Brief Contributions

On Spatial Orders and Location Codes

Leo J. Stocco, *Member*, *IEEE*, and Gunther Schrack

Abstract—Spatial orders such as the Morton (Z) order, U-order, or X-order have applications in matrix manipulation, graphic rendering, and data encryption. It is shown that these spatial orders are single examples of entire classes of spatial orders that can be defined in arbitrary numbers of dimensions and base values. Second, an algorithm is proposed that can be used to transform between these spatial orders and Cartesian coordinates. It is shown that the efficiency of the algorithm improves with a larger base value. By choosing a base value that corresponds to the available memory page size, the computational effort required to perform operations such as matrix multiplication can be optimized.

Index Terms—Spatial order, location code, matrix multiplication, encryption, dilation, quadtree, octree.

1 Introduction

Computer memory is sequential and, therefore, one-dimensional. To store a matrix that has two or more dimensions, the cells are ordered and stored sequentially, effectively transforming the matrix into a vector. Accessing a particular matrix cell involves computing the location code of that cell, which is the corresponding index into the vector that contains the matrix cell data. The default indexing scheme used by C and C++ compilers, among others, is to store each row sequentially. For this method, the location code i is computed in the following equation for a matrix with R elements per row and Cartesian coordinates specified by a row r and column c:

$$i = R \times r + c. \tag{1}$$

This method of indexing is often inefficient when performing operations on large matrices. When rows are stored sequentially, each cell in a column is separated by an entire row of data. Therefore, accessing columns of data when performing matrix multiplication, for example, may result in a great deal of memory paging.

This is a problem often encountered in computer graphics. Manipulating (i.e. translating, rotating, rendering, etc.) a graphic object on a large high-resolution screen requires one to index neighboring pixels of the extremely large array that comprises the screen buffer. Under the conventional indexing method (1), pixels that are neighbors vertically are stored an entire horizontal scan line apart from one another in memory.

The discrepancy between matrix and memory locality is overcome by combining a nonconventional algorithm such as matrix multiplication [1] and location code arithmetic [2] with a spatial order. Consider an $n \times m$ domain of integer coordinates. A curve that visits every coordinate point exactly once is called a space-filling curve. Peano [3] proposed such a curve in 1890, as did Hilbert [4] a year later and, more recently, Morton [5] and Schrack and Liu [6]. Usually, a square domain of dimension $2^n \times 2^n$ with nonnegative coordinates is chosen with the space-filling curve starting at the origin and ending in one of the coordinates $(2^n - 1, 0)$, $(2^n - 1, 2^n - 1)$, and $(0, 2^n - 1)$. An integer label is attached to each coordinate, called the location code, starting with 0 and advancing by one for each step, the last one being $2^{2n} - 1$.

Such a domain with location codes generated by a space-filling curve is called a spatial order. A space-filling curve or spatial order can be interpreted as a mapping that linearizes a 2D or higher dimensional space. Many spatial orders have been proposed, in particular the Hilbert order and the Morton or Z-order, both of which have specific properties that makes them useful in different applications in computing.

In Section 2, variations on three existing spatial orders are described, and they are extended in Section 3 to encompass orders with arbitrary and nonsquare dimensions but have easy-to-compute location codes. Section 4 presents a fast algorithm for interleaving integers that does not rely on table lookup. Interleaving is a necessary function for converting between Cartesian coordinates and a location code. Section 5 compares execution times for the computations presented in Section 4 for different orders of different base values. Last, concluding remarks are made in Section 6.

2 Existing Spatial Orders

When a spatial indexing strategy is used to store a matrix, neighboring cells are stored in close proximity to one another, thereby reducing the amount of memory paging that is required as an operation such as multiplication is carried out. For a 2×2 matrix, such as in Fig. 1, there are three possible orders in which one could traverse all four cells.

Starting from the reference cell, the second sequential cell can either be an adjacent cell or the opposite cell. If it is an adjacent cell, the third sequential cell can either be the other adjacent cell or the opposite cell. If the third cell is the other adjacent cell, the resulting order resembles a "Z" and is called the O_Z-order (see Fig. 2). On the other hand, if the third cell is the opposite cell, the O_U-order results. Finally, if the second cell is the opposite cell, the next two cells must be the two adjacent cells, and the O_X-order results [7]. Although other orders are possible, they can be obtained by reflecting one of the orders in Fig. 2 through a vertical, horizontal, and/or diagonal axis. For the 1-bit coordinate systems (i.e., $X, Y \in \{0, 1\}$) shown in Fig. 2, the Cartesian coordinates and corresponding location codes are shown in Table 1.

Separating each 2-bit location code into its most significant bit (MSB) and least significant bit (LSB), the location codes of the three orders are computed in the following equations, where $\oplus$ represents the XOR operation, "$\ll n$" represents a left shift by n bits, an overbar represents the two's complement, and the subscript X_b is used to denote that variable X contains b bits:

$$O_Z(X_1, Y_1) = [Y_1 \quad X_1] = Y_1 \ll 1 + X_1, \tag{2}$$

$$O_U(X_1, Y_1) = [Y_1 \quad X_1 \oplus Y_1] = Y_1 \ll 1 + X_1 \oplus Y_1, \tag{3}$$

$$O_X(X_1, Y_1) = [X_1 \oplus Y_1 \quad X_1] = (X_1 \oplus Y_1) \ll 1 + X_1. \tag{4}$$

Since left-shift operations commute, it can be shown that for any function $F(X_b, Y_b)$ that uses only left-shift, OR and XOR operations, such as (2)-(4), the following equation holds for any integer n and for variables that comprise any number of bits b:

$$F(X_b \ll n, Y_b \ll n) = F(X_b, Y_b) \ll n. \tag{5}$$

Note, however, that (5) does not hold if $F(X_b, Y_b)$ contains right-shift operations.

Consequently, shifting X_1 and Y_1 to the left by 2 bits is equivalent to shifting $O_Z(X_1, Y_1)$, $O_U(X_1, Y_1)$, and $O_X(X_1, Y_1)$ to the left by 2 bits and results in orders that are similar to those in Fig. 2 but where each subsequent cell is the reference of a corresponding 2×2 block of cells, as shown in Fig. 3.

• *The authors are with the Department of Electrical and Computer Engineering, University of British Columbia, 2332 Main Mall, Vancouver, BC V6T 1Z4, Canada. E-mail: {leos, schrack}@ece.ubc.ca.*

Manuscript received 18 July 2007; accepted 1 July 2008; published online 15 Sept. 2008.
Recommended for acceptance by J. Vetter.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2007-07-0325.
Digital Object Identifier no. 10.1109/TC.2008.171.

0018-9340/09/$25.00 © 2009 IEEE Published by the IEEE Computer Society
Authorized licensed use limited to: The University of British Columbia Library. Downloaded on March 16, 2009 at 15:47 from IEEE Xplore. Restrictions apply.

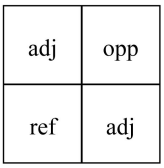

Fig. 1. 2 × 2 matrix.

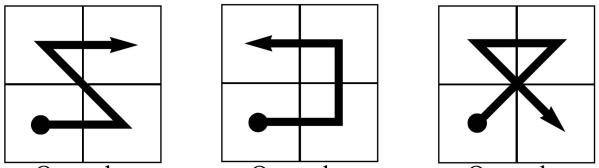

Fig. 2. Spatial orders with 1-bit coordinates.

Finally, ORing each of the 1-bit orders $F(X_1, Y_1)$ with its left-shifted counterpart

$$F(X_2, Y_2) = F(X_1, Y_1) \ll 2 + F(X_1, Y_1) \tag{6}$$

results in the 2-bit orders $F(X_2, Y_2)$ shown in Fig. 4. In Fig. 4, each of the original 2 × 2 1-bit orders from Fig. 2 is repeated in each of the 2 × 2 blocks visited by the left-shifted location codes from Fig. 3.

This order repeats in a fractallike manner each time the Cartesian coordinates are increased by 1 bit. Therefore, O_Z, O_U, and O_X (2)-(4) can be extended to any $2^n \times 2^n$ Cartesian coordinate system, with an arbitrarily large n, by substituting the left-shift and addition operations with the interleave function $INT(arg1, arg2)$, as shown in the following:

$$O_Z(X, Y) = INT(Y, X), \tag{7}$$

$$O_U(X, Y) = INT(Y, X \oplus Y), \tag{8}$$

$$O_X(X, Y) = INT(X \oplus Y, X). \tag{9}$$

The 2D interleave function $INT(arg1, arg2)$ accepts two integer arguments and returns an integer that is twice as long as the maximum bit length of its arguments. For example, the function $INT(Y_8, X_8)$ returns a 16-bit result when used to interleave two 8-bit integers X_8 and Y_8 whose individual bits are expressed in lowercase:

$$X_8 = x_7x_6x_5x_4x_3x_2x_1x_0, \tag{10}$$

$$Y_8 = y_7y_6y_5y_4y_3y_2y_1y_0, \tag{11}$$

TABLE 1
Location Codes of Three 1-Bit Orders

Cartesian			O_Z-order		O_U-order		O_X-order	
Y	X	$X \oplus Y$	Dec	Bin	Dec	Bin	Dec	Bin
0	0	0	0	00	0	00	0	00
0	1	1	1	01	1	01	3	11
1	0	1	2	10	3	11	2	10
1	1	0	3	11	2	10	1	01

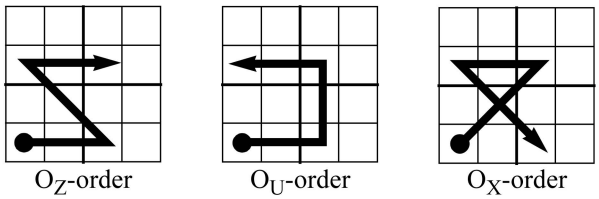

Fig. 3. One-bit spatial orders shifted left by 2 bits.

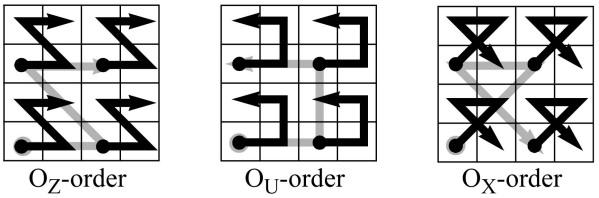

Fig. 4. Two-bit spatial orders.

$$INT(Y_8, X_8) = y_7x_7y_6x_6y_5x_5y_4x_4y_3x_3y_2x_2y_1x_1y_0x_0. \tag{12}$$

The interleave function is easily extended to any number of dimensions. In general, it returns an integer whose bit length is equal to the dimension multiplied by the maximum bit length of its arguments. For example, the 3D $INT(Z, Y, X)$ function returns a 12-bit result when used to interleave three 4-bit integers X_4, Y_4, and Z_4:

$$INT(Z_4, Y_4, X_4) = z_3y_3x_3z_2y_2x_2z_1y_1x_1z_0y_0x_0. \tag{13}$$

3 Generalized Spatial Orders

There are eight possible versions of each 2D order since each can be reflected through the vertical, horizontal, and diagonal axes. These reflections are accomplished by inverting the first or second argument or by interchanging the order of arguments. For example, all eight versions of the O_Z-order class are shown in Fig. 5, where the reference node is indicated by a dot. Since there are three classes of orders (O_Z, O_U, and O_X), there is a total of 24 possible 2D orders.

Each 2D order is derived from a bit operation that results in a unique ordering of the integers 0 to 3, of which there are $4! = 24$ possibilities, as expected. All 24 can be obtained by combining either column from any two of the three sections shown in Table 2 (Sections A, B, or C).

In three dimensions, there are three variables (X, Y, and Z), and the fundamental spatial element is a cube that has eight vertices. Therefore, all possible 3D orders can be derived from bit operations that result in a unique ordering of the integers 0 to 7,

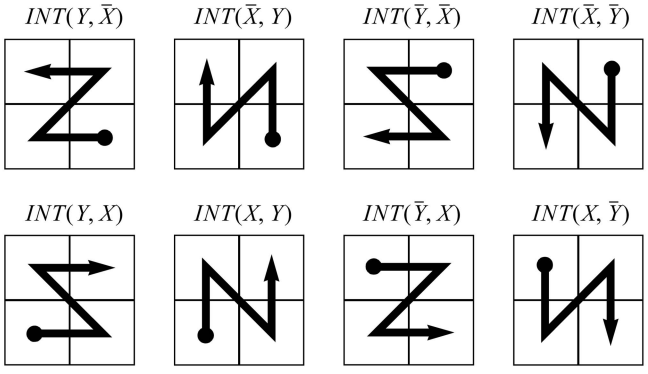

Fig. 5. Eight possible versions of the O_Z-order class.

Authorized licensed use limited to: The University of British Columbia Library. Downloaded on March 16, 2009 at 15:47 from IEEE Xplore. Restrictions apply.

TABLE 2
Variables for Computing a Location Code

Section A		Section B		Section C	
Y	$\bar{Y}$	X	$\bar{X}$	$X \oplus Y$	$\overline{X \oplus Y}$
0	1	0	1	0	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	0	1

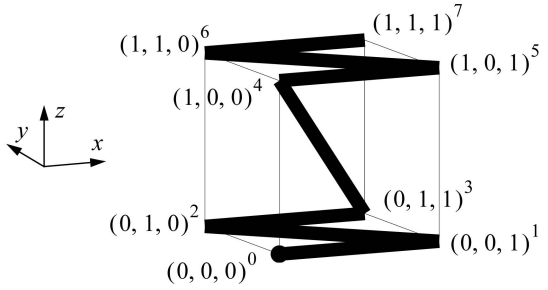

Fig. 6. $O_{01234567} = INT(Z, Y, X)$.

of which there are $8! = 40{,}320$ possibilities. Any one of these orders may be obtained by an interleaving operation performed on a combination of functions of the three Cartesian coordinates: X, Y, and Z. For example, $INT(Z, Y, X)$ results in the 3D order shown in Fig. 6. Although Fig. 6 is clearly composed of two Z patterns, many 3D orders are not so easily identifiable. Therefore, the order in Fig. 6 is classified as $O_{01234567}$, where the subscript refers to the eight-digit octal number that corresponds to the sequence in which the vertices of the spatial cube are visited. Each vertex is indexed by concatenating its binary Cartesian coordinates (Z, Y, X), where the associated octal number is shown as a superscript.

Similarly, it can be shown that the U/X order $(O_{02315674})$ shown in Fig. 7 can be obtained by the location code formula, $INT(Z, X \oplus Y, \bar{Z}(Y) + Z(\bar{X}))$.

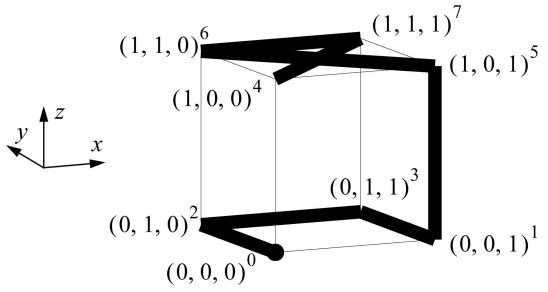

Fig. 7. $O_{02315674} = INT(Z, X \oplus Y, \bar{Z}(Y) + Z(\bar{X}))$.

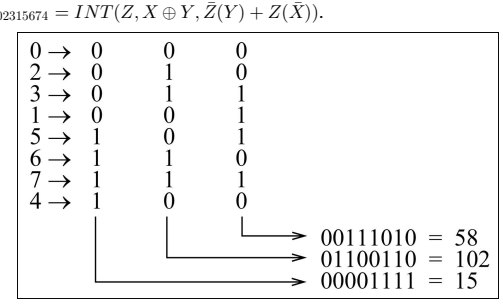

Fig. 8. Vertex table of $O_{02315674}$.

TABLE 3
Bit Functions for 3D Location Codes

Pattern	Function	Pattern	Function
00001111^{15}	Z	10000111^{135}	$\bar{Y}(\overline{X \oplus Z}) + Y(Z)$
00010111^{23}	$\overline{X \oplus Y}(X) + X \oplus Y(Z)$	10001011^{139}	$\bar{Y}(\bar{X}) + Y(Z)$
00011011^{27}	$\bar{X}(Z) + X(Y)$	10001101^{141}	$\bar{X}(\bar{Y}) + X(Z)$
00011101^{29}	$\bar{Y}(Z) + Y(X)$	10001110^{142}	$\overline{X \oplus Y}(\bar{X}) + X \oplus Y(Z)$
00011110^{30}	$\bar{Y}(Z) + Y(X \oplus Z)$	10010011^{147}	$\bar{Z}(\overline{X \oplus Y}) + Z(Y)$
00100111^{39}	$\bar{X}(Y) + X(Z)$	10010101^{149}	$\bar{Z}(\overline{X \oplus Y}) + Z(X)$
00101011^{43}	$\overline{X \oplus Y}(Z) + X \oplus Y(Y)$	10010110^{150}	$\overline{X \oplus Y \oplus Z}$
00101101^{45}	$\bar{Y}(Z) + Y(\overline{X \oplus Z})$	10011001^{153}	$\overline{X \oplus Y}$
00101110^{46}	$\bar{Y}(Z) + Y(\bar{X})$	10011010^{154}	$\bar{Z}(\overline{X \oplus Y}) + Z(\bar{X})$
00110011^{51}	Y	10011100^{156}	$\bar{Z}(\overline{X \oplus Y}) + Z(\bar{Y})$
00110101^{53}	$\bar{Z}(Y) + Z(X)$	10100011^{163}	$\bar{Z}(\bar{X}) + Z(Y)$
00110110^{54}	$\bar{Z}(Y) + Z(X \oplus Y)$	10100101^{165}	$\overline{X \oplus Z}$
00111001^{57}	$\bar{Z}(Y) + Z(\overline{X \oplus Y})$	10100110^{166}	$\bar{Z}(\bar{X}) + Z(X \oplus Y)$
00111010^{58}	$\bar{Z}(Y) + Z(\bar{X})$	10101001^{169}	$\bar{Z}(\bar{X}) + Z(\overline{X \oplus Y})$
00111100^{60}	$Y \oplus Z$	10101010^{170}	$\bar{X}$
01000111^{71}	$\bar{Y}(X) + Y(Z)$	10101100^{172}	$\bar{Z}(\bar{X}) + Z(\bar{Y})$
01001011^{75}	$\bar{Y}(X \oplus Z) + Y(Z)$	10110001^{177}	$\bar{X}(\bar{Z}) + X(Y)$
01001101^{77}	$\overline{X \oplus Y}(Z) + X \oplus Y(X)$	10110010^{178}	$\overline{X \oplus Y}(\bar{Z}) + X \oplus Y(Y)$
01001110^{78}	$\bar{X}(Z) + X(\bar{Y})$	10110100^{180}	$\bar{Y}(\overline{X \oplus Z}) + Y(\bar{Z})$
01010011^{83}	$\bar{Z}(X) + Z(Y)$	10111000^{184}	$\bar{Y}(\bar{X}) + Y(\bar{Z})$
01010101^{85}	X	11000011^{195}	$\overline{Y \oplus Z}$
01010110^{86}	$\bar{Z}(X) + Z(X \oplus Y)$	11000101^{197}	$\bar{Z}(\bar{Y}) + Z(X)$
01011001^{89}	$\bar{Z}(X) + Z(\overline{X \oplus Y})$	11000110^{198}	$\bar{Z}(\bar{Y}) + Z(X \oplus Y)$
01011010^{90}	$X \oplus Z$	11001001^{201}	$\bar{Z}(\bar{Y}) + Z(\overline{X \oplus Y})$
01011100^{92}	$\bar{Z}(X) + Z(\bar{Y})$	11001010^{202}	$\bar{Z}(\bar{Y}) + Z(\bar{X})$
01100011^{99}	$\bar{Z}(X \oplus Y) + Z(Y)$	11001100^{204}	$\bar{Y}$
01100101^{101}	$\bar{Z}(X \oplus Y) + Z(X)$	11010001^{209}	$\bar{Y}(\bar{Z}) + Y(X)$
01100110^{102}	$X \oplus Y$	11010010^{210}	$\bar{Y}(\bar{Z}) + Y(X \oplus Z)$
01101001^{105}	$X \oplus Y \oplus Z$	11010100^{212}	$\overline{X \oplus Y}(\bar{Z}) + X \oplus Y(X)$
01101010^{106}	$\bar{Z}(X \oplus Y) + Z(\bar{X})$	11011000^{216}	$\bar{X}(\bar{Y}) + X(\bar{Z})$
01101100^{108}	$\bar{Z}(X \oplus Y) + Z(\bar{Y})$	11100001^{225}	$\bar{Y}(\bar{Z}) + Y(\overline{X \oplus Z})$
01110001^{113}	$\overline{X \oplus Y}(Y) + X \oplus Y(\bar{Z})$	11100010^{226}	$\bar{Y}(\bar{Z}) + Y(\bar{X})$
01110010^{114}	$\bar{X}(Y) + X(\bar{Z})$	11100100^{228}	$\bar{X}(\bar{Z}) + X(\bar{Y})$
01110100^{116}	$\bar{Y}(X) + Y(\bar{Z})$	11101000^{232}	$\overline{X \oplus Y}(\bar{X}) + X \oplus Y(\bar{Z})$
01111000^{120}	$\bar{Y}(X \oplus Z) + Y(\bar{Z})$	11110000^{240}	$\bar{Z}$

To determine a location code formula that produces a particular spatial order, the following steps are taken. First, the desired order is determined. For example, the order shown in Fig. 7 is $O_{02315674}$.

Next, the octal vertex indices are entered into a table and converted to binary, and the decimal integer corresponding to each column is calculated, as shown in Fig. 8.

Note that since the vertex table starts from an ordering of the integers 0 through 7, each column will always produce an 8-bit integer with exactly four ones and four zeros. There are only 70 8-bit integers that satisfy these criteria, all of which

Authorized licensed use limited to: The University of British Columbia Library. Downloaded on March 16, 2009 at 15:47 from IEEE Xplore. Restrictions apply.

TABLE 4
Valid and Invalid 3D Location Code Formulas

Coordinate			Operation			Location Code		
Z	Y	X	$Y \oplus Z$	$X \oplus Y$	$X \oplus Z$	A	B	C
0	0	0	0	0	0	0	0	0
0	0	1	0	1	1	0	1	3
0	1	0	1	1	0	3	3	6
0	1	1	1	0	1	3	2	5
1	0	0	1	0	1	5	4	5
1	0	1	1	1	0	5	5	6
1	1	0	0	1	1	6	7	3
1	1	1	0	0	0	6	6	0

appear in Table 3. Each integer in the table is displayed next to a binary function that will produce the associated bit pattern as a function of the Cartesian coordinates X, Y, and Z. In this example, the MSB is 15, which corresponds to Z, the middle bit is 102, which corresponds to $X \oplus Y$, and the LSB is 58, which corresponds to $\bar{Z}(Y) + Z(\bar{X})$.

Finally, the three bit pattern functions are interleaved to produce a formula for computing the location code, which is $INT(Z, X \oplus Y, \bar{Z}(Y) + Z(\bar{X}))$ in this example.

Alternatively, if a computationally simple location code formula is preferred, one may be derived by choosing three bit patterns directly from Table 3. Note, however, that combining any three of the patterns from Table 3 will not necessarily produce a valid order. Since any pattern cannot be used more than once in a location code formula, the 70 patterns could be combined to produce $70 \times 69 \times 68 = 328{,}440$ unique formulas. However, as stated earlier, there are only 40,320 possible 3D spatial orders. Take formula A, for example:

$$A = INT(Z, Y, Y \oplus Z). \tag{14}$$

It does not result in an ordering of the integers 0 through 7 (see Table 4) because one of the Cartesian coordinates (X) does not appear anywhere in the formula. This is a necessary condition for a valid location code formula, but it is not a sufficient condition. Although formula B does produce a valid order (see Table 4)

$$B = INT(Z, Y, X \oplus Y), \tag{15}$$

formula C does not, even though formula C does include all three Cartesian coordinates, X, Y, and Z:

$$C = INT(Y \oplus Z, X \oplus Y, X \oplus Z). \tag{16}$$

Consequently, a location code formula that is developed in this way must be verified to ensure that it results in a valid spatial order before it can be used.

For the sake of consistency, a similar procedure can be used to develop 2D orders by consulting Table 5, which is a 2D version of Table 3 that provides functions for all six possible 4-bit integers that contain exactly two ones and two zeros.

The preceding two methods allow one to develop a spatial order that is tailored to the specific requirements of an application.

TABLE 5
Bit Functions for 2D Location Codes

Pattern	Function	Pattern	Function	Pattern	Function
0011^3	Y	0110^6	$X \oplus Y$	1010^{10}	$\bar{X}$
0101^5	X	1001^9	$\overline{X \oplus Y}$	1100^{12}	$\bar{Y}$

When performance is the primary concern such as in a 3D graphics application, one of the easy-to-compute examples on the top row of Fig. 9 might be a good choice. They include a Z/Z order, A, an X/X order, B, and a U/U order, C:

$$A)\ O_{01452367} = INT(Y, Z, X), \tag{17}$$

$$B)\ O_{05412763} = INT(X \oplus Y, Z, X), \tag{18}$$

$$C)\ O_{02641375} = INT(Y, X \oplus Y, Z). \tag{19}$$

If, on the other hand, it is more important that particular elements be neighbors such as in a matrix computation application, one of the orders from the middle row of Fig. 9 may be a better choice. They include an order that always goes to a nearest neighbor in a clockwise direction, D, and a counterclockwise direction, E, and one which always goes to a diagonal neighbor, F:

$$D)\ O_{01326457} = INT\big(Z, \overline{Z}(Y) + Z(\overline{X \oplus Y}), \bar{Z}(X \oplus Y) + Z(Y)\big), \tag{20}$$

$$E)\ O_{02315467} = INT\big(Z, \bar{Z}(X \oplus Y) + Z(Y), \overline{Z}(Y) + Z(\overline{X \oplus Y})\big), \tag{21}$$

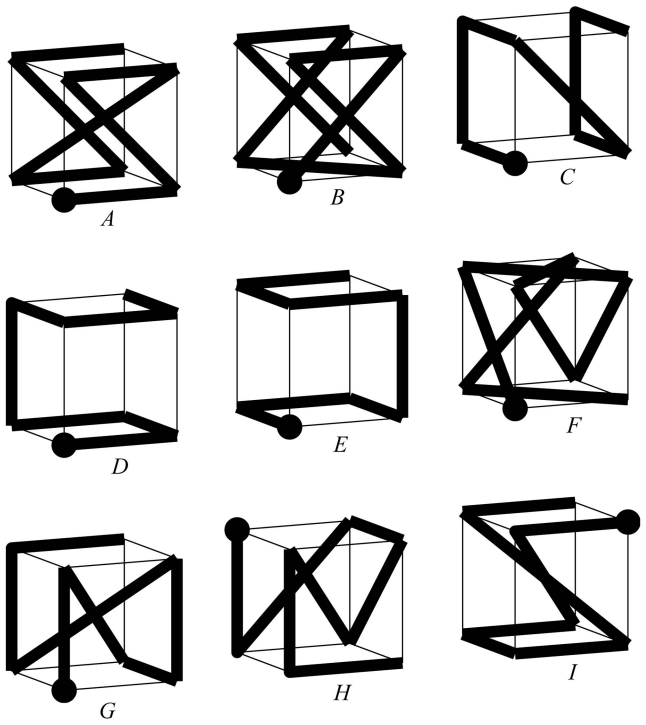

Fig. 9. Example spatial orders.

Authorized licensed use limited to: The University of British Columbia Library. Downloaded on March 16, 2009 at 15:47 from IEEE Xplore. Restrictions apply.

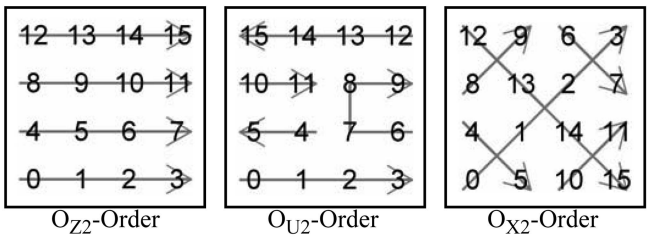

Fig. 10. Two-bit interleaved 2D orders.

$$F)\ O_{06534721} = INT\big(\bar{Z}(X \oplus Y) + Z(\overline{Y}), \overline{Z}(X) + Z(X \oplus Y), \overline{Z}(Y) + Z(X)\big). \tag{22}$$

Finally, if a more arbitrary pattern is preferred such as in a data encryption application, one of the examples from the bottom row (G through I) of Fig. 9 may be the best choice:

$$G)\ O_{04315267} = INT\big(\overline{Y}(X \oplus Z) + Y(Z), \overline{X}(Y) + X(Z), \overline{Z}(Y) + Z(\overline{X \oplus Y})\big), \tag{23}$$

$$H)\ O_{62753401} = INT\big(\overline{Y}(\overline{X \oplus Z}) + Y(\overline{Z}), \overline{X \oplus Y}(\overline{X}) + X \oplus Y(\overline{Z}), \overline{Z}(Y) + Z(\overline{X \oplus Y})\big), \tag{24}$$

$$I)\ O_{54320167} = INT(\overline{Y \oplus Z}, Y, \overline{X \oplus Z}). \tag{25}$$

Voxel ordering is a good way to scramble and descramble 3D data since the data is completely rearranged based on a key that identifies one of 40,320 possible orders.

All preceding 2D and 3D orders were all computed using base-1 interleaving functions (12), (13). These functions interleave individual bits of the operands. Other orders can be derived by using higher order interleaving functions. For example a base-2 interleave function interleaves the bits of its operands 2 bits at a time. For example, INT_2 is used to interleave X_8 (10) and Y_8 (11) as follows:

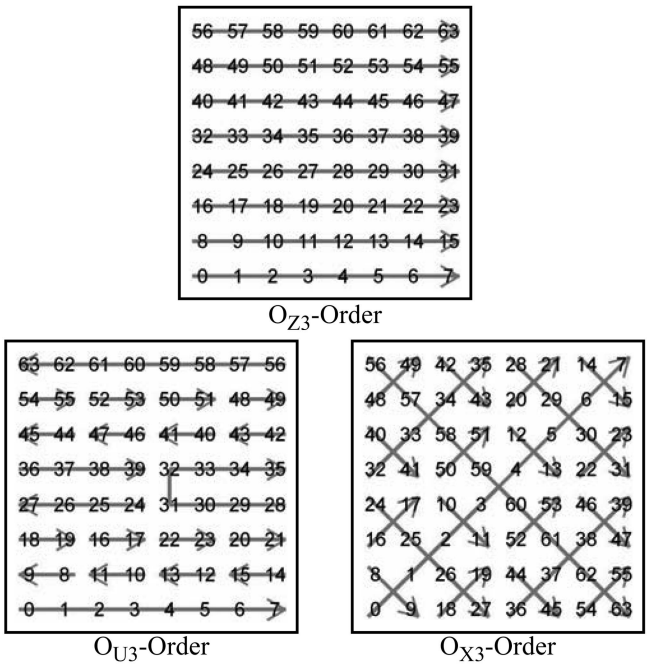

Fig. 11. Three-bit interleaved 2D orders.

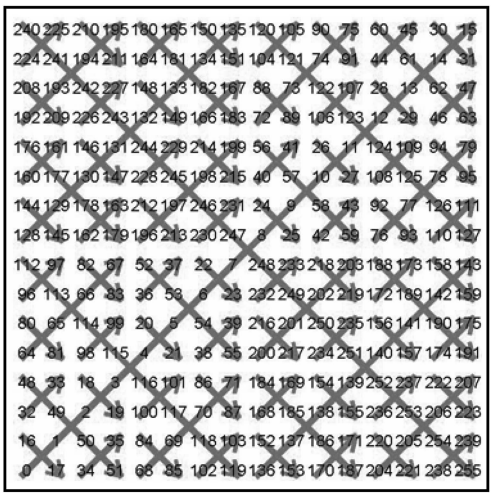

Fig. 12. Four-bit interleaved O_{X4}-order.

$$INT_2(Y_8, X_8) = y_7y_6x_7x_6y_5y_4x_5x_4y_3y_2x_3x_2y_1y_0x_1x_0. \tag{26}$$

Substituting the INT_2 function for the INT function in (7)-(9) results in the orders shown in Fig. 10.

Of course, the 2-bit 2D orders shown in Fig. 10 are easily extended to any number of bits and any number of dimensions. For example, 3-bit 2D interleaving results in the O_{Z3}-, O_{U3}-, and O_{X3}-orders shown in Fig. 11, and generating the X-order using 4-bit interleaving results in the O_{X4}-order shown in Fig. 12.

As with the 1-bit orders, the orders generated using higher order interleaving functions also repeat in a fractallike manner. For example, if any of the three-bit orders shown in Fig. 11 are used to map a 64×64 matrix, each 8×8 submatrix will be arranged in a sequence similar to that shown in Fig. 11 for the individual cells.

These orders are also easily extended to higher dimensions, nonsquare matrices, or any combination of the two, as shown respectively by the following equations:

$$INT_2(Z_4, Y_4, X_4) = z_3z_2y_3y_2x_3x_2z_1z_0y_1y_0x_1x_0, \tag{27}$$

$$INT_{12}(Y_3, X_6) = y_2x_5x_4y_1x_3x_2y_0x_1x_0, \tag{28}$$

$$INT_{213}(Z_4, Y_2, X_6) = z_3z_2y_1x_5x_4x_3z_1z_0y_0x_2x_1x_0. \tag{29}$$

Although nonsquare spatial orders have unique location codes, they do not repeat in the same fractallike manner that is observed with square matrices.

For example, the O_{Z12}-, O_{U12}-, and O_{X12}-orders are shown in Fig. 13. When these orders are used to map a 4×16 matrix, only the O_{Z12}-order repeats in a fractallike manner (see Fig. 14). In the case of the O_{U12}- and O_{X12}-orders, the 2×4 submatrices are not even exact duplicates of those shown in Fig. 13.

Similarly, nonsquare orders are easily extended to any number of dimensions and/or interleaving bit order. For example, the order computed by (29) is shown in Fig. 15. This order is referred to as $O_{213\text{-}01234567}$ since it is obtained by a 2-1-3 interleaving of the 01234567 order.

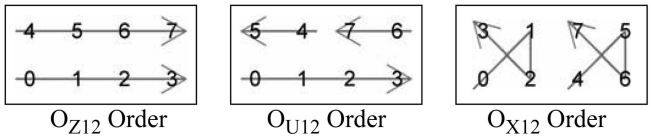

Fig. 13. Nonsquare 2D orders.

Authorized licensed use limited to: The University of British Columbia Library. Downloaded on March 16, 2009 at 15:47 from IEEE Xplore. Restrictions apply.

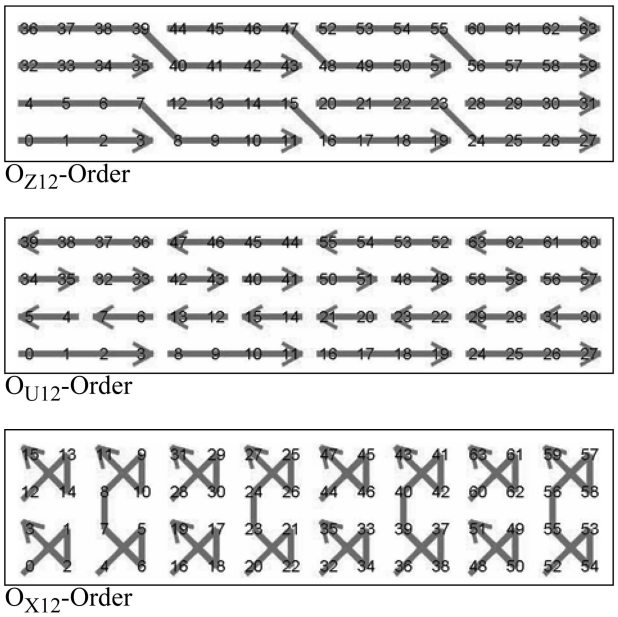

Fig. 14. O_{X12} order in a 4 × 16 matrix.

As suggested earlier, 3D orders can be difficult to display, such as $O_{213\text{-}01764532}$, which is shown in Fig. 16.

4 Interleaving Algorithms

The orders discussed in Section 3 require an interleaving of the bits of the Cartesian coordinates to arrive at a location code for the selected pattern. Interleaving is the process of dilating two or more integers (inserting groups of zeros between the significant bits), shifting them so the significant bits of each integer align with the inserted zeros in the other integers, and then ORing them together. For example the function $INT_1(X_4Y_4)$ is computed in the following equations, where *DIL* represents the dilate function:

$$DIL(X_4) = x_3 0 x_2 0 x_1 0 x_0, \tag{30}$$

$$DIL(Y_4) = y_3 0 y_2 0 y_1 0 y_0, \tag{31}$$

$$\begin{aligned} INT_1(Y_4, X_4) &= DIL(Y_4) \ll 1 + DIL(X_4) \\ &= y_3 x_3 y_2 x_2 y_1 x_1 y_0 x_0. \end{aligned} \tag{32}$$

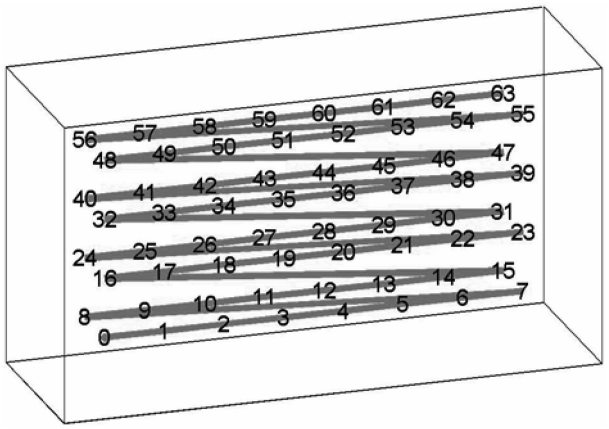

Fig. 15. $O_{213\text{-}01234567} = INT_{213}(Z, Y, X)$.

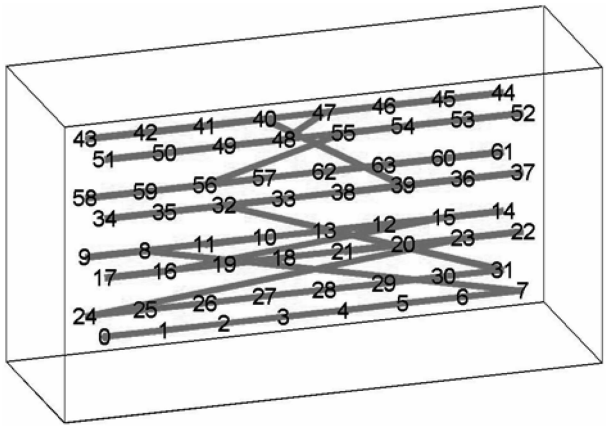

Fig. 16. $O_{213\text{-}01764532} = INT_{213}(Y \oplus Z, Y, X \oplus Y)$.

Similarly, converting from a location code to Cartesian coordinates requires integer contraction, the inverse of dilation. For example, a dilated 4-bit integer is converted back to a 4-bit integer in the following equation, where CTC represents the contract function, and □ represents a bit that may contain any value (i.e., don't care):

$$X_4 = CTC(x_3 \square x_2 \square x_1 \square x_0). \tag{33}$$

Historically, the preferred method of dilation has been by table lookup to avoid a cumbersome series of mask and shift operations that would be required to insert a zero between each bit. This, however, requires a large lookup table when the Cartesian coordinates are large. An integer with n bits has 2^n possible values and its associated dilated value is $2n-1$ bits long. The size of the required dilation table is given as follows:

$$\text{Dilate table size} = 2^n(2n-1) \text{ bits}. \tag{34}$$

A contraction table is even larger than a dilation table because it must contain 2^{2n-1} entries for undilated integers with n bits, most of which are duplicates of one another due to the abundance of "don't care" bits. The size of a contraction table for a dilated n-bit integer is given as follows:

$$\text{Contract table size} = 2^{2n-1}n \text{ bits}. \tag{35}$$

These table sizes grow exponentially with n, as shown in Fig. 17. Note that in Fig. 17, the y-axis is in units of 1 Kbyte for the dilation table and in units of 10 Mbytes for the contraction table. Also, note that the contract table can be reduced in size to that of a dilate table by performing a preoperation that interleaves the bits, thereby eliminating the "don't care" bits and halving the length of the integer to contract [8].

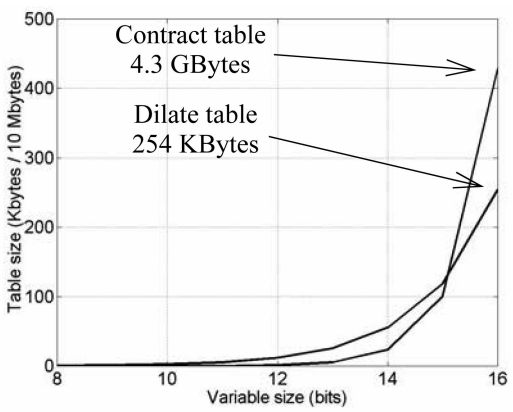

Fig. 17. Dilate and contract table sizes versus integer length.

Authorized licensed use limited to: The University of British Columbia Library. Downloaded on March 16, 2009 at 15:47 from IEEE Xplore. Restrictions apply.

Although memory is becoming an increasingly cheap commodity, it can be computationally expensive to access. As described in Section 1, one of the primary reasons spatial orders are used is to reduce memory paging. If a lookup table is used to calculate location codes, that table must occupy memory that could otherwise be used to store matrix cells. When the table is large, the table itself may have to be paged in and out of memory as different elements are accessed.

Another method for dilating integers, originally proposed by the authors in [9], does not use lookup tables. Instead, the bits are spread out efficiently using a series of left-shift, OR, and AND (mask) operations. The following equations demonstrate the step-by-step dilation of the 8-bit integer X_8:

$$X'_8 = (X_8 + (X_8 \ll 4)) \,\&\, \text{F0F} = x_7x_6x_5x_4 0000 x_3x_2x_1x_0, \tag{36}$$

$$\begin{aligned} X''_8 &= (X'_8 + (X'_8 \ll 2)) \,\&\, 3333 \\ &= x_7x_6 00 x_5x_4 00 x_3x_2 00 x_1x_0, \end{aligned} \tag{37}$$

$$\begin{aligned} X'''_8 &= (X''_8 + (X''_8 \ll 1)) \,\&\, 5555 \\ &= x_7 0 x_6 0 x_5 0 x_4 0 x_3 0 x_2 0 x_1 0 x_0. \end{aligned} \tag{38}$$

First, X_8 is divided into two groups of 4 bits that are separated by four zeros, X'_8 (36). Next, each of the two 4-bit groups is separated into two 2-bit groups X''_8 (37), and last, each 2-bit group is separated into two 1-bit groups X'''_8 (38), which is the dilation of X_8. Note that all masks are shown in hexadecimal format.

This procedure for dilating an integer is easily extended to integers with any number of bits n, where the number of required iterations is $log_2(n)$. For example, dilating a 16-bit integer would require one additional step, which would split the original integer into two 8-bit groups, separated by eight zeros, prior to continuing with (36)-(38). Since the preceding function dilates the integer into a repeating pattern of one zero followed by 1 bit, it is referred to as the DIL_{11} function. Consequently, DIL_{22} is identical to DIL_{11} but stops after (37) as is DIL_{44}, which stops after (36). Clearly, higher order dilation functions require fewer steps and, thus, reduced computational effort.

A similar method can be used for nonuniform dilations, as long as the number of zeros is greater than the number of bits. For example, $DIL_{32}(X_8)$ is accomplished by adjusting the shift operations and masks as follows:

$$\begin{aligned} X'_8 &= (X_8 + (X_8 \ll 6)) \,\&\, \text{3C0F} \\ &= x_7x_6x_5x_4 000000 x_3x_2x_1x_0, \end{aligned} \tag{39}$$

$$\begin{aligned} X''_8 &= (X'_8 + (X'_8 \ll 3)) \,\&\, \text{18C63} \\ &= x_7x_6 000 x_5x_4 000 x_3x_2 000 x_1x_0 \\ &= DIL_{32}(X_8). \end{aligned} \tag{40}$$

When the number of zeros is less than the number of bits, a less efficient dilation function must be used that first ORs together the most and least significant group of bits in its first iteration

$$X'_8 = (X_8 + (X_8 \ll 3)) \,\&\, 603 = x_7x_6 0000000 x_1x_0 \tag{41}$$

and then fills in an intermediate group of bits with each subsequent iteration:

$$X''_8 = X'_8 + (X_8 \,\&\, C) \ll 1 = x_7x_6 0000 x_3x_2 0 x_1x_0, \tag{42}$$

$$\begin{aligned} X'''_8 &= X''_8 + (X_8 \,\&\, 03) \ll 2 \\ &= x_7x_6 0 x_5x_4 0 x_3x_2 0 x_1x_0 \\ &= DIL_{12}(X_8). \end{aligned} \tag{43}$$

The computational effort required by this version of the dilate function grows linearly with the bit length of the operand, as opposed to logarithmically with the previous version.

A similar algorithm is used to implement the contract operation which reverses the procedure described above. For example, the three integers that were dilated in (36)-(43) are contracted as follows:

$$\begin{aligned} X_8 &= CTC_{11}(X'''_8), \\ X'''_8 &= x_7 0 x_6 0 x_5 0 x_4 0 x_3 0 x_2 0 x_1 0 x_0, \end{aligned} \tag{44}$$

$$\begin{aligned} X''_8 &= (X'''_8 + (X'''_8 \gg 1)) \,\&\, 3333 \\ &= x_7x_6 00 x_5x_4 00 x_3x_2 00 x_1x_0, \end{aligned} \tag{45}$$

$$\begin{aligned} X'_8 &= (X''_8 + (X''_8 \gg 2)) \,\&\, \text{F0F} \\ &= x_7x_6x_5x_4 0000 x_3x_2x_1x_0, \end{aligned} \tag{46}$$

$$\begin{aligned} X_8 &= (X'_8 + (X'_8 \gg 4)) \,\&\, \text{FF} \\ &= x_7x_6x_5x_4x_3x_2x_1x_0, \end{aligned} \tag{47}$$

$$\begin{aligned} X_8 &= CTC_{32}(X''_8), \\ X''_8 &= x_7x_6 000 x_5x_4 000 x_3x_2 000 x_1x_0, \end{aligned} \tag{48}$$

$$\begin{aligned} X'_8 &= (X''_8 + (X''_8 \gg 3)) \,\&\, \text{3C0F} \\ &= x_7x_6x_5x_4 000000 x_3x_2x_1x_0, \end{aligned} \tag{49}$$

$$X_8 = (X'_8 + (X'_8 \gg 6)) \,\&\, \text{FF} = x_7x_6x_5x_4x_3x_2x_1x_0, \tag{50}$$

$$\begin{aligned} X_8 &= CTC_{12}(X'''_8), \\ X'''_8 &= x_7x_6 0 x_5x_4 0 x_3x_2 0 x_1x_0, \end{aligned} \tag{51}$$

$$X''_8 = (X'''_8 \,\&\, 03) + (X'''_8 \,\&\, 18) \gg 1 = x_3x_2x_1x_0, \tag{52}$$

$$X'_8 = X''_8 + (X'''_8 \,\&\, \text{C0}) \gg 2 = x_5x_4x_3x_2x_1x_0, \tag{53}$$

$$X_8 = X'_8 + (X'''_8 \,\&\, 600) \gg 3 = x_7x_6 0000000 x_1x_0. \tag{54}$$

The dilate operation can be used by the interleave functions that compute location codes for the spatial orders described in Section 3. For example, the interleave operations referred to by (12), (13), and (26)-(29) are computed by the following equations:

$$INT_1(Y_8, X_8) = DIL_{11}(Y_8) \ll 1 + DIL_{11}(X_8), \tag{55}$$

$$\begin{aligned} INT_1(Z_4, Y_4, X_4) &= DIL_{21}(Z_4) \ll 2 \\ &\quad + DIL_{21}(Y_4) \ll 1 + DIL_{21}(X_4), \end{aligned} \tag{56}$$

$$INT_2(Y_8, X_8) = DIL_{22}(Y_8) \ll 2 + DIL_{22}(X_8), \tag{57}$$

$$\begin{aligned} INT_2(Z_4, Y_4, X_4) &= DIL_{42}(Z_4) \ll 4 \\ &\quad + DIL_{42}(Y_4) \ll 2 + DIL_{42}(X_4), \end{aligned} \tag{58}$$

$$INT_{12}(Y_3, X_6) = DIL_{21}(Y_3) \ll 2 + DIL_{12}(X_6), \tag{59}$$

Authorized licensed use limited to: The University of British Columbia Library. Downloaded on March 16, 2009 at 15:47 from IEEE Xplore. Restrictions apply.

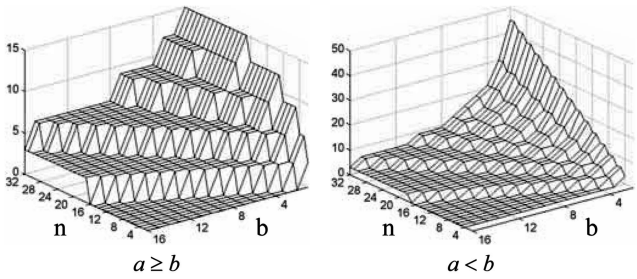

Fig. 18. Number of binary operations per integer dilation.

$$INT_{213}(Z_4, Y_2, X_6) = DIL_{42}(Z_4) \ll 4 + DIL_{51}(Y_2) \ll 2 + DIL_{33}(X_6). \quad (60)$$

Note that the interleave functions shown in (55)-(60) hold regardless of which dilation algorithm is chosen, the one proposed here or the more classic table lookup version.

The contract operation can be used in a similar fashion to convert from location codes back to Cartesian coordinates. For example, the Cartesian coordinates are recovered from the dilated integer computed in (60) by the following equations:

$$X_6 = CTC_{33}(INT_{213}(Z_4, Y_2, X_6)), \quad (61)$$

$$Y_2 = CTC_{51}(INT_{213}((Z_4, Y_2, X_6) \gg 3)), \quad (62)$$

$$Z_4 = CTC_{42}(INT_{213}((Z_4, Y_2, X_6) \gg 4)). \quad (63)$$

5 Performance

Each iteration of the dilate operation involves three binary operations: a left shift, an OR, and an AND. Since the number of iterations must be an integer value, it can be shown that the number of binary operations to dilate an n-bit integer can be computed by the following equations:

$$\#\mathrm{Op}(DIL_{ab}) = 3 \times ceil(\log_2(n/b)); \quad a \geq b, \quad (64)$$

$$\#\mathrm{Op}(DIL_{ab}) = 3 \times (ceil(n/b) - 1); \quad a < b. \quad (65)$$

Equations (64) and (65) are plotted for integers of up to $n = 32$ and bit groups of up to $b = 16$ in Fig. 18. Note that contraction requires the same number of binary operations as dilation except that one additional AND operation is required in the first iterations when $a < b$ (see (52)).

Computing a location code for a square 2×2 matrix that is mapped using a O_{Zb}-order requires two dilation operations, one left-shift operation, and one OR operation. Therefore, the total number of binary operations required is given by the following equation, which is plotted for an $n = 32$ integer in Fig. 19:

$$\#\mathrm{Op}(INT_b(Y_n, X_n)) = 6 \times ceil(\log_2(n/b)) + 2. \quad (66)$$

It is clear from Fig. 19 that the computational effort required to compute a location code depends heavily on the interleaving order b. For a 32-bit operand, it ranges from 8 to 32 binary operations for 16-bit and 1-bit interleaving, respectively. Therefore, there is a substantial performance incentive for choosing the spatial order that uses the highest order interleaving possible, achieved by selecting the spatial order that consumes as near as possible to one

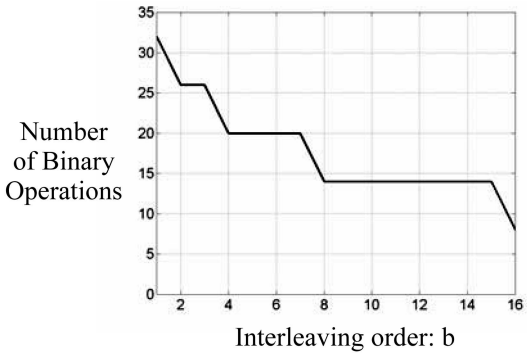

Fig. 19. Number of binary operations per location code.

page of memory. For a spatial order with k dimensions where each cell stores m-bytes of data, the amount of memory M used by the smallest complete submatrix is calculated as follows:

$$\text{Memory Consumption } (O_{d_1,d_2,\ldots,d_k}); \quad M = m \prod_{i=1}^{k} 2^{d_i}. \quad (67)$$

The memory consumption is independent of whether a Z-, U-, or X-order is chosen. The particular choice of order depends more on whether it is more important to have adjacent or diagonal entries in close proximity to one another.

For a square matrix where all d_i values are equal, (67) can be rearranged to solve for in terms of k, M, and m:

$$d_i = \frac{1}{k} \log_2 \left(\frac{M}{m} \right). \quad (68)$$

For example, for a system with a 256,000-page size, multiplying a 2D matrix of 4-byte values with 32-bit Cartesian coordinates, the optimum value of d_i is eight (from (68)). Therefore, the $\mathrm{O}_{\mathrm{Z88}}$-order would be a good choice since the associated 256×256 submatrices could be accommodated by a single page of memory and each location code would only require 14 binary operations to compute (from Fig. 19).

6 Conclusion

This paper investigates, in depth, the spatial orders of $2^n \times 2^n$ domains in two and three dimensions. If the operations involved with the generation of the orders are restricted to logical operations, three spatial order classes, O_{Z}, O_{U}, and O_{X}, emerge with eight orders each for each class of 2D spaces and 40,320 orders each for each class of 3D spaces.

Additional orders resulting from extending the interleaving operations to groups of more than 1 bit are also considered, including their ability to map nonsquare domains.

Underlying the definition and generation of spatial orders is the operation of interleaving coordinates, which in turn requires the dilation and contraction (undilation) of integers. Memory-preserving algorithms are also proposed to perform these operations for all of the spatial orders previously mentioned.

Finally, the computational burden of dilation and interleaving algorithms are compared, and it is described how performance can be maximized by choosing the optimal spatial order for a particular memory page size.

The contributions of this paper include a generalization of existing square spatial orders, a proposal for a new class of spatial orders that involves partially dilated Cartesian coordinates, a proposal for a set of algorithms to dilate and contract integers to compute the spatial orders previously mentioned, an analysis of the performance of said algorithms, and a proposal for a method of optimizing the choice of spatial order to a particular application.

Authorized licensed use limited to: The University of British Columbia Library. Downloaded on March 16, 2009 at 15:47 from IEEE Xplore. Restrictions apply.

REFERENCES

[1] V. Valsalam and A. Skjellum, "A Framework for High-Performance Matrix Multiplication Base on Hierarchical Abstractions, Algorithms and Optimized Low-Level Kernels," *Concurrency and Computation: Practice and Experience*, vol. 14, John Wiley & Sons, pp. 805-839, 2002.
[2] G. Schrack, "Finding Neighbors of Equal Size in Linear Quadtrees and Octrees in Constant Time," *CVGIP: Image Understanding*, vol. 55, no. 3, pp. 221-230, 1992.
[3] G. Peano, "Sur une Courbe, Qui Remplit Toute une Aire Plaine," *Math. Ann.*, vol. 36, pp. 157-160, 1890.
[4] D. Hilbert, "Über die Stetige Abbildung einer Linie auf ein Flächenstück," *Math. Ann.*, vol. 38, pp. 459-460, 1891.
[5] G.M. Morton, *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*, technical report, IBM, Mar. 1966.
[6] G. Schrack and X. Liu, "The Spatial U-Order and Some of Its Mathematical Characteristics," *Proc. IEEE Pacific Rim Conf. Comm., Computers and Signal Processing (PACRIM '95)*, pp. 416-419, 1995.
[7] X. Liu and G.F. Schrack, "A New Ordering Strategy Applied to Spatial Data Processing," *Int'l J. Geographical Information Science*, vol. 12, no. 1, pp. 3-22, 1988.
[8] F.C. Holroyd and D.C. Mason, "Efficient Linear Quadtree Construction Algorithm," *Image and Vision Computing*, vol. 8, no. 3, pp. 218-224, 1990.
[9] L. Stocco and G. Schrack, "Integer Dilation and Contraction for Quadtrees and Octrees," *Proc. IEEE Pacific Rim Conf. Comm., Computers and Signal Processing (PACRIM '95)*, pp. 426-428, 1995.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.

Authorized licensed use limited to: The University of British Columbia Library. Downloaded on March 16, 2009 at 15:47 from IEEE Xplore. Restrictions apply.