Journal of Materials Processing Technology 139 (2003) 219–225

Journal of Materials Processing Technology

www.elsevier.com/locate/jmatprotec

Rapid prototyping for self-similarity design

S.C. Soo, K.M. Yu*

Department of Manufacturing Engineering, The Hong Kong Polytechnic University, Hung Hom, Hong Kong, PR China

Abstract

This paper proposes a method for the rapid prototyping (RP) of self-similar objects. RP technology makes available the physical generation of a solid object. However, the contemporary design capability of a computer-aided design (CAD) system can atmost use the non-uniform rational B-spline (NURBS) modelling method. When the design has a very complicated shape with self-similarity, contemporary CAD systems can no longer handle the object. In fact, most objects in nature are self-similar. It is valuable to develop a method that can fabricate self-similar objects. Natural objects, such as mountains, clouds, and trees have irregular or fragmented features with self-similarity. These natural objects can be realistically described the by fractal geometry method, while Euclidean geometry is mainly used to represent simple man-made objects such as polyhedra. A typical self-similar fractal solid, the Menger sponge, will be used to illustrate the method.

In order to represent a self-similar fractal object for RP, the first task is to develop a method to model the object in a computable form. In this paper, a new data structure, called the radial-blossoming tree (RBT) structure, is proposed and implemented in order to bridge the gaps among CAD, RP and fractal geometry. Based on the RBT representation, an RP toolpath can be traced out more efficiently. The Menger sponge will then be produced layer by layer from the RP machine.
© 2003 Elsevier Science B.V. All rights reserved.

Keywords: Rapid prototyping; Fractal geometry; Data structure; Fractal modelling

1. Introduction

Computer-aided design (CAD) is at the heart of the present industrial revolution. The design capabilities of contemporary CAD systems are either based on the solid modelling method or on the surface modelling method [1]. The surface modelling method gives a precise and a convenient way to sculpture free-form surfaces, while the solid modelling method provides an unambiguous and information-complete solution to define a three-dimensional object [2]. On the other hand, rapid prototyping (RP) technology is used with CAD to make available the physical generation of a solid object.

Natural objects, such as mountains, clouds, and trees have irregular or fragmented features with self-similarity. These natural objects can be realistically described by fractal geometry while Euclidean geometry is mainly used to represent man-made objects such as squares, circles and triangles. It is thus valuable to develop a method that can manufacture self-similar designed objects. Here, fractal geometry is used in the representation since it has the important geometric property of self-similarity. Fractal geometry can also provide a simple way to represent jewellery design which needs to be aesthetically appealing to humans. However, no commercial CAD systems are to the efficient manipulation of fractal geometry. For example, the non-uniform rational B-spline (NURBS) method in surface modelling cannot be used to model a fractal curve due to fragmentation. Further, the constructive solid geometry (CSG) method in solid modelling will represent an object hierarchically, but it cannot model the self-similarity in the fractal object.

The RP cycle can be broken down into stages as shown in Fig. 1. A computer model of an Euclidean object is first generated with the help of CAD systems. The facetting step is to approximate the original model with facets. Obviously, this has inherent accuracy problems. Recently, there have been much research on how to minimize this problem by using direct slicing [3]. The purpose of the slice model is to provide information for the generation of a layer manufacture toolpath. Contour data are obtained from slicing the STL file or the computer model directly. Slicing is a time-consuming process and it further approximates the STL file or the computer model with slab volumes. As a result, much research have been done in proposing different slicing algorithms to achieve a more accurate result [4].

* Corresponding author.
E-mail address: mfkmyu@polyu.edu.hk (K.M. Yu).

0924-0136/03/$ – see front matter © 2003 Elsevier Science B.V. All rights reserved.
doi:10.1016/S0924-0136(03)00223-1

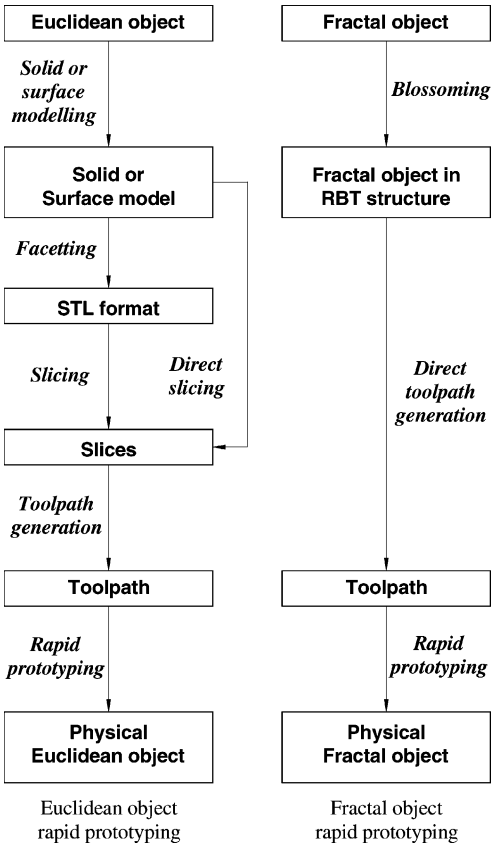

Fig. 1. The RP workflow.

In the next stage, a toolpath will be generated. Finally, the information in the toolpath is converted to RP machine codes for physical prototype fabrication.

In order to manufacture fractal objects, say with RP, a suitable modelling method needs to be developed. The method should build a computer model that can communicate with the RP process. In the following sections, a new data structure to model fractal solids will be explained in detail. Based on the data structure, the facetting and slicing approximation stages in the RP process can be omitted (Fig. 1) and their associated accuracy problems avoided. Moreover, the RP toolpath will be obtained directly from the data structure.

2. Fractal geometry

A fractal is by definition a set for which the Hausdorff–Besicovitch dimension strictly exceeds the topological dimension [5]. The Hausdorff–Besicovitch dimension is also termed the fractal dimension of the set. It is used to describe the fragmentation of an object, as the fractal dimension is usually a non-integer which is a measure of the roughness, or fragmentation, of the object. It provides a very convenient way to represent or to construct self-similar fractals

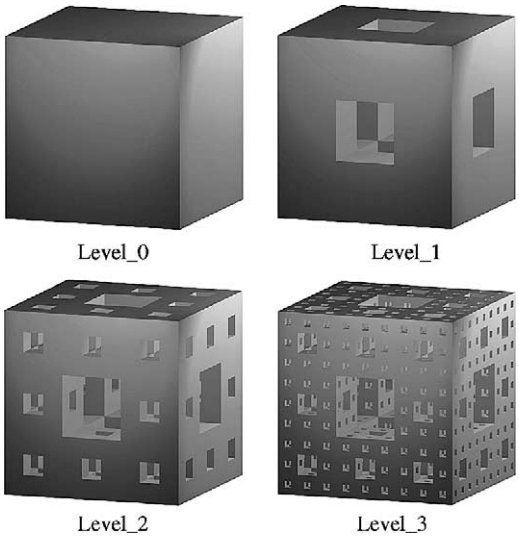

Fig. 2. The first four iterations of the Menger sponge.

[6]. A typical self-similar fractal object, the Menger sponge as shown in Fig. 2, will be used to illustrate the proposed method. The Menger sponge is created by removing cubes from the centre and side-walls of the generator, the size of the cubes being removed is one-third that of the generator cube. The process is repeated to produce a higher level Menger sponge.

3. Radial-blossoming tree (RBT) data structure

The Menger sponge building follows an omni-direction recursive sub-cubes (subdivision-cubes) removal. In this paper, a new data structure, called the RBT structure, is proposed. The generator cube will be the root node of the tree. The sub-cubes are related to their parents through tree node blossoming. Since sub-cubes are removed from all directions, the tree blossoming is balanced. Finally, whether a sub-cube is retained or removed will be indicated by assigning a status of solid or void respectively to the terminal node. Based on the RBT representation, the fractal solid can be represented efficiently.

The building unit of the data structure is a *26-connected* data node called the *b-node* ($26 = 3^3 - 1$). Three pieces of information will be stored in a node. A b-node is defined as follows:

```
struct b-node
{
   float len;
   bool fill;
   b-node *T, *TE, *TNE, *TN, *TNW, *TW, *TSW,
*TS, *TSE, *CE, *CNE, *CN, *CNW, *CW, *CSW,
*CS, *CSE, *B, *BE, *BNE, *BN, *BNW, *BW,
*BSW, *BS, *BSE;
}
```

The information consists of:

1. A parameter ***len*** to store the side length of the cube.
2. A Boolean value ***fill*** = {1 or 0}. This value keeps track of the material providing status. If the value is 1, the position needs to be filled with material while 0 indicates no filling.
3. Twenty-six pointers provide topological links to its offsprings. The pointers are grouped into three sets, the top set, the centre set and the bottom set. There are nine pointers in the centre set. For example, the east pointer of centre is named **CE** and the northeast pointer of centre is labelled **CNE**. Similarly, the remaining pointers of the centre set are **CN**, **CNW**, **CW**, **CSW**, **CS** and **CSE**. Both the top and the bottom sets have nine pointers and each pointer is labelled with its direction relative to the centre. For the top set, the upward pointer is labelled as **T**, the others have corresponding labels of **TE**, **TNE**, **TN**, **TNW**, **TW**, **TSW**, **TS** and **TSE**. Likewise for the bottom set, the corresponding labels are **B**, **BE**, **BNE**, **BN**, **BNW**, **BW**, **BSW**, **BS** and **BSE**. Thus, neighbourhood information in the 26 directions can be easily obtained from the pointers.

3.1. Level_0 RBT data structure

The Level_0 RBT data structure is a single b-node. The value ***len*** and ***fill*** will be instantiated but not the 26 pointers. Fig. 3 shows the Level_0 RBT data structure.

3.2. Higher level RBT data structure

The single b-node of the Level_0 RBT data structure will blossom into 26 and only 26 child b-nodes to form the Level_1 RBT data structure. The 26 pointers in Level_0 will point to these 26 new b-nodes. Fig. 4 shows the Level_1 RBT data structure. Furthermore, each b-node will blossom into 26 new b-nodes to generate higher level data structure. Fig. 5 shows the Level_2 RBT data structure.

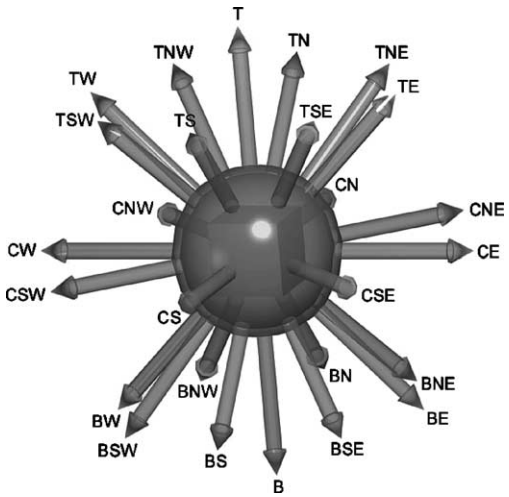

Fig. 3. The Level_0 RBT data structure.

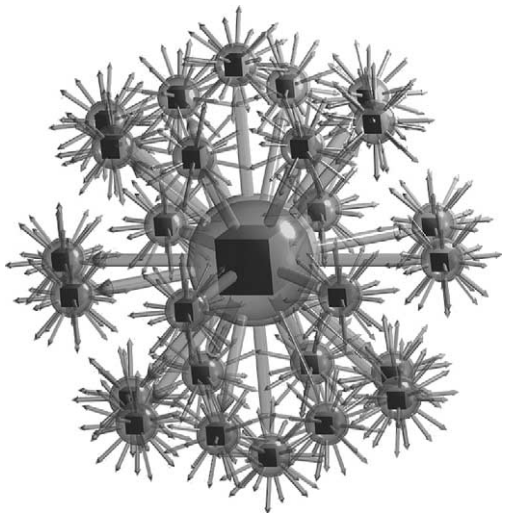

Fig. 4. The Level_1 RBT data structure.

3.3. Terminal cell node

In order to obtain the toolpath information directly from the RBT, a terminal cube model needs to be represented explicitly in the RBT. A *cell node* is, thus, introduced to represent the terminal b-node (Fig. 6), which is a *4-connected* data node.

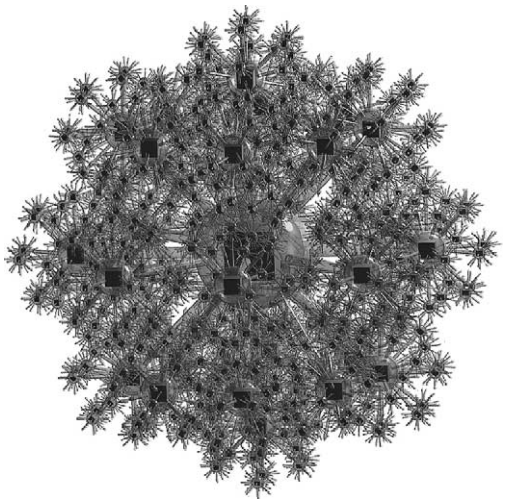

Fig. 5. The Level_2 RBT data structure.

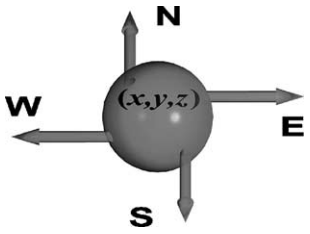

Fig. 6. A cell node.

A cell node is defined as follows:

```
struct cell
{
   float x, y, z;
   cell *E, *N, *W, *S;
}
```

Two pieces of information will be stored in the cell node:

1. Coordinate values (x, y, z) provide the positioning information for toolpath generation.
2. Four pointers are used to provide the topological information for layerwise toolpath generation. The pointers are **N** for the north direction, **E** for the east direction, **S** for the south direction and **W** for the west direction.

The cell nodes will form a layerwise data structure to represent the terminal cube (Fig. 7).

The required number of cell nodes $\mathbf{N}^3$ is determined as follows: denote the width of a single toolpath, say filament of fused deposition modelling (FDM) machine, by δ. Then **N** is determined by the following equation:

$$N = \text{Round}\left(\frac{\text{len}}{\delta}\right)$$

The Round(·) is a function to round off the value to an integer. Thus, the number of cell nodes is determined by $\mathbf{N}^3$. Afterwards, all terminal nodes will have this number of cell nodes. For example, when $\mathbf{N} = 2$, four cell nodes will be used to form a circular-linked list. From the terminal cube, the coordinate values of the four bottom vertices are copied to the corresponding coordinate values of the four cell nodes. Similarly, the coordinate values of the four top vertices are copied to the four top cell nodes. Thus, the terminal cube is represented by two layers of a circular-linked list. Also, there are two pointers to mark the starting cell node of each layer. Fig. 8 illustrates a processed terminal node.

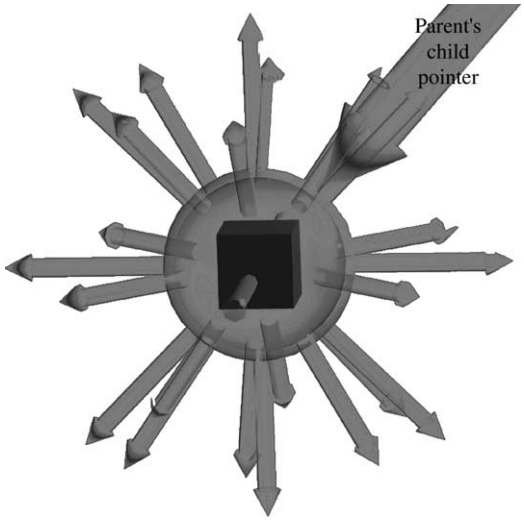

Fig. 7. A terminal b-node.

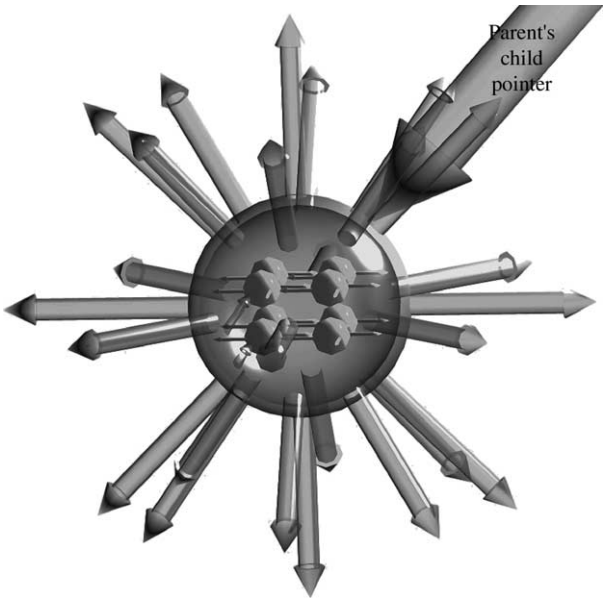

Fig. 8. Terminal node with $N^3 = 2^3$ cell nodes, filament resolution equals to 2.

4. The RBT data structure generation

4.1. Initialise the RBT

Firstly, a b-node is created and the original size of the cube is stored. Next, assign all 26 pointers with null value. This first b-node is also denoted as Level_0 of the RBT data structure. Fig. 3 illustrates the b-node.

4.2. Level_1 RBT generation

The Level_1 RBT data structure can now be generated.

Step 1. Mark ***fill*** $= 0$ for the Level_0 RBT.
Step 2. Create a new b-node and assign all its pointers with null value.
Step 3. Assign the **BSW** pointer of the Level_0 RBT to point to the new b-node.
Step 4. Repeat Steps 2 and 3 to allocate the remaining 25 new b-nodes.
Step 5. Mark the ***fill*** value of all new b-nodes with the proper value. If the corresponding cube of the b-node is void, mark ***fill*** $= 0$. Otherwise, mark ***fill*** $= 1$.

Thus, the Level_1 RBT data structure is formed as shown in Fig. 4.

4.3. Higher level RBT generation

The higher level RBT generation is similar to the Level_1 RBT generation. Treat all b-nodes in the previous level RBT

as the parent b-node in Level_0. The general procedure for next level generation is as follows:

Step 1. Mark ***fill*** = 0 for all b-nodes of the preceding level.
Step 2. Choose the preceding b-node which is pointed by the **BSW** pointer. For convenience, the chosen b-node is called the parent node.
Step 3. Create a new b-node and assign all its pointers with null value.
Step 4. Assign the **BSW** pointer parent node to point to the new b-node.
Step 5. Repeat Steps 3 and 4 to allocate the remaining 25 new b-nodes for the parent node.
Step 6. Mark the ***fill*** value of all the new b-nodes with a proper value. If the corresponding cube of the b-node is void, mark ***fill*** = 0. Otherwise, mark ***fill*** = 1.
Step 7. Repeat Steps 2–6 for allocating the remaining 25,326 new b-nodes for the remaining 25 parent nodes.

Thus, the higher level RBT data structure is formed to represent the Menger sponge. For example, the Level_2 RBT data structure is shown in Fig. 5.

5. Direct toolpath generation

With the help of the RBT data structure, the RP toolpath can be generated more efficiently. Two types of toolpath in each layer have to be considered. They are the contour toolpath and the area-filling toolpath. All toolpaths can be obtained from the relation of the pointer directions and the chaining operation among the data nodes. For convenience, the status of a cell node will be classified as follows:

(i) A cell has at least one pointer which is either pointing to a null value or a ***fill*** = 0 cell node.
(ii) A cell that does not have a null value pointer will provide area-filling information. The cell is called an *interior cell*.

5.1. Traversal algorithm

The toolpath generation is, in fact, an RBT traversal algorithm. The traversal is largely depth-first. For chaining cells as a toolpath, a vector annihilation operation is performed.

5.1.1. Vector annihilation

If the traversal encounters a null value, it will return to the parent b-node and find a neighbouring cell in the direction where the corresponding pointer statuses are a "mirror" to the current cell. Fig. 9(a) and (b) illustrates the vector annihilation process.

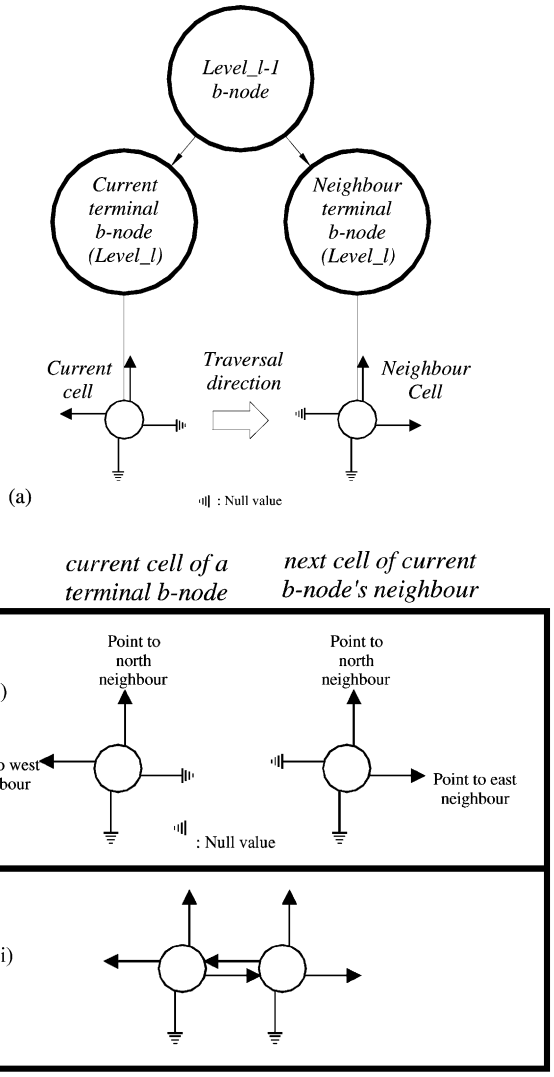

Fig. 9. (a) Neighbour cell for annihilation. (b) Vector annihilation: (i) before annihilation; (ii) after annihilation.

For example, assume the traversal direction is to the east and the statuses of the pointers are:

(i) **E**: points to null;
(ii) **N**: points to the north neighbour;
(iii) **W**: points to the west neighbour;
(iv) **S**: points to null;

then, the appropriate next cell will have pointers of:

(a) **E**: points to the east neighbour (opposite to (i));
(b) **N**: points to the north neighbour (same as (ii));
(c) **W**: points to null (opposite to (iii));
(d) **S**: points to null (same as (iv));

Fig. 10 shows the flowchart of the traversal algorithm. The key to derive a specific traversal algorithm is to define a neighbourhood system. In other words, the traversing order among the b-nodes has to be properly defined. Two traversal algorithms are derived to generate the RP toolpath.

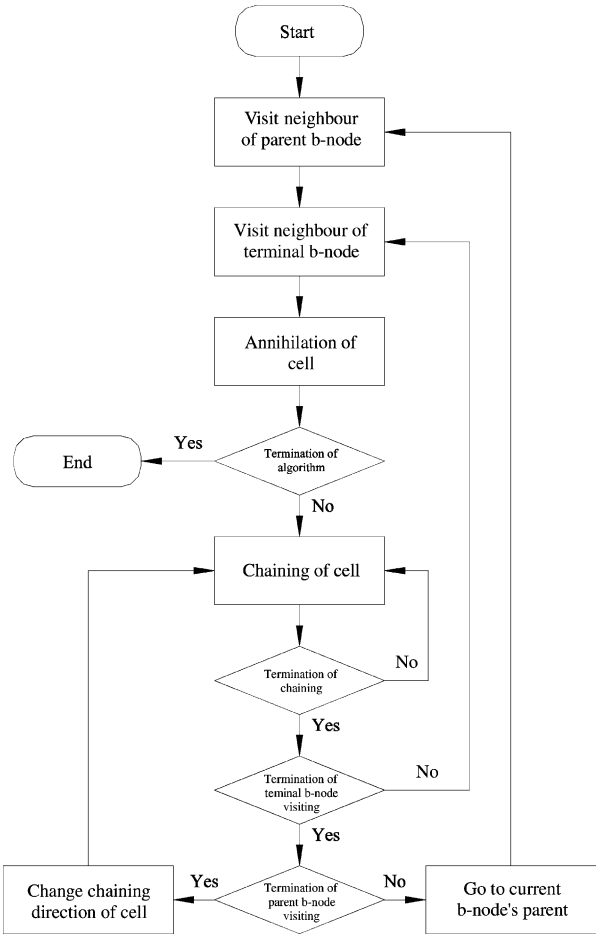

Fig. 10. The RBT data structure traversal algorithm.

5.2. Contour toolpath generation—circular traversal

In a slice, all contour toolpaths are a closed polygon and the traversing order is defined counter-clockwise, i.e.:

$$\begin{array}{ccccc} \mathbf{xNW} & \leftarrow & \mathbf{xN} & \leftarrow & \mathbf{xNE} \\ \downarrow & & & & \uparrow \\ \mathbf{xW} & & \mathbf{x} & & \mathbf{xE} \\ \downarrow & & & & \uparrow \\ \mathbf{xSW} & \rightarrow & \mathbf{xS} & \rightarrow & \mathbf{xSE} \end{array}$$

where **x** means **B**, **C** or **T**, the corresponding set of b-nodes. Notice that the algorithm is terminated when the starting cell is visited twice. All contour toolpaths in the slice are similarly obtained. Other layers of the RBT are then traversed.

5.3. Area-filling toolpath generation—raster traversal

The zig-zag toolpath is a common area-filling toolpath. The traversing order is defined as

$$\begin{array}{ccccc} \mathbf{xNW} & \rightarrow & \mathbf{xN} & \rightarrow & \mathbf{xNE} \\ \uparrow & & & & \\ \mathbf{xW} & \leftarrow & \mathbf{x} & \leftarrow & \mathbf{xE} \\ & & & & \uparrow \\ \mathbf{xSW} & \rightarrow & \mathbf{xS} & \rightarrow & \mathbf{xSE} \end{array}$$

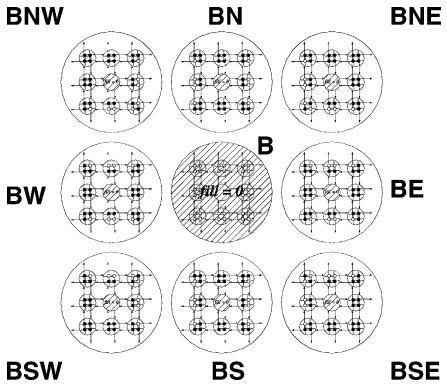

Fig. 11. The Level_2 RBT data structure. The black dotted nodes provide the contour toolpath information while the white nodes provide the area-filling information. Moreover, nodes not providing any toolpath information are hatched.

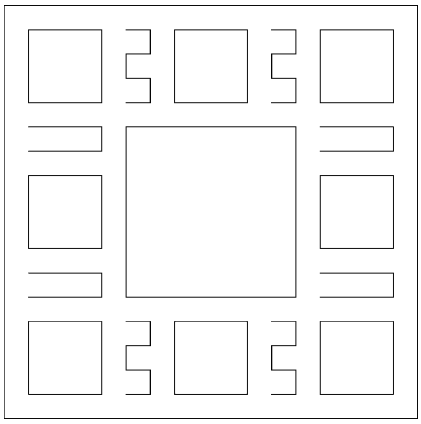

Fig. 12. The toolpath corresponding to Fig. 11.

The algorithm will terminate when all interior cells are visited. As a result, all the area-filling toolpaths can be obtained. Fig. 11 shows the first layer RBT data structure. Fig. 12 shows the top view of first layer toolpath of a Level_2 Menger Sponge in two filaments resolution. Fig. 13 shows all the toolpaths.

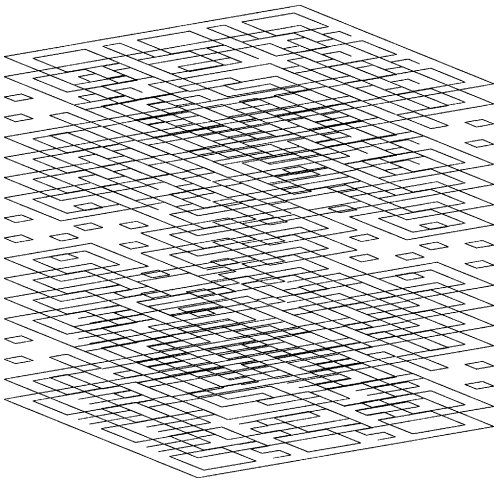

Fig. 13. All toolpaths of Level_2 RBT data structure (filament resolution is 2).

6. FDM toolpath format

From the previous sections, the toolpath data are simply polylines or polygons. In order to make the fractal object directly with the RP machine, the geometric information in the toolpath has to be converted into a machine-readable form. Here, the machine code will be in SML file format of the FDM 1600 RP machine [7]. The toolpath information required by the FDM 1600 machine is the x, y and z coordinate values, which are also in ASCII. The file is then downloaded to the FDM 1600 machine for physical fractal object making.

7. Conclusions

A new data structure is designed. A layerwise traversal of the data structure is also devised which is suitable for fabricating a fractal solid with RP or CNC machining. For instance, the method is applicable to stereolithography (SL), FDM, three-dimensional printing (3DP) and solid ground curing (SGC). In addition, the resolution of the toolpath can be controlled by the number of cell nodes in the terminal b-nodes.

Acknowledgements

The work described in this paper was supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region (Project No. PolyU 5141/98E) and the Hong Kong Polytechnic University (Project No. B-Q261).

References

[1] T.T. Wohlers, Rapid Prototyping and Tooling: State of the Industry, Wohlers Associates, 1999.
[2] I. Zeid, CAD/CAM Theory and Practice, McGraw-Hill, New York, 1991.
[3] R. Jamieson, H. Hacker, Direct slicing of CAD models for rapid prototyping, Rapid Prototyping J. 1 (2) (1995) 4–12.
[4] E. Sabourin, S.A. Houser, J.H. Bohn, Adaptive slicing using stepwise uniform refinement, Rapid Prototyping J. 2 (4) (1996) 20–26.
[5] B.B. Mandelbrot, The Fractal Geometry of Nature, Freeman, New York, 1982.
[6] K.J. Falconer, Techniques in Fractal Geometry, Wiley, New York, 1997.
[7] FDM 1600 Manual Release 2.0, Stratasys, 1995.