The Millar Polyhedron and its use in the Construction of Octrees

R. J. MILLAR,* M. E. C. HULL* AND J. H. FRAZER†

*Department of Computing Science, University of Ulster, Newtownabbey, Co. Antrim BT37 0QB, UK

†Department of Design in Industry, University of Ulster, York Street, Belfast BT15 1ED, UK

This paper presents the Millar Polyhedron, a cubic spiral, by mathematical definition. The defined ordering is given by look-up tables and it is shown how these have a simple implementation on a computer system. A link between this ordering and an octree data structure is then established and it is demonstrated how the cubic spiral can be used to climb over voxel-based input data in the construction of an octree—directly yielding the octree nodes due to its particular ordering. The paper concludes with the suggested transputer implementation in occam 2 with some practical results.

Received August 1991, revised October 1991

1. INTRODUCTION

Octrees [1] are often constructed from model data in a bottom-up manner [11]. Although these techniques only reference each raw voxel value once, they involve the building of (temporary) nodes, some of which will subsequently be coalesced into a higher-order octree node. This will take place if all eight children of a node have the same value. At this point, the voxel values are being indirectly inspected again and again.

This paper addresses this problem and introduces the Millar Polyhedron as a linear octree construction technique with order $o(n)$, where n is the number of voxels.

The Millar Polyhedron is presented in this paper as a cubic spiral—a 3D space-filling curve [2]. Its recursive definition is given mathematically before an efficient implementation as a series of look-up tables is presented. It will be explained how these tables provide a mapping between a node within the space-filling curve and a (x, y, z) co-ordinate in the cubic modelling space. Examples will demonstrate the use of these tables.

It will be established that the ordering of the Millar Polyhedron exactly matches that of a depth-first octree traversal. Thus, using the Millar Polyhedron to direct the path of a walk around the raw voxel data will reference each voxel once and once only directly from the voxel values as they are encountered. No temporary or intermediate nodes are required.

Finally, the implementation of the recursive definition in an iterative manner makes each application of the table mapping an independent calculation. Thus, the definition is inherently concurrent and a suggested transputer [5] implementation is presented. Results from this implementation are included to support this conclusion and to indicate the time-complexity and linear nature of the algorithm.

2. THE MILLAR POLYHEDRON

The Millar Polyhedron is a 3D space-filling curve. Cole [1] has discussed the 2D Hilbert polygon [2] and related it to quadtrees [10]. The Millar Polyhedron is essentially a cubic spiral. Given a 3D cubic lattice of points—say the centre points of the voxels of a modelling space—the curve is defined to start at one corner of the lattice and climb around every point to emerge at a different corner. Each point is visited only once by the curve. A mathematical definition follows before the curve is applied to the construction of octrees.

2.1. Definition of the Millar Polyhedron

Define the zero-order polyhedron, $\mathbf{M_0}$, as a single point in space.

In general, the ith order polyhedron, $\mathbf{M_i}$ $(i>0)$, is formed recursively from the coupling of eight copies of $\mathbf{M_{i-1}}$. Each of these copies will have a different transformation applied to it before they are linked together in a specified order into a cubic form. Figure 1 illustrates $\mathbf{M_0}$, $\mathbf{M_1}$ and $\mathbf{M_2}$.

The transformations which are applied to the eight copies of the lower order polyhedron $\mathbf{M_{i-1}}$ in the formation of a polyhedron $\mathbf{M_i}$ are illustrated in Figure 2.

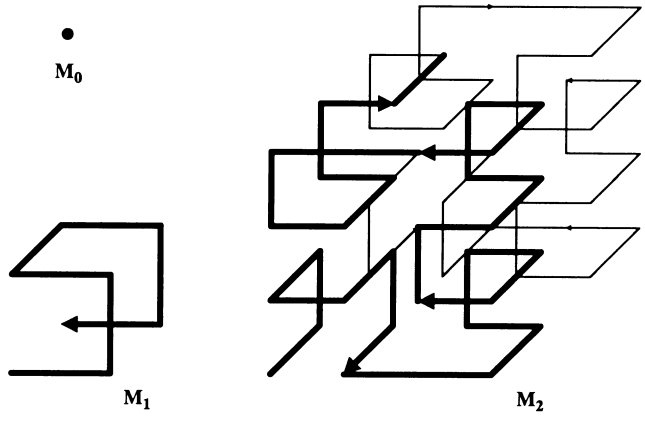

FIGURE 1. The Millar Polyhedra $\mathbf{M_0}$, $\mathbf{M_1}$ and $\mathbf{M_2}$.

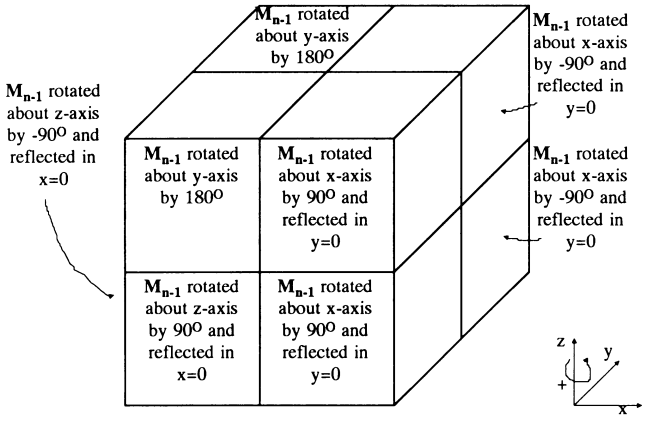

FIGURE 2. Transformations on $\mathbf{M}_{i-1}$ to form $\mathbf{M}_i$.

These eight transformed versions of $\mathbf{M}_{i-1}$ are linked together to form the Millar Polyhedron $\mathbf{M}_i$ in the order given by Figure 3.

2.2. Mapping of node number → (*x*, *y*, *z*) coordinate

To relate this voxel walk-round ordering to a cubic modelling space, it is necessary to establish a mapping from the node number (i.e. the value of a count of the number of voxels already visited) to a (*x*, *y*, *z*) coordinate—which typically would allow a datum value corresponding to this coordinate to be extracted from a database. These data values can be voxels belonging to any application of spatial enumeration modelling. For example, the authors have demonstrated the use of this technique in modelling the data for X-ray absorption at regular points within a patient as generated by a computed tomography scanner [3, 4, 8]. Again, the mapping is presented here in isolation from the application of the technique.

Suppose the Millar Polyhedron $\mathbf{M}_i$ is traversing a set of 2^{3i} nodes which are arranged in a cubic form of side 2^i nodes.

An efficient implementation of the mapping is based on look-up tables which have the advantage of being able to be pre-calculated for any Polyhedron $\mathbf{M}_i$. These will find the (*x*, *y*, *z*) coordinate for the node number **n** from 2^{3i} nodes numbered as 0 to $2^{3i}-1$ in exactly **i** table accesses.

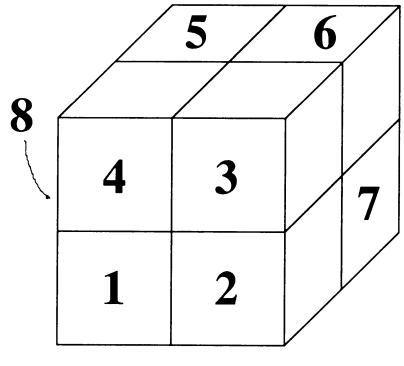

FIGURE 3. Order in which $\mathbf{M}_i$ traverses $\mathbf{M}_{i-1}$.

The mapping requires a total of 24 tables. Only tables (1) and (4) are presented in Figure 4 for the purposes of illustration. The complete set of tables are appended.

These look-up tables are used in the following manner.

1. Express the node number **n** in binary form:
   $\mathbf{a_1a_2a_3}\ldots\ \ldots\mathbf{a_{3i}}$
2. Group the digits in triplets $\mathbf{a_1a_2a_3}$, $\mathbf{a_4a_5a_6}$, etc.
3. Look up the first triplet, $\mathbf{a_1a_2a_3}$, in table (1). This will yield $\mathbf{x_1}$, $\mathbf{y_1}$ and $\mathbf{z_1}$—the first digits of the binary form of the coordinates (*x*, *y*, *z*). The table look-up will also yield the number of the next table which should be accessed for the next triplet, if any. Note that all leading zeros must be included in the binary form of the node number.

With the defined ordering of $\mathbf{M}_1$, table (1) can first be formed. This table relates the eight nodes of the *M*, polyhedron, numbered as 0 to 7—the digit triple in the table, to their (*x*, *y*, *z*) position in space. For this simple polyhedron, each of *x*, *y* and *z* can only be 0 or 1. Tables (2) to (6) represent $\mathbf{M}_1$ under the five distinct transformations outlined in Figure 2 for the formation of $\mathbf{M}_i$ from $\mathbf{M}_{i-1}$. For example, table (4) is the rotation of $\mathbf{M}_i$ by 180° about the *Y*-axis. The other tables are formed by the composition of these transformations and it is

	DIGIT TRIPLET	x	y	z	NEXT TABLE
TABLE (1)	000	0	0	0	2
	001	1	0	0	3
	010	1	0	1	3
	011	0	0	1	4
	100	0	1	1	4
	101	1	1	1	5
	110	1	1	0	5
	111	0	1	0	6
TABLE (4)	000	1	0	1	15
	001	0	0	1	16
	010	0	0	0	16
	011	1	0	0	1
	100	1	1	0	1
	101	0	1	0	12
	110	0	1	1	12
	111	1	1	1	8

FIGURE 4. Typical look-up tables for node number → (*x*, *y*, *z*) coordinates

found that they form a closed set of 24 distinct transformations.

Consideration of the orientations of the Millar Polyhedron $\mathbf{M_1}$, leads to the conclusion that this set is complete. The first node can be at any of eight positions (the eight corner nodes of the cube), the second node at any of three positions (the nodes which are adjacent to the first node), similarly the third node at eight of two positions and then the other nodes are dictated by the polyhedron ordering. This might suggest 48 orientations, but considering that, for example, the orientations of Figure 5 are not differentiated, then each orientation has one exact match traversed in the opposite direction.

Thus, we obtain

$$(8 \times 3 \times 2)/2 = 24$$

distinct (unordered) orientations.

2.3. Mapping of (x, y, z) coordinate → node number

Depending on the application, it may be necessary to reverse the Millar Polyhedron walk-round. For example, an aircraft using a terrain model for navigation might need to know what lies ahead at coordinates (a, b, c). Obtaining this information requires an interrogation of the model by node number. Thus, the inverse mapping from (x, y, z) coordinates to node number is presented for completeness.

Once again this mapping is achieved by a set of 24 look-up tables. These tables are the converse of those presented in Section 2.2 and tables (1)′ and (4)′ are given in Figure 6 for use in the example which follows.

The inverse mapping is performed in a similar manner to that of Section 2.2. A xyz-triplet is formed by taking the most significant bit of each of the x, y and z ordinates expressed in binary. This is used as a key to table (1)′ to yield the three most significant bits of the node number in binary form. Again, the table which should be used next for any remaining bits is quoted.

3. OCTREE CONSTRUCTION USING THE MILLAR POLYHEDRON

It has been observed in Section 2.1 that the Millar Polyhedron visits nodes in a spiral manner around a cube. If this cube was divided into eight equal-sized sub-cuboids, then it would be found that each of these sub-cubes also has its nodes visited in the same fashion. This would hold true if the sub-division was continued until a point was reached where no further sub-division was possible, i.e. the voxel level.

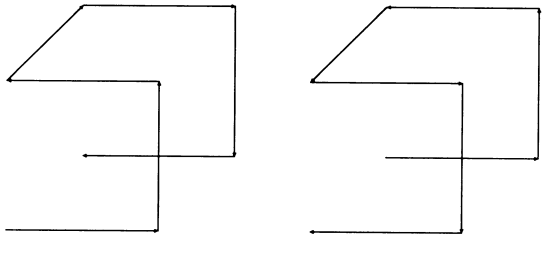

FIGURE 5. Identical orientations when unordered.

	xyz TRIPLET	DIGIT TRIPLET	NEXT TABLE
Table (1)′	000	000	2
	001	011	4
	010	111	6
	011	100	4
	100	001	3
	101	010	3
	110	110	5
	111	101	5
Table (4)′	000	010	16
	001	001	16
	010	101	12
	011	110	12
	100	011	1
	101	000	15
	110	100	1
	111	111	8

FIGURE 6. Sample tables for the mapping of (x, y, z) coordinates → node number.

This process of sub-division is exactly the same method as is used in the construction of a classical octree [9]. If when traversing the voxels a count is kept of the number of voxels having the same datum value (be it colour or X-ray absorption), then this will collapse identical voxels inside a cubic volume into a single data value and count. The result of such an operation is that the nodes of an octree can be directly constructed from the output of the walk over the voxels using the path dictated by a Millar Polyhedron.

A node counter is initialized to zero. The node number to (x, y, z) mapping is applied and the voxel value at position (x, y, z) is queried from the model database. If the voxel value is the same as the previous node, a count is incremented, otherwise the previous count and voxel value is output and a new count started. The node number is incremented and the process repeated. An algorithm presentation is given in Section 4. The node number to (x, y, z) mapping is the key aspect of this algorithm. To provide a clearer understanding of the mapping, two examples follow:

Example 1. What is the (x, y, z) coordinate for node 30 in a $\mathbf{M_2}$ polyhedron?

$$\text{Node } 30_{10} = \text{Node } 011110_2$$

First triplet is 011 and accessing table (1) yields the sequence 0,0,1 for x, y, z respectively with the information that table (4) should be accessed next.

Complete calculation:

Node # Digit Triples	Use Table #	x	y	z
0 1 1	(1)	0	0	1
1 1 0	(4)	0	1	1
(Not needed)	(12)			

Reading out (x, y, z) by column yields (00,01,11), i.e. (0,1,3). Figure 7 illustrates an $\mathbf{M_2}$ system with axes x, y, z. The character **X** marks the node number 30 (with nodes numbered as 0 to 63). It can be seen that (0,1,3) is the correct co-ordinate.

Example 2. What is the node number in a $\mathbf{M_2}$ polyhedron for (0,1,3)?

TRIPLET x	y	z	#	TABLE TRIPLET	DIGIT
0	0	1		(1)'	011
0	1	1		(4)'	110

Node number is 011110_2 or 30 (denary), which is correct by the previous example.

Thus, it can be seen how the look-up tables provide an iterative definition of the path of the Millar Polyhedron.

A typical data stream produced by the Millar Polyhedron walk-round when used with an 'identical node' count as a method of data compression might be

16, 23, 16, 25, 7, 24, 25, 23, 64, 10, 2, 19, ...

The entries in the data stream are essentially data tuples of (number of voxels, colour number), (number of voxels, colour number), ..., i.e. there are 16 voxels of colour 23, followed by 16 voxels of colour 25, etc. The first 16 voxels in this example would be the first two cubes of voxels walked over by the Millar Polyhdron ordering.

In octree construction, a number of voxels is sought which is a member of the 'cubic' numbers (1, 8, 64, 512, ...) and which are of constant colour. This counting of the voxels is from the start of the walk over. Thus, taking the example data stream above, it can be seen that the 16 voxels of colour 23 must be sub-divided to the next lowest cubic number (8) as two cubes of 8 voxels of colour 23. Similarly for the next 16 voxels. The seven voxels of colour 24 would have to form seven 'cubes' of one voxel of colour 24. The 25 voxels of colour 23 forms a total of 64 voxels from the start of the encoding. Thus, the section of octree formed would be as shown in Figure 8.

In summary, the data stream above is modified to read as:

8, 23, 8, 23, 8, 25, 8, 25, 1, 7, 1, 7
1, 7, 1, 7, 1, 7, 1, 7, 1, 7, 1, 23
8, 23, 8, 23, 8, 23, 64, 10, 1, 19, 1, 19, ...

Note also from this example how the next block of voxels had to give up a single voxel to complete a sub-divided cuboid. However, it does serve to illustrate how the octree can be formed in a bottom-up manner by a single, non-backtracking pass over the data stream. It should be appreciated that the division of the Millar Polyhedron directed walk-round data into cubic numbers can be achieved by a modification in the way in which the algorithm counts its similar voxels. It is not necessary to pass over the walk-round data as done here to achieve this end.

A further level of data compression can be achieved for a monochrome display system as it is not necessary to record the colours. All that is required is a convention that the first block of voxels are of foreground colour. Then the next block must be of background colour, the next of foreground colour, etc. Should the first block need to be of background colour then simply record a zero-sized block of foreground colour at the start.

4. A CONCURRENT IMPLEMENTATION

The application of the node number to (x, y, z) co-ordinate mapping for any node is a completely independ-

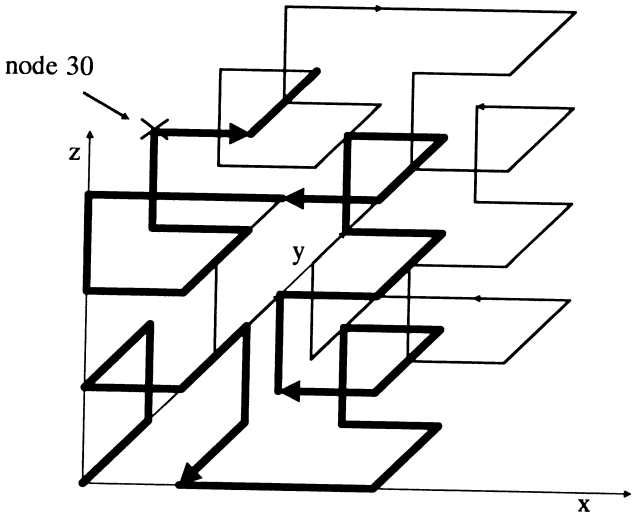

FIGURE 7. The position of node 30 in a $\mathbf{M_2}$ polyhedron.

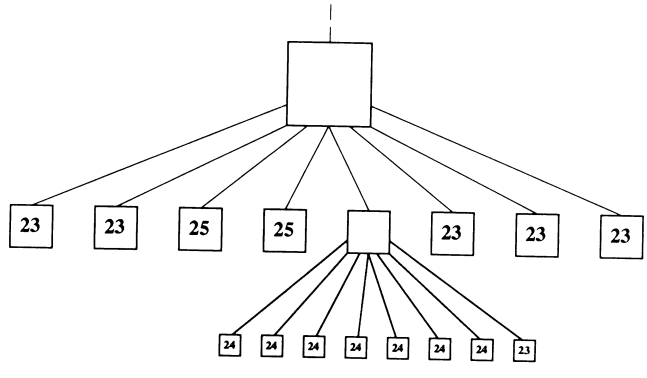

FIGURE 8. A section of octree formed from a data stream.

ent calculation. Thus, the processing for a number of nodes can be performed concurrently.

Since the Millar Polyhedron is a linear curve, visiting (say) **n** nodes, concurrent octree construction can be achieved by allowing the first processor to walk around nodes 0 to (*n*/(number of processors))−1, the second processor to walk around nodes (*n*/(number of processors)) to (2**n*/(number of processors))−1, and so on. It is then only necessary to examine the last and first tuple at a join of two partial curves to determine if they have the same voxel value and thus to coalesce them to one tuple.

An outline (sequential) implementation in occam 2 [6] of the octree construction algorithm is as follows. It should be noted that this routine does not split counts into any form of cubic number.

```
PROCEDURE walkround (CHAN OF INT; INT from.walkround, VAL INT first.node, last.node)
  SEQ
    x, y, z:=node.no.to.xyz(first.node)
    datum:=from.database(x, y, z)—input data
    last.datum:=datum
    count:=1
    SEQ node.no=first.node+1 FOR last.node—first.node
      SEQ
        x, y, z:=node.no.to.xyz(node.no)
        datum:=from.database(x, y, z)
        IF
          (datum=last.datum)
            count:=count+1
          (datum <> last.datum)
            SEQ
              from.walkround ! count; last.datum
              count:=1
              last.datum:=datum
  from.walkround ! count; last.datum
```

A parallel form of this algorithm can be easily obtained:

```
VAL INT no.processors IS 4:
VAL INT no.nodes.each IS no.nodes/no.processors:
[no.processors]CHAN OF INT; INT data.stream:
PAR
  data.stream.joiner(data.stream)
  PAR i=0 FOR no.processors
    walkround(data.stream[i], i* no.nodes.each, ((i+1) *no.nodes.each)−1)
```

The only assumptions in this method are that the total number of nodes is exactly divisible by the number of processors, which is often the case (and if it is not then this is easily overcome by a small modification to the last.node value on the final processor), and that the input database is available to all processors.

This outline occam 2 code also illustrates the flexible nature of this approach. When the number of processors needs to be increased then only a simple change of a constant is required.

Actual results obtained from such an implementation confirm the concurrent nature of the algorithm and indeed the processor-bound nature of the task of octree construction from voxel-based input data. The table in Figure 9 quotes sample timings for completing a Millar Polyhedron directed walk over voxel data in a cubic modelling space of side 256 voxels using different numbers of T414 transputers at 10 MHz link speed. It indicates a linear increase in effective processor power against the number of transputers in use. It should be kept in mind, however, that the walk over the input data need only occur once for any given set of data. Thereafter, the encoded data can be stored for future reference.

Further concurrency can be achieved by pipelining the table look-up operations. In processing 2^{3i} nodes, each node will require exactly **i** table look-ups. Each of these **i** look-ups is dependent on the previous table for

Number of Transputers	Time taken (minutes)
1	120
4	30
8	15
16	8

FIGURE 9. Typical times for a complete Millar Polyhedron walk over 256^3 voxels of data.

a 'next table' number, so no concurrency is available there. However, if **i** processors are in a pipeline, the first processor can perform the first table look-up for the second node while the second processor performs the second table look-up for the first node.

5. CONCLUSIONS

The Millar Polyhedron presents an efficient method of climbing around voxel-based data in the construction of a model which is to be held in an octree data structure. It eliminates the need for temporary nodes and only references each voxel once. It is a linear algorithm. The look-up tables defined in the paper are an effective way to specify the ordering and have the value of a fixed speed of operation which is constant whatever the current position in the ordering or whatever the level of detail in the model. The tables need only be specified once and this can be performed external to any application package and the tables written into the application code.

The form of the ordering and the data it produces are ideally suited to the construction of octrees, being identical to that of a linear octree, i.e. a depth-first octree traversal. An algorithm to do this has been presented and implemented in a parallel processing system.

Analysis of the technique and the results obtained from the transputer/occam 2 implementation have demonstrated how the Millar Polyhedron walk-round is inherently concurrent. The use of the transputers has exploited this concurrency and provided a flexible and an expandable system. Concurrent processing is vital to deal with the processor bound nature of the task of octree construction.

REFERENCES

[1] A. J. Cole, Compaction techniques for raster scan graphics using space-filling curves. *The Computer Journal*, **30**, pp. 87–92 (1987).

[2] D. Hilbert, Ueber stetige abildung einer linie auf ein flachenstuck. *Mathematische Annalen*, **38**, pp. 459–460 (1981).

[3] G. N. Hounsfield, Computerised transverse axial scanning (tomography). *British Journal of Radiology*, **46**, pp. 1016–1022 (1973).

[4] M. E. C. Hull, J. H. Frazer and R. J. Millar, Octree-based modelling of computed-tomography images. *IEE Proceedings Part I: Communications, Speech and Vision*, **137**, pp. 118–122 (1990).

[5] INMOS Limited, *The Transputer Reference Manual*, Prentice-Hall, New York (1988).

[6] INMOS Limited, *occam 2 Reference Manual*. Prentice-Hall, New York (1988).

[7] D. Meagher, *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects*. IPL-TR-80-111, Rensselaer Polytechnic Institute, Troy, NY (1980).

[8] R. J. Millar, *Octree Construction for Imaging*. DPhil Thesis, University of Ulster (1989).

[9] M. A. Oliver and N. E. Wiseman, Operations on quadtree encoded images. *The Computer Journal*, **26**, pp. 83–91 (1983).

[10] H. Samet, The quadtree and related hierarchical data structures. *ACM Computing Surveys*, **16**, pp. 187–260 (1984).

[11] K. Yamaguchi, *et al.* Octree-related data structures and algorithms. *IEEE Computer Graphics Applications*, **4**(1), pp. 53–59 (1984).

APPENDIX: COMPLETE LOOK-UP TABLES FOR NODE NUMBER TO (x,y,z) MAPPING

	Digit Triple	x	y	z	Next Table
Table (1)	0 0 0	0	0	0	2
	0 0 1	1	0	0	3
	0 1 0	1	0	1	3
	0 1 1	0	0	1	4
	1 0 0	0	1	1	4
	1 0 1	1	1	1	5
	1 1 0	1	1	0	5
	1 1 1	0	1	0	6
Table (2)	0 0 0	0	0	0	1
	0 0 1	0	1	0	7
	0 1 0	0	1	1	7
	0 1 1	0	0	1	8
	1 0 0	1	0	1	8
	1 0 1	1	1	1	9
	1 1 0	1	1	0	9
	1 1 1	1	0	0	10
Table (3)	0 0 0	0	0	0	11
	0 0 1	1	0	0	1
	0 1 0	1	1	0	1
	0 1 1	0	1	0	12
	1 0 0	0	1	1	12
	1 0 1	1	1	1	13
	1 1 0	1	0	1	13
	1 1 1	0	0	1	14
Table (4)	0 0 0	1	0	1	15
	0 0 1	0	0	1	16
	0 1 0	0	0	0	16
	0 1 1	1	0	0	1
	1 0 0	1	1	0	1
	1 0 1	0	1	0	12
	1 1 0	0	1	1	12
	1 1 1	1	1	1	8

	Digit Triple	x	y	z	Next Table
Table (5)	0 0 0	0	1	1	17
	0 0 1	1	1	1	13
	0 1 0	1	0	1	13
	0 1 1	0	0	1	16
	1 0 0	0	0	0	16
	1 0 1	1	0	0	1
	1 1 0	1	1	0	1
	1 1 1	0	1	0	18
Table (6)	0 0 0	1	1	0	10
	0 0 1	1	0	0	19
	0 1 0	1	0	1	19
	0 1 1	1	1	1	15
	1 0 0	0	1	1	15
	1 0 1	0	0	1	20
	1 1 0	0	0	0	20
	1 1 1	0	1	0	1
Table (7)	0 0 0	0	0	0	21
	0 0 1	0	1	0	2
	0 1 0	1	1	0	2
	0 1 1	1	0	0	19
	1 0 0	1	0	1	19
	1 0 1	1	1	1	15
	1 1 0	0	1	1	15
	1 1 1	0	0	1	22
Table (8)	0 0 0	0	1	1	13
	0 0 1	0	0	1	20
	0 1 0	0	0	0	20
	0 1 1	0	1	0	2
	1 0 0	1	1	0	2
	1 0 1	1	0	0	19
	1 1 0	1	0	1	19
	1 1 1	1	1	1	4

	Digit Triple	x	y	z	Next Table
Table (9)	0 0 0	1	0	1	23
	0 0 1	1	1	1	15
	0 1 0	0	1	1	15
	0 1 1	0	0	1	20
	1 0 0	0	0	0	20
	1 0 1	0	1	0	2
	1 1 0	1	1	0	2
	1 1 1	1	0	0	24
Table (10)	0 0 0	1	1	0	6
	0 0 1	0	1	0	12
	0 1 0	0	1	1	12
	0 1 1	1	1	1	13
	1 0 0	1	0	1	13
	1 0 1	0	0	1	16
	1 1 0	0	0	0	16
	1 1 1	1	0	0	2
Table (11)	0 0 0	0	0	0	3
	0 0 1	0	0	1	21
	0 1 0	0	1	1	21
	0 1 1	0	1	0	17
	1 0 0	1	1	0	17
	1 0 1	1	1	1	24
	1 1 0	1	0	1	24
	1 1 1	1	0	0	16
Table (12)	0 0 0	1	1	0	18
	0 0 1	0	1	0	10
	0 1 0	0	0	0	10
	0 1 1	1	0	0	3
	1 0 0	1	0	1	3
	1 0 1	0	0	1	4
	1 1 0	0	1	1	4
	1 1 1	1	1	1	17

	Digit Triple	x	y	z	Next Table
Table (13)	0 0 0	0	1	1	8
	0 0 1	1	1	1	5
	0 1 0	1	1	0	5
	0 1 1	0	1	0	10
	1 0 0	0	0	0	10
	1 0 1	1	0	0	3
	1 1 0	1	0	1	3
	1 1 1	0	0	1	15
Table (14)	0 0 0	1	0	1	16
	0 0 1	1	0	0	23
	0 1 0	1	1	0	23
	0 1 1	1	1	1	18
	1 0 0	0	1	1	18
	1 0 1	0	1	0	22
	1 1 0	0	0	0	22
	1 1 1	0	0	1	3
Table (15)	0 0 0	1	0	1	4
	0 0 1	1	1	1	9
	0 1 0	1	1	0	9
	0 1 1	1	0	0	6
	1 0 0	0	0	0	6
	1 0 1	0	1	0	7
	1 1 0	0	1	1	7
	1 1 1	0	0	1	13
Table (16)	0 0 0	1	0	1	14
	0 0 1	0	0	1	4
	0 1 0	0	1	1	4
	0 1 1	1	1	1	5
	1 0 0	1	1	0	5
	1 0 1	0	1	0	10
	1 1 0	0	0	0	10
	1 1 1	1	0	0	11

	Digit Triple	x	y	z	Next Table
Table (17)	0 0 0	0	1	1	5
	0 0 1	0	1	0	22
	0 1 0	0	0	0	22
	0 1 1	0	0	1	11
	1 0 0	1	0	1	11
	1 0 1	1	0	0	23
	1 1 0	1	1	0	23
	1 1 1	1	1	1	12
Table (18)	0 0 0	1	1	0	12
	0 0 1	1	1	1	24
	0 1 0	1	0	1	24
	0 1 1	1	0	0	14
	1 0 0	0	0	0	14
	1 0 1	0	0	1	21
	1 1 0	0	1	1	21
	1 1 1	0	1	0	5
Table (19)	0 0 0	1	1	0	24
	0 0 1	1	0	0	6
	0 1 0	0	0	0	6
	0 1 1	0	1	0	7
	1 0 0	0	1	1	7
	1 0 1	0	0	1	8
	1 1 0	1	0	1	8
	1 1 1	1	1	1	23
Table (20)	0 0 0	0	1	1	22
	0 0 1	0	0	1	8
	0 1 0	1	0	1	8
	0 1 1	1	1	1	9
	1 0 0	1	1	0	9
	1 0 1	1	0	0	6
	1 1 0	0	0	0	6
	1 1 1	0	1	0	21

	Digit Triple	x	y	z	Next Table
Table (21)	0 0 0	0	0	0	7
	0 0 1	0	0	1	11
	0 1 0	1	0	1	11
	0 1 1	1	0	0	23
	1 0 0	1	1	0	23
	1 0 1	1	1	1	18
	1 1 0	0	1	1	18
	1 1 1	0	1	0	20
Table (22)	0 0 0	0	1	1	20
	0 0 1	0	1	0	17
	0 1 0	1	1	0	17
	0 1 1	1	1	1	24
	1 0 0	1	0	1	24
	1 0 1	1	0	0	14
	1 1 0	0	0	0	14
	1 1 1	0	0	1	7
Table (23)	0 0 0	1	0	1	9
	0 0 1	1	0	0	14
	0 1 0	0	0	0	14
	0 1 1	0	0	1	21
	1 0 0	0	1	1	21
	1 0 1	0	1	0	17
	1 1 0	1	1	0	17
	1 1 1	1	1	1	1
Table (24)	0 0 0	1	1	0	19
	0 0 1	1	1	1	18
	0 1 0	0	1	1	18
	0 1 1	0	1	0	22
	1 0 0	0	0	0	22
	1 0 1	0	0	1	11
	1 1 0	1	0	1	11
	1 1 1	1	0	0	9