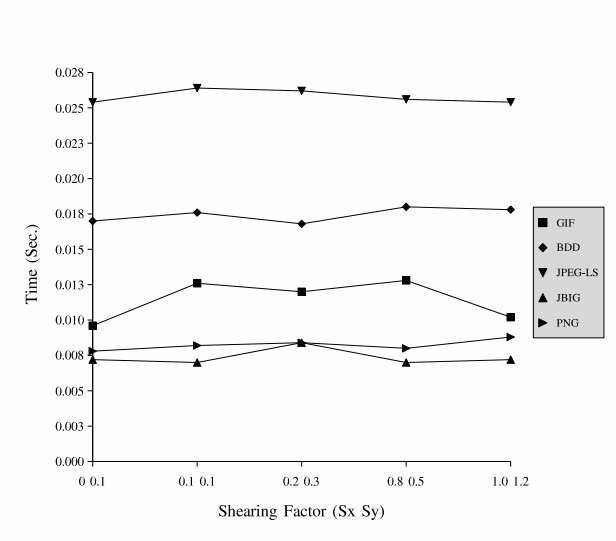

Figure 17: Performance comparison of image shearing (256x256 pixels).

Figure 19: Original Lena image (64x64 pixels).

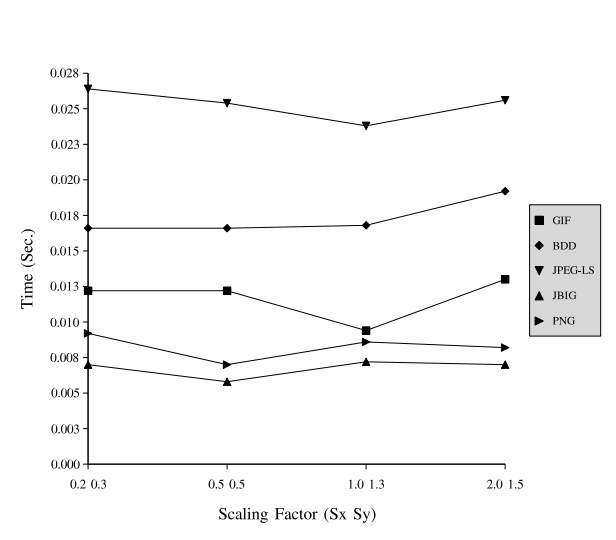

Figure 18: Performance comparison of image scaling (256x256 pixels).

Figure 20: Image Translation by (37, 23) pixels.

Figure 21: Image Orthogonal Rotation by 180 degrees.

Figure 22: Image Arbitrary Rotation by 12 degrees.

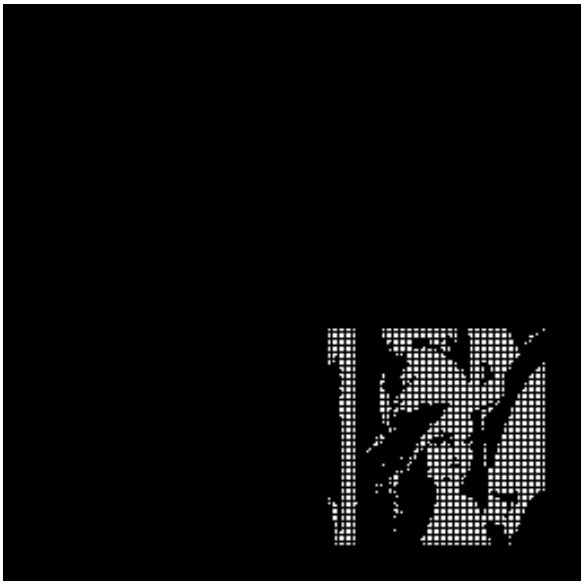

Figure 23: Image Scaling by Factor (1.5, 1.5).

Figure 24: Image Shearing by Factor (0.2, 0.1).

algorithms in term of execution time. In transformations of small image (64x64 pixels), our proposed algorithm outperforms all other standard lossless coding algorithms. Normally, standard lossless image coding algorithms do not support geometric transformations, hence images encoded using these algorithms have to be converted to raster or bitmap representation before transformations can be applied. Contrasting to our proposed algorithm, the transformations can be performed directly on the BDD representation of the image which eliminates time for conversion back and forth between bitmap and their native representation. As we have analyzed, the proposed algorithm runs slower as the size of the image increased. By increasing the test image from 64x64 pixels to 256x256 pixels which is 16 times enlarging of area, the execution time of the proposed algorithm increases by the factor of 3.4, 2.7, 3.9, 3.0 and 2.8 for image translation, orthogonal, arbitrary rotation, shearing and scaling, respectively. However, the performance is still comparable to the other standard lossless coding algorithms.

References

[1] Bryant, R.E., "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Comput.*, Vol. C35, No. 8, pp. 677-691, 8/86

[2] Bryant, R.E., "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," *ACM Computing Surveys*, Vol. 24, No. 3, pp. 293-318, 9/92

[3] Lursinsap C., Kanchanasut K., and Siriboon T., "Basic Binary Decision Diagram Operations for Image Processing," *Proc. 3rd Asian Computing Science Conf., Lecture Notes in Computer Science*, Springer-Verlag, pp. 368-370, 1997

[4] Sarkar, D., "Boolean function-based approach for encoding of binary images," *Pattern Recognition Letters*, 17(8), pp. 839-848, 1996

[5] Sarkar, D., Banerjee, S., and Chattopadhyay, S., "Translation and rotation of binary images encoded as minimized Boolean functions," *Pattern Recognition Letters*, 18 pp. 157-163, 1997

[6] Starkey, M., Bryant, R.E.,"Using Ordered Binary Decision Diagrams for Compressing Images and Image Sequences," *Technical Reports*, CMU-CS-95-105