Geometric Modelling of General Sierpinski Fractals using iterated function system in Matlab

ZHIYONG ZHU
Northwest A&F University
College of Science
Taicheng Road 3, 712100 Yangling
CHINA
yzzhu0412@163.com

ENMEI DONG*
Northwest A&F University
College of Science
Taicheng Road 3, 712100 Yangling
CHINA
enmei117@sohu.com

Abstract: Study on properties of general Sierpinski fractals, including dimension, measure, Lipschitz equivalence, etc is very interesting. Like other fractals, general Sierpinski fractals are so complicated and irregular that it is hopeless to model them by simply using classical geometry objects. In [22], the authors the geometric modelling of a class of general Sierpinski fractals and their geometric constructions in Matlab base on iterative algorithm for the purpose of studying fractal theory. In this paper, we continue such investigation by adding certain rotation structure and obtain some results by extending our approaches to three dimensional space. Our results may be used for any graphical goal, not only for mathematical reasons.

Key–Words: general Sierpinski carpet, general Sierpinski gasket, general Sierpinski sponge, IFS, deterministic algorithm, random iterated algorithm, Matlab

1 Introduction

Geometric modelling of fractal objects not only plays an important role in the study of fractal theory, but also is a difficult process in the field of computer graphics. Computer simulation is based on the basic theory of fractal. But drawing strange and amazing fractal images by computer, simultaneously in turn can provide us the most intuitive discussion and explanation, greatly promoting the development of fractal theory. Computer graphics provides many algorithms to generate fractal images, such as IFS(Iterated Function System), L-system, recursive, time-escaped algorithm, etc. There are various source programs are designed based each algorithm. However, to our knowledge, most algorithm and source programs are for graphical goal and often done by non-mathematical researchers. This makes them are difficult to understand and application for most mathematical researchers because of the lack of professional knowledge of programming language(such as Visual Basic, Visual C++, Delphi, Java, etc). Also, for this reason, the following question aries naturally.

Question 1. *How to use a popular and easy-to-understand way as much as possible, in a real-time information exchange interface, such that the mathematical researchers can well obtain the desired images in their study by modifying a few parameters, without need to know much about the complex computer programming language?*

The present paper does not give a complete answer to Question 1, which is likely to be extremely hard. It does, however, study the question in several important special case that should allow us to gain some deep insight into the problem.

Fixing an integer $n \geq 2$, let $\mathcal{D}_1 = \{0, 1, \cdots, n-1\}^2$ and $\mathcal{D}_2 = \{1, \cdots, n\}^2$. For $A \subset \mathcal{D}_1, B \subset \mathcal{D}_2$, we assume that $1 < \sharp A + \sharp B < n^2$ to exclude the trivial case, where $\sharp A$ and $\sharp B$ denote the cardinalities of A and B respectively. Let $T := T(A, B) \subset \mathbb{R}^2$ be the unique non-empty compact set satisfying the following set equation:

$$T = [(T + A) \cup (B - T)]/n.$$

We shall call $T(A, B)$ a general Sierpinski carpet throughout this paper. Let $\alpha = (1, 0)$ and $\beta = (1/2, \sqrt{3}/2)$. If we recast the above sets $\mathcal{D}_1$ and $\mathcal{D}_2$ as follows:

$$\mathcal{D}_1 = \Big\{k_1\alpha + k_2\beta : k_1 + k_2 \leq n-1 \text{ and } k_1, k_2 \in \mathbb{N} \cup \{0\}\Big\}$$

and

$$\mathcal{D}_2 = \Big\{k_1\alpha + k_2\beta : 2 \leq k_1 + k_2 \leq n \text{ and } k_1, k_2 \in \mathbb{N}\Big\},$$

then $T(A, B)$ is called a general Sierpinski gasket. Notice that here A or B may be empty. In general,

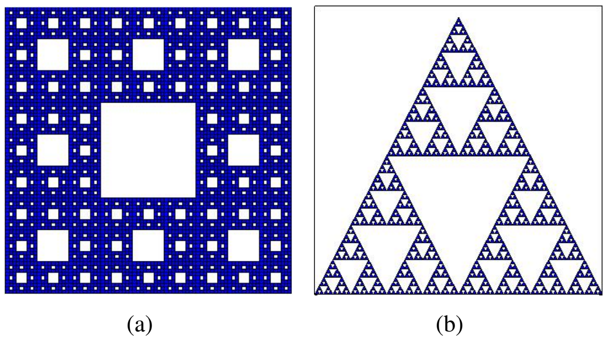


Figure 1: (a) Sierpinski carpet (b) Sierpinski gasket

$T(A,B)$ is of 180° rotation structure when $B \neq \emptyset$. A familiar example of a general Sierpinski carpet is the Sierpinski carpet(see Figure 1(a)), in which $n = 3, A = \mathcal{D}_1 \setminus \{(1,1)\}\}$, $B = \emptyset$. A familiar example of a general Sierpinski gasket is the Sierpinski gasket (see Figure 1(b)), in which $n = 2, A = \{(0,0),(1,0),(1/2,\sqrt{3}/2)\}$, $B = \emptyset$.

Let $Q = [0,1]^2$(resp. $\triangle$), where

$$\triangle = \{c_1\alpha + c_2\beta : c_1 + c_2 \leq 1 \text{ and } 0 \leq c_1, c_2 \leq 1\}$$

be the equilateral triangle with lower-left coordinate $(0,0)$ and side 1. We define $T^0(A,B) = Q$,

$$T^1(A,B) = [(Q+A) \cup (B-Q)]/n,$$

and recurrently,

$$T^{k+1}(A,B) = [(T^k(A,B)+A) \cup (B - T^k(A,B))]/n$$

for $k \geq 1$. Then $T^k(A,B)$ is a union of squares (resp. equilateral triangles) of size $1/n^k$ (called them the k-squares (resp. k-triangles)). For simplify, we shall call $T^k(A,B)$ a k-squares (resp. k-triangles). Clearly $T^{k+1}(A,B) \subset T^k(A,B)$, and

$$T(A,B) = \bigcap_{k=1}^{\infty} T^k(A,B).$$

Recently, many works have been devoted to the study of properties of general Sierpinski fractals that described as above include, dimension, measure, Lipschitz equivalence, etc, see [8, 10–12, 17, 18, 21–25] and the references in all of these. There has been notable progress on the study of general Sierpinski fractals. Yet much is still unknown, and this progress has led to more unanswered questions, especially the general Sierpinski fractals with certain rotation structure. For such fractal, how to achieve the goal described in question 1 has become the most concerned problem for most mathematical researchers at present.

An fundamental concept in fractal geometry is iterated function systems. It is introduced in Hutchinson [9] as a unified way of generating and classifying a broad class of fractals which contains classical Cantor sets, dragon curves, limit sets of Kleinian groups, Sierpinski gaskets, Julia set, and much more. Among all algorithms for generating fractal images, IFS algorithm may be most popular one. There many application on it(see [1,2,4,7,14,16,22] and references there in). IFS algorithm include deterministic algorithm and random iterated algorithm. The mathematical basis of deterministic algorithm was developed in Hutchinson [9]. The random iterated algorithm was developed in Barnsley and Demko [2]. IFS algorithm is very simple, it is easy to implement in any programming language and the results it generates are very spectacular and may be used for any mathematical reason, not only for graphical goal.

Matlab is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numerical computation. A proprietary programming language developed by MathWorks. It has a strong audience within the applied mathematics community. Matlab was first adopted by researchers and practitioners in control engineering, Little's specialty, but quickly spread to many other domains. It is now also used in education, in particular the teaching of linear algebra, numerical analysis, and is popular amongst scientists involved in image processing. For further details on Matlab, see [13].

Notice that the powerful advantages of Matlab in numerical calculation and graphic visual ability and its programming language is more easily understood and mastered by researchers in mathematics. In [22], the authors study the geometric modelling of general Sierpinski with no rotation structure and their geometric constructions in Matlab environment base on IFS algorithm for the purpose of mathematical research. In this paper, we continue such investigation started in [22]. We remove the restriction on no rotation structure and obtain the geometric modelling of $T(A,B)$ and $T^k(A,B)$, which are needed in the study of properties of general Sierpinski fractal. The objects considered in the present paper are complex and intriguing since the 180° rotation structure are allowed. Although the idea used was early developed by others, the analysis is technical and the results obtained are valuable and interesting to the readers.

The paper is organized as follows: In Section 2, we briefly review the iterated function system and corresponding algorithms: deterministic algorithm and random iterated algorithm. We discuss the geometric modelling of $T^k(A,B)$ using deterministic algorithm in Section 3, and give corresponding source program designed in Matlab programming language. In Section 4, we present the random iterated algorithm implemented in the Matlab programming language to generate $T(A, B)$. In Section 5, we discuss the extension of our methods in three dimensional space. Section 6 is conclusion. Several examples are given in Section 3, Section 4 and Section 5 to illustrate our results.

2 Preliminary

2.1 Iterated Function System

Let (X, d) be a complete metric space, often Euclidean space $\mathbb{R}^n$. We say that the mapping $S : X \to X$ is a contraction with contraction ratio r if

$$r = \sup_{x \in X, x \neq y} \frac{d(S(x), S(y))}{d(x, y)} < 1.$$

In particular, we say that a contraction S with contraction ratio r is a similitude if $d(S(x), S(y)) = rd(x, y)$ for all $x, y \in X$.

We call a finite family of contractions $\{S_i : i = 1, 2, \cdots, N\}$ are defined in (X, d) an iterated function system or IFS, and denote it by $\{X : S_i, i = 1, 2, \cdots, N\}$. If r_i is contraction ratio of S_i, $i = 1, 2, \cdots, N$, then $r = max\{r_1, r_2, \cdots, r_N\}$ is called contraction ratio of IFS. Let $\mathcal{F}(X)$ denote the class of all non-empty subsets of X. For $A, B \subset \mathcal{F}(X)$, the Hausdorff metric between A and B is defined by

$$d_H(A, B) = max\left[\sup_{x \in A} d(x, B), \sup_{y \in B} d(A, y)\right].$$

By [9], $(\mathcal{F}(X), d_H)$ is a complete metric space when (X, d) is a complete metric space. It is a standard fact that every contraction map (in a complete metric space) has a unique fixed point. Applying the fact to $(\mathcal{F}(X), d_H)$, we can obtain the following contraction principle on $(\mathcal{F}(X), d_H)$, similar description can also be seen in many literatures, such as [5, 6, 9].

Lemma 1. *Let $\{X : S_i, i = 1, 2, \cdots, N\}$ be an iterated function system with contraction ration r and mapping $S : \mathcal{F}(X) \to \mathcal{F}(X)$ be defined by*

$$S(A) = \bigcup_{i=1}^{N} S_i(A), \text{ for any } A \in \mathcal{F}(X).$$

Then S is a contraction with ratio r and there is a unique fixed point(attractor or invariant set) K that satisfies

$$K = S(K) = \bigcup_{i=1}^{N} S_i(K)$$

and for any $A \in \mathcal{F}(X)$

$$K = \lim_{n \to \infty} S^n(A). \quad (1)$$

2.2 Deterministic Algorithm and Random Iterated Algorithm

Considering the iterated function system $\{\mathbb{R}^2 : S_i, i = 1, 2, \cdots, N\}$, where S_i is an affine transformation with the form

$$S_i\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_i & b_i \\ c_i & d_i \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e_i \\ f_i \end{pmatrix}, \quad (2)$$

$i = 1, 2, \cdots, N$. According to the Lemma 1, we can obtain two algorithms for generating fractal images on plane. One is deterministic algorithm, the other is random iterated algorithm.

Deterministic algorithm: The mathematical basis of this method is very simple, and it is the result of the Lemma 1. Here is basic process: Choose a nonempty set $A_0 \in \mathcal{F}(\mathbb{R}^2)$. Then compute successively the sequence of sets $\{A_m = S^m(A_0)\}_{m=1}^{\infty}$ by

$$A_{m+1} = S(A_m) = \bigcup_{i=1}^{N} S_i(A_m).$$

It follows from 1 that the sequence $\{A_m\}$ converges to the attractor K of the IFS $\{X : S_i, i = 1, 2, \cdots, N\}$ in the Hausdorff metric. In fact, if m is large enough, then A_m is approximately equal to K, and is basically indistinguishable. Thus the image of attractor drawn by us is actually A_m with large enough m. This approach to generate fractal requires heavy amount of memory, because in each iteration generate some image and to store image generated by affine transformation requires large amount of memory.

Random iterated algorithm: Let $P = \{p_1, p_2, \cdots, p_N\}$ be a set of probability weights, where p_i can be thought of as relative weight for each S_i and $\sum_{i=1}^{N} p_i = 1$. In general, we take

$$p_i \approx \frac{|detA_i|}{\sum_{i=1}^{N} |detA_i|},$$

where

$$A_i = \begin{pmatrix} a_i & b_i \\ c_i & d_i \end{pmatrix}$$

and $|detA_i|$ denote the determinant of A_i, $i = 1, 2, \cdots, N$. If $|detA_i| = 0$, we take a small positive number as p_i(For example $p_i = 0.01$) and make appropriate adjustments for other p_k such that $\sum_{i=1}^{N} p_i = 1$. An iterated function system with probabilities consists of an iterated function system $\{\mathbb{R}^2 :$

$S_1, S_2, \cdots, S_N\}$ together with a set $\{p_1, p_2, \cdots, p_N\}$ of probability weights. We often denote such iterated function system by

$$\{\mathbb{R}^2 : S_1, S_2, \cdots, S_N, p_1, p_2, \cdots, p_N\}.$$

This random method is different from the deterministic approach in that the initial set is a singleton point and at each level of iteration, just one of the defining contraction transformations is used to calculate the next level. At each level, the contraction transformation is randomly selected and applied. Points are plotted, except for the early ones, and are discarded after being used to calculate the next value. The random algorithm avoids the need of a large computer memory, it is best suited for the small computers on which one point at a time can be calculated and display on a screen. On the other hand it takes thousand of dots to produce an image in this way that does not appear too skimpy. Here is basic process: Choose an arbitrary point $x_0 \in \mathbb{R}^2$ as start point, we randomly choose a mapping S_i in $\{S_1, S_2, \cdots, S_N\}$ according to probability distribution $\{p_1, p_2, \cdots, p_N\}$, and move to $x_1 = S_i(x_0)$. We then make another random choice of S_j, and move to $x_2 = S_j(x_1)$. This continues indefinitely, we obtain a sequence $\{x_0, x_1, \cdots\}$. When N_{max} is large enough, according to Lemma 1, $\{x_n : n \geq N_{max}\}$ is indistinguishable from K. In particular, we choose $N_{max} = 0$ if $x_0 \in K$.

3 Construction of General Sierpinski Fractals Using Deterministic Algorithm

According to [3], we can obtain a general Sierpinski carpet(resp. general Sierpinski gasket) $T(A, B)$ described in introduction as follows: we decompose initial pattern $[0, 1]^2$(resp. $\triangle$) into n^2 closed subsquares(resp. subtriangles) in the obvious manner, so that these subsquares(resp. subtriangles) have disjoint interiors and sidelength $1/n$. We choose some subsquares(resp. subtriangles) according to the rule described by A and B, again divide each of these subsquares(resp. subtriangles) into n^2 congruent ones, chose the subsquares(resp. subtriangles) according the same rule and repeat the procedure inductively to the infinity, then we get the general Sierpinski gasket(resp. general Sierpinski carpet) $T(A, B)$. For any $k \geq 1$, then $T^k(A, B)$ described in introduction is the union of squares (resp. triangles) that are chosen in the step k. For $a \in A, b \in B$, set

$$S_a(x) = \frac{1}{n}(x + a), S_b(x) = \frac{1}{n}(b - x). \qquad (3)$$

Then $T(A, B)$ is unique invariant set of iterated function system $\{S_a\}_{a\in A} \cup \{S_b\}_{b\in B}$.

3.1 Steps for Creating $T^k(A, B)$(resp. $T(A, B)$) Using Deterministic Algorithm

For creating $T^k(A, B)$ using deterministic algorithm steps given below should be considered.

Step 1: Draw an initial pattern($[0, 1]^2$ or $\triangle$) on the plane, and decompose it into n^2 congruent ones in the obvious manner (for example, we can divide a given equilateral triangle into n^2 congruent triangles by drawing $n - 1$ lines, parallel to each edge and dividing the other two edges into n equal parts).

Step 2: Create a $2 \times 4l^k$ (resp. $2 \times 3l^k$) matrix of zeros to store vertex coordinates of all small squares (resp. triangles) chosen after each construction.

Step 3: Initialize the matrix in Step 2 with vertex coordinates of initial pattern which are described in the first step.

Step 4: Applying transformations (3) on vertex coordinates of initial pattern and repeat Step 1 for each subpattern obtained after applying transformations.

Step 5: Again apply transformations (3) on the vertex coordinates of all subpattern obtained in Step 4 and repeat Step 1 for each new subpattern.

Step 6: Repeat step 5 again and again, this step 5 can be repeated infinite number of times.

3.2 Source Program for Creating k-squares $T^k(A, B)$ Using Deterministic Algorithm

function Fractal_ square (E, F, n, k)
% FRACTAL_SQUARE: Display the geometric construction of k-squares $T^k(A, B)$.
% Call format: Fractal_square (E, F, n, k)
% E and F are two dimensional arrays with the form

$$[c_{11}, c_{12}, \ldots, c_{1m}; c_{21}, c_{22}, \ldots, c_{2m}],$$

where for any $1 \leq i \leq m$,

$$(nc_{1i}, nc_{2i}) \in \begin{cases} A, & \textit{when E is of such form;} \\ B, & \textit{when F is of such form.} \end{cases}$$

% $1/n$ *is contraction radio.*
% k is iterated depth.
% Divide a unit square $[0, 1]^2$ *into* n^2 *congruent squares by drawing* $n - 1$ *lines, parallel to a pair of opposite edges and dividing the other pair of opposite edges into n equal parts.*

```
t = 0 : 1/n : 1;
for i = 1 : (n + 1)
    tx(1) = t(i); tx(2) = t(i); ty = [0, 1];
    plot(tx, ty, 'b')
    hold on
    plot(ty, tx,′ b′)
    hold on
end
axis square
d = 1;
if isempty(E)
    LE = 0;
else
    LE =length(E(1, :));
end
if isempty(F)
    LF = 0;
else
    LF =length(F(1, :));
end
l = LE + LF;
M =zeros(2, 4 ∗ lˆk); % M is an 2 × 4l^k matrix of zeros which be used to store vertex coordinates of all small squares after each iteration.
M(:, 1 : 4) = [0, 1, 1, 0; 0, 0, 1, 1]; % Initialization of matrix M.
for h = 1 : k
    C = 1/n ∗ M; D = −1/n ∗ M;
    for i = 0 : LE − 1,
        M(:, i ∗ 4 ∗ (lˆ(h − 1)) + 1 : ((i + 1) ∗ (4 ∗ (lˆ(h − 1))))) = C(:, 1 : (4 ∗ (lˆ(h − 1)))) + E(:, i + 1) ∗ ones(1, 4 ∗ (lˆ(h − 1)));
    end
    for j = 0 : LF − 1
        M(:, ((LE + j) ∗ (4 ∗ (lˆ(h − 1)))) + 1 : (((LE + j) + 1) ∗ (4 ∗ (lˆ(h − 1))))) = D(:, 1 : (4∗(lˆ(h−1))))+F(:, j+1)∗ones(1, 4∗(lˆ(h−1)));
    end
    d = d/n;
% Divide each h−square into n^2 congruent squares
    for m = 1 : lˆh
        squaregrid(min(M(1, 4 ∗ m − 3 : 4 ∗ m)), min(M(2, 4 ∗ m − 3 : 4 ∗ m)), n, d);
    end
end
% Fill each small square in T^k(A, B) with blue.
for i = 1 : lˆk
    patch(M(1, 4 ∗ i − 3 : 4 ∗ i), M(2, 4 ∗ i − 3 : 4 ∗ i),′ b′);
end
set(gca,'xtick',[],'xticklabel',[]),
set(gca,'ytick',[],'yticklabel',[]) % Do not display the coordinate axis.
function squaregrid(x, y, r, s)
a = [x : s/r : x + s]; b = [y : s/r : y + s];
plot(a, meshgrid(b, a),′ b′)
hold on
plot(meshgrid(a, b), b,′ b′)
```

The percent-sign (%) implies that this is a remark statement after it, the text shown in italics following this sign. The remark statement is ignored when running program.

3.3 Source Program for Creating k-triangles $T^k(A, B)$ Using Deterministic Algorithm

```
function Fractal_triangle(E, F, n, k)
% FRACTAL_TRIANGLE: Display the geometric construction process of k−triangles T^k(A, B).
% Call format: Fractal_triangle(E, F, n, k).
% E and F are two dimensional arrays with the form
```

$$[c_{11}, c_{12}, \ldots, c_{1m}; c_{21}, c_{22}, \ldots, c_{2m}],$$

where for any $1 \le i \le m$,

$$(nc_{1i}, nc_{2i}) \in \begin{cases} A, & \text{when E is of such form;} \\ B, & \text{when F is of such form.} \end{cases}$$

```
% 1/n is contraction radio.
% k is iterated depth.
trianglegrid([0, 1, 1/2], [0, 0, sqrt(3)/2], n); % Draw an equilateral triangle with vertex coordinates (0, 0), (1, 0), (1/2, √3/2), and divide it into n^2 congruent triangles.
axis square, hold on
if isempty(E)
    LE = 0;
else
    LE =length(E(1, :));
end
if isempty(F)
    LF = 0;
else
    LF =length(F(1, :));
end
l = LE + LF;
M =zeros(2, 3 ∗ lˆk); % M is an 2 × 3l^k matrix of zeros which be used to store vertex coordinates of all small triangles after each iteration.
% Initialization of M.
M(:, 1 : 3) = [0, 1, 1/2; 0, 0, sqrt(3)/2];
for h = 1 : k
    C = 1/n ∗ M;
    D = −1/n ∗ M;
    for i = 0 : (LE − 1)
        M(:, i ∗ 3 ∗ (lˆ(h − 1)) + 1 : ((i + 1) ∗ (3 ∗ (lˆ(h − 1))))) = C(:, 1 : (3 ∗ (lˆ(h − 1)))) + E(:, i + 1) ∗ ones(1, 3 ∗ (lˆ(h − 1)));
    end;
    for j = 0 : (LF − 1)
        M(:, ((LE + j) ∗ (3 ∗ (lˆ(h − 1)))) + 1 : (((LE + j) + 1) ∗ (3 ∗ (lˆ(h − 1))))) = D(:, 1 : (3∗(lˆ(h−1))))+F(:, j+1)∗ones(1, 3∗(lˆ(h−1)));
    end;
    for m = 1 : l^h % Divide each equilateral triangle in T^h(A, B) into n^2 congruent triangles.
        trianglegrid(M(1, 3*m-2 : 3*m), M(2, 3*m - 2 : 3*m), n)
    end
end
% Fill each equilateral triangle in T^k(A, B) by blue.
for i = 1 : l^k
    patch(M(1, 3*i - 2 : 3*i), M(2, 3*i - 2 : 3*i),'b');
end
set(gca,'xtick',[],'xticklabel',[]),
set(gca,'ytick',[],'yticklabel',[])  % Do not display the coordinate axis.
function trianglegrid(x, y, n)
plot([x(1), x(2), x(3), x(1)], [y(1), y(2), y(3), y(1)])
hold on
a = [linspace(x(1), x(2), n+1); linspace(y(1), y(2), n + 1)];
b = [linspace(x(2), x(3), n+1); linspace(y(2), y(3), n + 1)];
c = [linspace(x(3), x(1), n+1); linspace(y(3), y(1), n + 1)];
for i = 2 : n
    plot([a(1, i), b(1, n+2-i)], [a(2, i), b(2, n+2-i)],'b')
    hold on
end
for j = 2 : n
    plot([b(1, j), c(1, n+2-j)], [b(2, j), c(2, n+2-j)],'b')
    hold on
end
for k = 2 : n
    plot([c(1, k), a(1, n + 2 - k)], [c(2, k), a(2, n + 2 - k)],'b')
    hold on
end
```

Remark 2. *The difference between here and source program presented in Section 3.3 is that we replace the command of dividing square by the command of dividing triangle in each step construction.*

3.4 Some Examples

Saving the text in a file called Fractal_ square.m (resp. Fractal_triangle.m) in your current directory. We can obtain the fine structure of $T^k(A, B)$ by calling corresponding self-defining function: Fractal_ square(E, F, n, k)(resp.Fractal_triangle(E, F, n, k)) in the Command Window of Matlab.

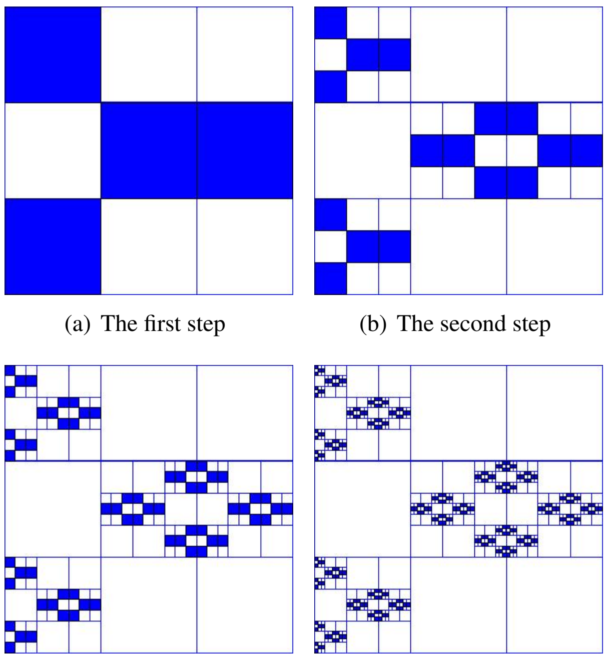

(a) The first step (b) The second step (c) The third step (d) The fourth step

Figure 2: The first four steps of constructing General Sierpinski carpet $T(A_1, B_1)$.

Example 1. *Let* $n = 3$*,*

$$A_1 = (0, 0), (0, 2), (2, 1), B_1 = (2, 2).$$

We can obtain the fine structures of the first four steps of constructing General Sierpinski carpet $T(A_1, B_1)$ *by running the following instructions in turn:*

Fractal_square$([0, 0, 2/3; 0, 2/3, 1/3], [2/3; 2/3], 3, k)$,

$k = 1, 2, 3, 4$,*see Figure 2.*

Example 2. *Let* $n = 3$*,*

$$A_2 = \{(1, 0), (\frac{1}{2}, \frac{\sqrt{3}}{2}), (\frac{3}{2}, \frac{\sqrt{3}}{2})\},$$

$$B_2 = (\frac{3}{2}, \frac{\sqrt{3}}{2}), (\frac{5}{2}, \frac{\sqrt{3}}{2}), (2, \sqrt{3})\}.$$

We can obtain the fine structures of the first four steps of constructing general Sierpinski gasket $T(A_2, B_2)$ *by running the instructions in turn: Fractal_triangle*$(E, F, 3, k)$*, where*

$$E = [1/3, 1/6, 1/2; 0, sqrt(3)/6, sqrt(3)/6],$$

$$F = [1/2, 5/6, 2/3; sqrt(3)/6, sqrt(3)/6, sqrt(3)/3],$$

$k = 1, 2, 3, 4$,*see Figure 3.*

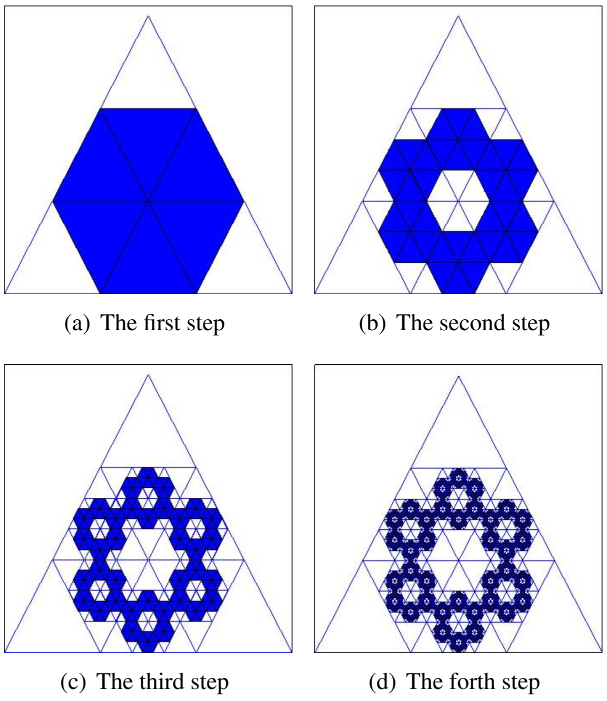

(a) The first step (b) The second step (c) The third step (d) The forth step

Figure 3: The first four steps of constructing general Sierpinski triangle $T(A_2, B_2)$.

When either of A and B is empty set, we can also obtain the fine structure of $k-$squares(resp. $k-$triangles) $T^k(A, B)$ according to the method presented in Example 3(resp. Example 4).

Example 3. *Let* $n = 4$,

$$A_3 = \{(0,0),(3,0),(1,1),(2,1),(1,2),(2,2),$$
$$(0,3),(3,3)\}, B_3 = \emptyset,$$

We can obtain the fine structures of the first four steps of constructing general Sierpinski carpet $T(A_3, B_3)$ *by running the following instructions in turn:*

$$Fractal\_square(E, F, 4, k), k = 1, 2, 3, 4,$$

where $E = [0, 3/4, 1/4, 1/2, 1/4, 1/2, 0, 3/4; 0, 0, 1/4, 1/4, 1/2, 1/2, 3/4, 3/4]$, $F = []$, *see Figure 4.*

Example 4. *Let* $n = 2$,

$$A_4 = \{(0,0),(1,0),(\frac{1}{2},\frac{\sqrt{3}}{2})\}, B_4 = \emptyset.$$

We can obtain the fine structures of the first four steps of constructing classic Sierpinski gasket $T(A_4, B_4)$ *by running the instructions in turn:*

$Fractal\_triangle([0, 1/2, 1/4; 0, 0, sqrt(3)/4], [], 2, k)$,

$k = 1, 2, 3, 4$, *see Figure 5.*

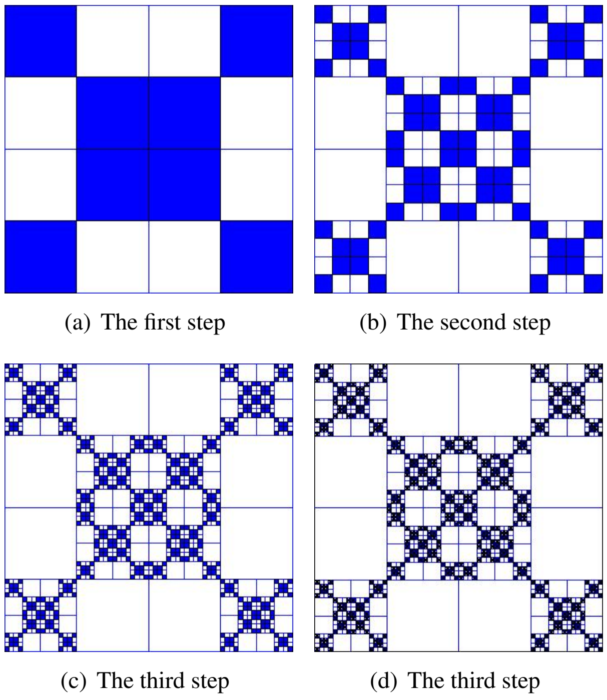

(a) The first step (b) The second step (c) The third step (d) The third step

Figure 4: The first four steps of constructing General Sierpinski carpet $T(A_3, B_3)$.

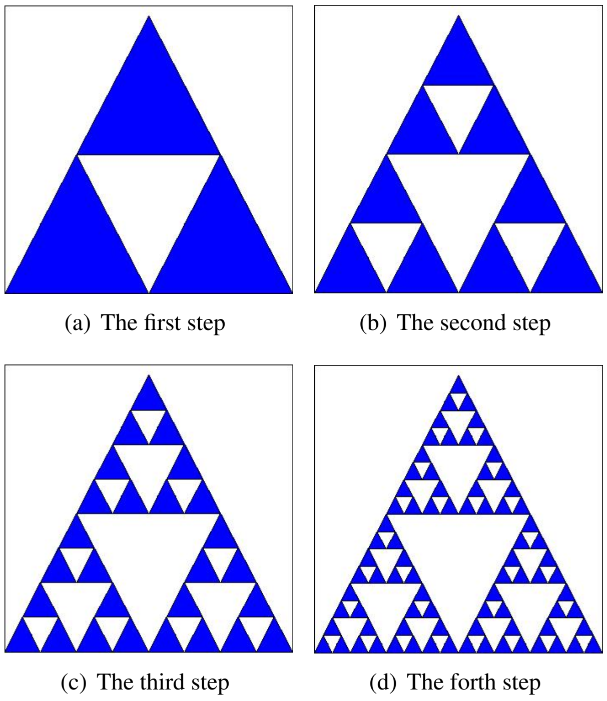

(a) The first step (b) The second step (c) The third step (d) The forth step

Figure 5: The first four steps of constructing Sierpinski gasket $T(A_4, B_4)$.

Remark 3. *Dividing each square(resp. equilateral triangle) chosen in each step for generating $T(A, B)$ into n^2 congruent squares(resp. triangles) will be helpful for researcher to investigate deeper the connection between higher and lower k-squares (resp. k-triangles). We can make the grid not shown in the figure by deleting the function of drawing grid in source programs presented as above when we don't want grid.*

Remark 4. *Our results in present paper include the results in our previous paper [22].*

4 Applying Random Iterated Algorithm to Generate General Sierpinski Fractal

Applying random iterated algorithm to generate fractal, the principle is clear and simple, it is easy to implement in any programming language and the results it generates are very spectacular and may be used for any graphical goal, not only for mathematical reasons. The method may be illustrated with the Yuval Fishers special copyrighter(see [20]) which receives as entry an arbitrary image (may be a point) and applies to it the set of affine transformation, generating a new image. The image obtained is transmitted, using a feedback process, on the entry of the copyrighter and the process is repeated for several times. For example, consider that the transformations are those which describe the Sierpinski gasket. If we test the Yuval Fisher copyrighter for different initial images we can observe that the final image is the same, so it not depends on the initial image but is defined by the affine transforms applied to it. It is one of the most used methods of generating a self-similar fractals. For creating $T(A, B)$ using random iterated algorithm steps given below should be considered.

Step 1: Choose $(x, y) = (0, 0)$ as starting point.

Step 2: Let (x_1, y_1) be the point obtained by applying a transformation in the IFS, where each transformation are chosen with probability $1/n$.

Step 3: Repeat the step 2 with (x_1, y_1) as initial point.

Step 4: Repeat step 3 again and again, this step 3 can be repeated infinite number of times.

4.1 Source Program for Creating $T(A, B)$ Using Random Iterated Algorithm

```
function Sierpinskifractal $(k, n, M, P)$
% SIERPINSKIFRACTAL: Drawing general Sierpinski carpet(resp. triangle) $T(A, B)$ by random iterated algorithm.
% Call format: Sierpinskifractal $(k, n, M, P)$.
% $k$ is the number of iteration.
% $n$ is the number of affine transformations in IFS.
% $M = [a_{11}, a_{12}, \cdots, a_{16}; a_{21}, a_{22}, \cdots, a_{26}; \cdots, a_{n1}, a_{n2}, \cdots, a_{n6}]$ is a $n \times 6$ array, where
    $[a_{i1}, a_{i2}, \cdots, a_{i6}] = [a_i, b_i, e_i, c_i, d_i, f_i]$
satisfying (2).
% $P = (p_1, p_2, \cdots, p_n)$ is a dimension array with $\sum_{i=1}^{n} p_i = 1$.
$x = 0$; $y = 0$; $r$ =rand$(1, k)$; $B$ =zeros$(2, k)$;
$w$ =zeros$(1, n)$;$w(1) = P(1)$;
for $i = 2 : n$
    $w(i) = w(i-1) + P(i)$;
end
$m = 1$;
for $i = 1 : k$
    for $j = 1 : n$
        if $r_i < w(j)$
            $a = M(j, 1); b = M(j, 2); e = M(j, 3); c = M(j, 4); d = M(j, 5); f = M(j, 6)$;
            break;
        end
    end
    $x = a * x + b * y + e; y = c * x + d * y + f$;
    $B(1, m) = x$; $B(2, m) = y$;
    $m = m + 1$;
end
plot($B(1, :)$,$B(2, :)$,'.','markersize',0.5)
set(gca,'xtick',[],'xticklabel',[]);
set(gca,'ytick',[],'yticklabel',[])
```

4.2 Comments and Examples

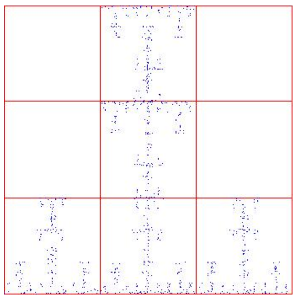
Remark 5. *We can't obtain the fine structure of $k-$squares(resp. $k-$triangles) by using random iterated algorithm. But we can obtain close approximation of general Sierpinski carpet(resp. general Sierpinski gasket) $T(A, B)$ faster than using deterministic algorithm and need lesser computer memory.*

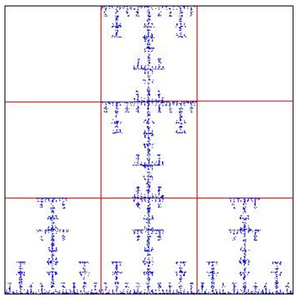
Remark 6. *For facilitating the readers to observe and understand, one may add the command lines which divide $[0, 1]^2$(resp. equilateral triangle) into n^2 congruent ones (see source programs in Section 3 or [22] for detail) at the beginning of program, the rest remain the same.*

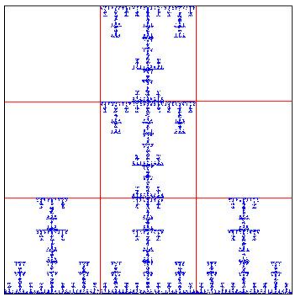
Example 5. *Let $n = 5$,*

$$A_5 = \{(0, 0), (1, 0), ((2, 0))\}, B_5 = \{(2, 2), (2, 3)\}.$$

We can obtain close approximation of general Sierpinski carpet $T(A_5, B_5)$ by calling the function: fractal $(k, 5, M, P)$ in the Matlab command window prompt, as shown in Figure 6. Table 1 lists corresponding transformations.


(a) After 1000 iterations


(b) After 5000 iterations


(c) After 10000 iterations

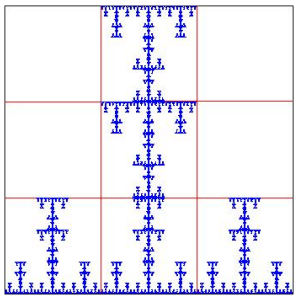

(d) After 50000 iterations

Figure 6: General Sierpinski carpet $T(A_5, B_5)$ generated using some iterations.

Table 1: $T(A_5, B_5)$

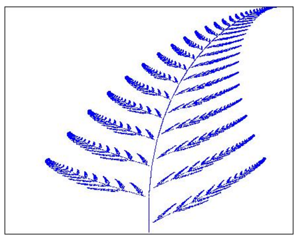
S_i	a	b	c	d	e	f	p
S_1	1/3	0	0	1/3	0	0	1/5
S_2	1/3	0	0	1/3	1/3	0	1/5
S_3	1/3	0	0	1/3	2/3	0	1/5
S_4	-1/3	0	0	-1/3	2/3	2/3	1/5
S_5	-1/3	0	0	-1/3	2/3	1	1/5

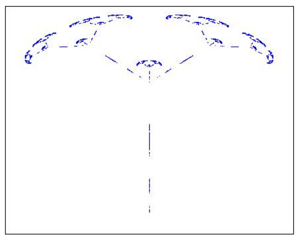
Remark 7. *This program can draw more fractals except the general Sierpinski fractals mentioned in this paper.*

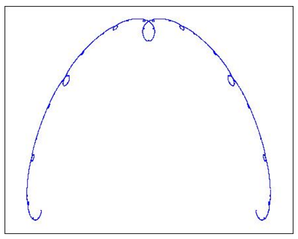
Example 6. *We can obtain four famous fractals as shown in Figure 7, the corresponding transformations are listed in Table 2-Table 5.*

Table 2: Bernsley fern leaf

S_i	a	b	c	d	e	f	p
S_1	0	0	0	0.16	0	0	0.01
S_2	0.85	0.04	-0.04	0.85	0	80	0.85
S_3	0.2	-0.26	0.23	0.22	0	80	0.07
S_4	-0.15	0.28	0.26	0.24	0	20	0.07


(a) Bernsley fern leaf


(b) Fractal tree


(c) Levy curve

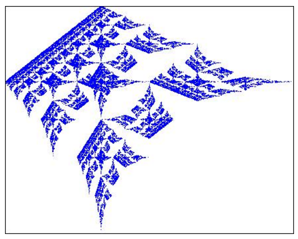

(d) Flamboyrent crown

Figure 7: Examples of Non-general Sierpinski fractals generated using 50000 iterations

Table 3: Fractal tree

S_i	a	b	c	d	e	f	p
S_1	0	0	0	0.5	0	0	0.05
S_2	0.42	-0.42	0.42	0.42	0	200	0.4
S_3	0.42	0.42	-0.42	0.42	0	200	0.4
S_4	0.1	0	0	0.1	0	200	0.15

Table 4: Levy curve

S_i	a	b	c	d	e	f	p
S_1	0.5	-0.5	0.5	0.5	0	0	0.05
S_2	0.5	0.5	-0.5	0.5	150	150	0.5

Table 5: Flamboyrent crown

S_i	a	b	c	d	e	f	p
S_1	0.25	0	0	0.5	0	0	0.154
S_2	0.5	0	0	0.5	-75	150	0.307
S_3	-0.25	0	0	-0.25	75	300	0.078
S_4	0.5	0	0	0.5	0	225	0.307
S_5	0.5	0	0	-0.25	150	375	0.154

5 Extension

It is well know that the topological structures of 3D general Sierpinski fractals become more challenging and intriguing than 2D general Sierpinski fractals. In this section, since IFS can be defined for any dimension, we shall consider the extension of our techniques to three dimensional general Sierpinski fractals. Fixing an integer $n \geq 2$, let $\mathcal{D} = \{0, 1, \cdots, n-1\}^3$. For $A \subset \mathcal{D}$, we assume that $1 < \sharp A < n^3$ to exclude the trivial case. Let $T(A) \subset \mathbb{R}^3$ be the unique non-empty compact set satisfying the following set equation:

$$T = (T + A)/n.$$

We shall call $T(A)$ a general Sierpinski sponge throughout this section. A familiar example of a general Sierpinski sponge is the Sierpinski sponge(see Figure 8), in which $n = 3$, $A = \mathcal{D} \backslash \{(1,1,0), (1,0,1$),(0,1,1), $(1,1,1), (2,1,1), (1,2,1), (1,1,2)\}\}$. Similarly, we define $T^0(A) = [0,1]^3$, $T^1(A) = (T^0(A) + A)/n$, and recurrently, $T^{k+1}(A) = (T^k(A) + A)/n$ for $k \geq 1$. Then $T^k(A)$ is a union of cubes of size $1/n^k$ (called them $k-$cubes). For simplify, we call T^k a $k-$cubes. Clearly $T^{k+1}(A) \subset T^k(A)$ and $T(A) = \cap_{k=1}^{\infty} T^k(A)$.

According to [3], we can obtain a general Sierpinski sponge $T(A)$ as follows: we divide initial cube $[0,1]^3$ into n^3 closed subcubes in the obvious manner, so that these subcubes have disjoint interiors and side length $1/n$. We choose some subcubes according to the rule described by A, again divide each of these subcubes into n^3 congruent ones, chose the subcubes according the same rule and repeat the procedure inductively to the infinity, then we get general Sierpinski sponge $T(A)$. For any $k \geq 1$, then $T^k(A)$ is the union of cubes that are chosen in the step k. Clearly, $T(A)$ and $T^k(A)$ are not of rotation structures mentioned as above. For $a \in A$, we define

$$S_a(x) = \frac{1}{n}(x + a). \tag{4}$$

Then $T(A)$ is unique invariant set of iterated function system $\{S_a\}_{a \in A}$.

In this section, we obtain the geometric modelling of such $T^k(A)$(or $T(A)$) by using two methods, which are the extension of our techniques used in two dimensional space. For conveniently, we shall call the outward lower-left vertex of cube the lower-left vertex throughout this section.

5.1 Source Program 1 for Creating $k-$ cubes $T^k(A)$ Using Deterministic Algorithm

Similar to the steps of creating $k-$squares $T^k(A, \emptyset)$ in [22], we shall consider the following steps of generating $k-$ cubes using deterministic algorithm.

Step 1: Select the lower-left vertex coordinate of initial cube it can be $(0, 0, 0)$.

Step 2: Compute the lower-left vertex coordinates of all $1-$cubes by applying transformations (4) on the lower-left vertex coordinate of initial cube.

Step 3: Again apply transformations (4) on the lower-left vertex coordinates of all new cubes obtained after applying transformations (4).

Step 4: Repeat step 3 again and again, this step 3 can be repeated infinite number of times.

Step 5: Create three empty arrays to store x, y and z coordinates of lower-left corner vertex of all $k-$cubes respectively.

Step 6: Draw $k-$ cubes by patch function which is the low-level graphics function for creating patch graphics objects in Matlab.

We implement the above algorithm in Matlab programming language as follows:

```
function Sierpinskicube1 $(M, x, y, z, d, n, k)$
% SIERPINSKICUBE1: Drawing general Sierpinski sponge using deterministic iterated algorithm.
% Call format: Sierpinskicube1 $(M, x, y, z, d, n, k)$.
% $M = [a_1, \cdots, a_m; b_1, \cdots, b_m; c_1, \cdots, c_m]$, where $(a_i/n, b_i/n, c_i/n)$, $i = 1, 2, \cdots, m(1 < m < n^3)$, are the lower-left coordinates of small cubes chosen according to certain rule after the first division for the initial cube.
% $x$ is the $x-$coordinate of lower-left corner vertex of initial cube.
% $y$ is the $y-$coordinate of lower-left corner vertex of initial cube.
% $z$ is the $z-$coordinate of lower-left corner vertex of initial cube.
% $d$ is the side length of initial cube.
% $1/n$ is the contraction ratio.
% $k$ is the number of iteration.
axis$([x, x+d, y, y+d, z, z+d])$
drawcube$(x, y, z, d)$
for $j = 1 : k$
    $a1 = []; b1 = []; c1 = [];$ % Three empty arrays, will be used to store the x, y and z coordinates of the lower-left corner of chosen cubes after the kth step construction respectively.
    for $i = 1$ :length$(x)$
        $x1 = x(i) + d/n * M(1,:);$
        $y1 = y(i) + d/n * M(2,:);$
        $z1 = z(i) + d/n * M(3,:);$
        $a1 = [a1, x1]; b1 = [b1, y1]; c1 = [c1, z1];$
    end
    $d = d/n; x = a1; y = b1; z = c1;$
end
for $i = 1$ :length$(x)$
    patch$(x(i) + [d, d, d, d, d], y(i) + [0, d, d, 0, 0],$
              $z(i) + [0, 0, d, d, 0], z(i) + [0, 0, d, d, 0])$
    patch$(x(i) + [0, d, d, 0, 0], y(i) + [0, 0, 0, 0, 0],$
              $z(i) + [0, 0, d, d, 0], y(i) + [0, 0, 0, 0, 0])$
    patch$(x(i) + [0, d, d, 0, 0], y(i) + [d, d, d, d, d],$
        z(i)+[0,0,d,d,0],x(i)+[0,d,d,0,0])
    patch(x(i) + [0,d,d,0,0],y(i) + [0,0,d,d,0],
        z(i)+[0,0,0,0,0],y(i)+[0,0,d,d,0])
    patch(x(i) + [0,d,d,0,0],y(i) + [0,0,d,d,0],
        z(i)+[d,d,d,d,d],x(i)+[0,0,0,0,0])
    patch(x(i) + [0,0,0,0,0],y(i) + [0,d,d,0,0],
        z(i)+[0,0,d,d,0],z(i)+[0,0,d,d,0])
end
axis equal
axis off
set(gcf,'color',[1,1,1])
function drawcube(x,y,z,d)
u =([0 1 1 0 0 0;1 1 0 0 1 1;1 1 0 0 1 1;0 1 1 0 0
0])*d + x(1);
v =([0 0 1 1 0 0;0 1 1 0 0 0;0 1 1 0 1 1;0 0 1 1 1
1])*d + y(1);
w =([0 0 0 0 0 1;0 0 0 0 0 1;1 1 1 1 0 1;1 1 1 1 0
1])*d + z(1);
for i = 1 : 6
    h =patch(u(:,i),v(:,i),w(:,i),'w');
    set(h,'edgecolor','k','facealpha',0.5)
end
h=gcf;
view(-33,18)
```

Example 7. *Let* $n = 3$,

$$A_1 = \{(0,0,0),(\frac{1}{3},0,0),(\frac{2}{3},0,0),(0,\frac{1}{3},0)$$
$$(\frac{2}{3},\frac{1}{3},0),(0,\frac{2}{3},0),(\frac{1}{3},\frac{2}{3},0),(\frac{2}{3},\frac{2}{3},0)$$
$$(0,0,\frac{1}{3}),(\frac{2}{3},0,\frac{1}{3}),(0,\frac{2}{3},\frac{1}{3}),(\frac{2}{3},\frac{2}{3},\frac{1}{3})$$
$$(0,0,\frac{2}{3}),(\frac{1}{3},0,\frac{2}{3}),(\frac{2}{3},0,\frac{2}{3}),(0,\frac{1}{3},\frac{2}{3})$$
$$(\frac{2}{3},\frac{1}{3},\frac{2}{3}),(0,\frac{2}{3},\frac{2}{3}),(\frac{1}{3},\frac{2}{3},\frac{2}{3}),(\frac{2}{3},\frac{2}{3},\frac{2}{3})\}$$

We can obtain the fine structures of first three steps of constructing Sierpinski sponge $T(A_1)$ by running the following instructions:

$$\textit{Sierpinskicube}1(M,x,y,z,1,3,k),\ \ k = 1,2,3,$$

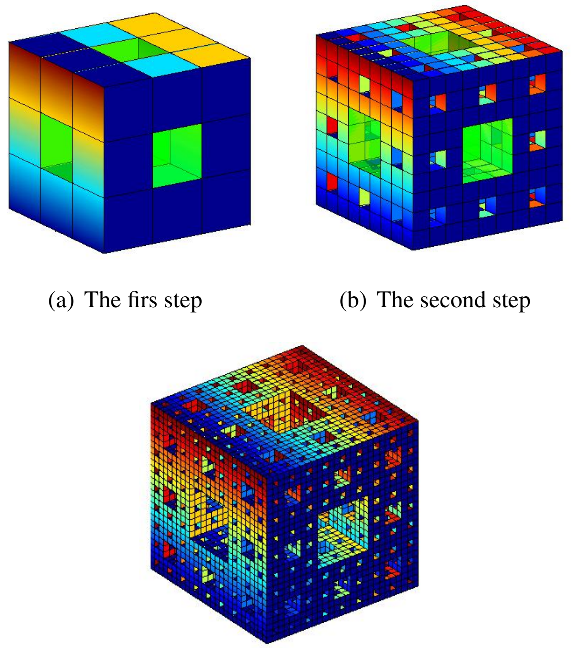
where

$$M = [0\,1\,2\,0\,2\,0\,1\,2\,0\,2\,0\,2\,0\,1\,2\,0\,2\,0\,1\,2;$$
$$0\,0\,0\,1\,1\,2\,2\,2\,0\,0\,2\,2\,0\,0\,0\,1\,1\,2\,2\,2;$$
$$0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,2\,2\,2\,2\,2\,2\,2\,2],$$

as shown in Figure 8.

Example 8. *Let* $n = 3$,

$$A_2 = \{(0,0,0),(0,\frac{2}{3},0),(\frac{2}{3},0,0),(\frac{2}{3},\frac{2}{3},0)$$
$$(0,0,\frac{2}{3}),(0,\frac{2}{3},\frac{2}{3}),(\frac{2}{3},0,\frac{2}{3}),(\frac{2}{3},\frac{2}{3},\frac{2}{3})\}$$



(a) The firs step (b) The second step

(c) The third step

Figure 8: The first three steps of constructing Sierpinski sponge $T(A_1)$.

We can obtain the fine structures of first four steps of constructing 3D Cantor dust $T(A_2)$ by running the following instructions:

$$\textit{Sierpinskicube}1(M,x,y,z,1,3,k),\ \ k = 1,2,3,4,$$

where

$$M = ([0\,0\,2\,2\,0\,0\,2\,2;0\,2\,0\,2\,0\,2\,0\,2;0\,0\,0\,0\,2\,2\,2\,2],$$

as shown in Figure 9.

Example 9. *Let* $n = 4$,

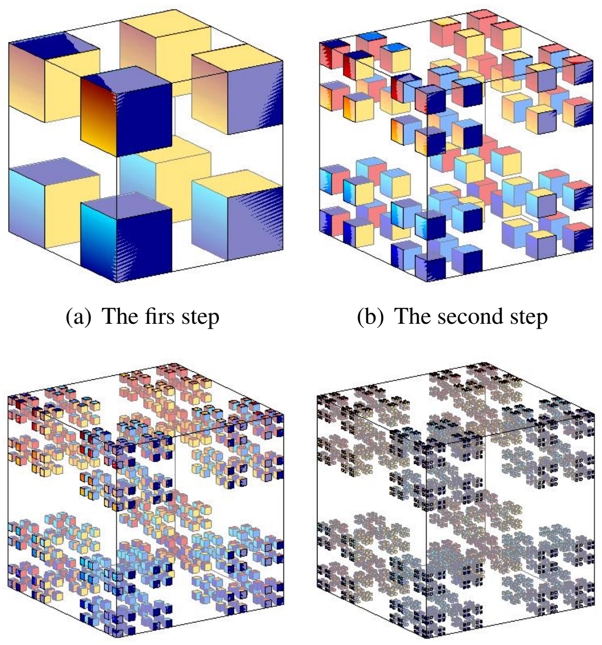
$$A_3 = \{(0,0,0),(0,\frac{3}{4},0),(\frac{3}{4},0,0),(\frac{3}{4},\frac{3}{4},0)$$
$$(\frac{1}{4},\frac{1}{4},\frac{1}{4}),(\frac{1}{4},\frac{2}{4},\frac{1}{4}),(\frac{2}{4},\frac{1}{4},\frac{1}{4}),(\frac{2}{4},\frac{2}{4},\frac{1}{4})$$
$$(\frac{1}{4},\frac{1}{4},\frac{2}{4}),(\frac{1}{4},\frac{2}{4},\frac{2}{4}),(\frac{2}{4},\frac{1}{4},\frac{2}{4}),(\frac{2}{4},\frac{2}{4},\frac{2}{4})$$
$$(0,0,\frac{3}{4}),(0,\frac{3}{4},\frac{3}{4}),(\frac{3}{4},0,\frac{3}{4}),(\frac{3}{4},\frac{3}{4},\frac{3}{4})\}.$$

We can obtain the fine structures of first three steps of constructing $T(A_3)$ by running the following instructions:

$$\textit{Sierpinskicube1}(M,x,y,z,1,4,k),\ \ k = 1,2,3,$$

where

$$M = [0\,0\,3\,3\,1\,1\,2\,2\,1\,1\,2\,2\,0\,0\,3\,3;$$
$$0\,3\,0\,3\,1\,2\,1\,2\,1\,2\,1\,2\,0\,3\,0\,3;$$
$$0\,0\,0\,0\,1\,1\,1\,1\,2\,2\,2\,2\,3\,3\,3\,3],$$


(a) The firs step (b) The second step

(c) The third step (d) The forth step

Figure 9: The first four steps of constructing 3D Cantor dust $T(A_2)$.

as shown in figure 10.

5.2 Source Program 2 for Creating $k-$ cubes $T^k(A)$ Using Deterministic Algorithm

For given rule $A \subset \mathcal{D} = \{0, 1, \cdots, n-1\}^3$. Note that the cube has good symmetry, so it is more easier to describe this rule by a three dimensional array with elements 0 or 1. For example, for Sierpinski sponge, if we denote the chosen cubes by 1, the others by 0, then we can use a three-dimensional array of Matlab to describe the corresponding rule as follows:

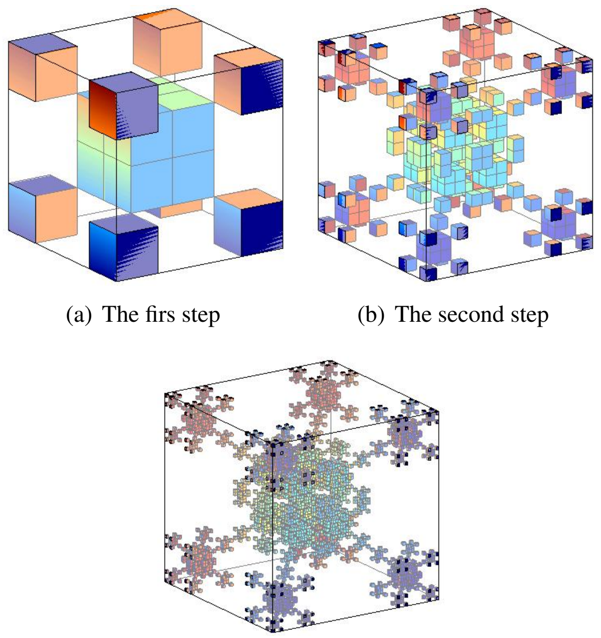
$$M(:,:,1) = \begin{matrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{matrix}$$

$$M(:,:,2) = \begin{matrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{matrix}$$

$$M(:,:,3) = \begin{matrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{matrix}.$$

So we can also use the following method for generating $k-$cubes $T^k(A)$. The general algorithm is:

Step 1: Select the lower-left vertex coordinate of initial cube it can be $(0, 0, 0)$.


(a) The firs step (b) The second step

(c) The third step

Figure 10: The first three steps of constructing $T(A_3)$

Step 2: Describe the rule by a three-dimensional array.

Step 3: Draw initial cube by patch function of Matlab.

Step 4: Traversal the array in step 2. If encounter 0, draw cube with size $1/n$ in the corresponding place with green using patch function of Matlab. If encounter 1, assign the lower-left coordinate and side length of corresponding small cube to the variables of lower-left coordinate and side length of initial square by applying transformations (4).

Step 5: For each small cube, execute the step 1 to 4.

Step 6: Repeat the step 5 again and again. If iterated depth is k, then the step 5 can be repeated k times.

Now, we implement the above algorithm in Matlab programming language as follows:
function Sierpinskicube2$(X_1, X_2, \cdots, X_n, t, d, h)$
% SIERPINSKICUBE2: Drawing $k-$cubes $T^k(A)$ using deterministic iterated algorithm.
% Call format:

$$\textit{Sierpinskicube2}(X_1, X_2, \cdots, X_n, t, d, h).$$

% $X_1, X_2, \cdots, X_n$ are all $n \times n$ dimensional arrays, in which each element is o or 1.
% t is the vertex coordinate of lower-left corner of initial cube.
% d is the side length of initial cube.

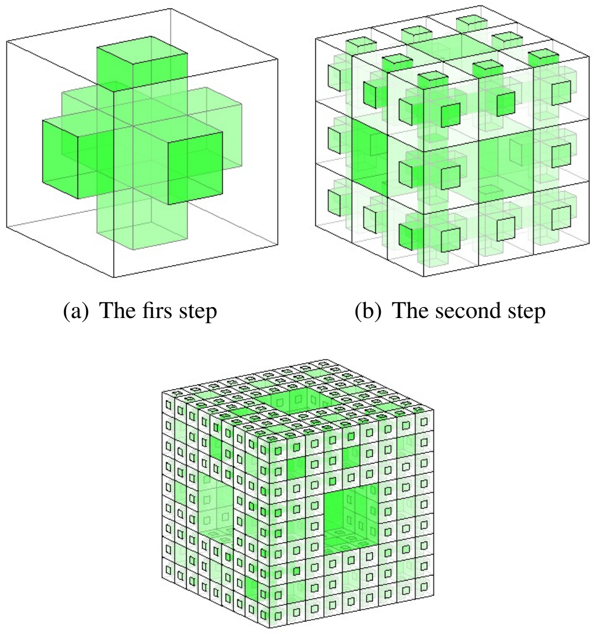
```
% $h$ is the iterated depth.
$M$ =cat($3, X_1, X_2, \cdots, X_n$);
$u = t(1)$;$v = t(2)$;$w = t(3)$;$l = d$;
if $h == 0$

    return
end
axis($[u, u+l, v, v+l, w, w+l]$)
drawwhite(t,d)
$m$ =length($M$);
for $i = 1 : m$
    for $j = 1 : m$
        for $k = 1 : m$
            if $M(i, j, k) == 0$
                drawcube($[t(1) + d*(k-1)/m, t(2)+d*(j-1)/m, t(3)+d*(i-1)/m], d/m$)
            else
                Sierpinskicube2($X_1, X_2, \cdots, X_n, [t(1)+d*(k-1)/m, t(2)+d*(j-1)/m, t(3)+d*(i-1)/m], d/m, h-1$)
            end
        end
    end
end
axis equal
axis off
function drawcube($t, d$)
$x$ =([0 1 1 0 0 0;1 1 0 0 1 1;1 1 0 0 1 1;0 1 1 0 0 0])$*d + t(1)$;
$y$ =([0 0 1 1 0 0;0 1 1 0 0 0;0 1 1 0 1 1;0 0 1 1 1 1])$*d + t(2)$;
$z$ =([0 0 0 0 0 1;0 0 0 0 0 1;1 1 1 1 0 1;1 1 1 1 0 1])$*d + t(3)$;
for $i = 1 : 6$
    $s$ =patch($x(:, i), y(:, i), z(:, i)$,'$g$');
    set(s,'$edgecolor$','k','$facealpha$',0.5)
end
$s$ =gcf;
view($-33, 18$)
function drawwhite($t, d$)
$x$ =([0 1 1 0 0 0;1 1 0 0 1 1;1 1 0 0 1 1;0 1 1 0 0 0])$*d + t(1)$;
$y$ =([0 0 1 1 0 0;0 1 1 0 0 0;0 1 1 0 1 1;0 0 1 1 1 1])$*d + t(2)$;
$z$ =([0 0 0 0 0 1;0 0 0 0 0 1;1 1 1 1 0 1;1 1 1 1 0 1])$*d + t(3)$;
for $i = 1 : 6$
    $s$ =patch($x(:, i), y(:, i), z(:, i)$,'$w$');
    set($s$,'edgecolor','$k$','facealpha',0.5)
end
$s$ =gcf;
view($-33, 18$)
```

Example 10. *We can obtain the fine structures of first three steps of constructing Sierpinski sponge that are different from Example 8 by running the following instructions:*

$$\textit{Sierpinskicube2}(X, Y, Z, [0\,0\,0], 1, h), h = 1, 2, 3,$$

where $X = [1\,1\,1; 1\,0\,1; 1\,1\,1]$, $Y = [1\,0\,1; 0\,0\,0; 1\,0\,1]$, $Z = [1\,1\,1; 1\,0\,1; 1\,1\,1]$, *as shown in Figure 11.*


(a) The firs step (b) The second step

(c) The third step

Figure 11: The first three steps of constructing Sierpinski sponge

Similarly, we also can obtain the different fine structures with Example 9 and Example 10. If we get rid of the step 2 in above algorithm(delete the 8th line 'drawcube(t, d)' in source program), we can get the image of complementary set of general Sierpinski sponge in $[0, 1]^3$(sometime, the structure of complementary set is important to us when we study fractal theory).

Example 11. *We can get the images of complementary sets of Sierpinski sponge and 3D Cantor dust generated using 3 iterations by running the following instructions:*

$$\textit{Sierpinskicube2}(X_i, Y_i, Z_i, [0\,0\,0], 1, 3), i = 1, 2,$$

where X_1, Y_1, Z_1 *are same as that of Example 10,* $X_2 = [1\,0\,1; 0\,0\,0; 1\,0\,1], Y_2 = [0\,0\,0; 0\,0\,0; 0\,0\,0], Z_2 = [1\,0\,1; 0\,0\,0; 1\,0\,1]$ *, as shown in Figure 12 .*

An example in case $n = 4$ as follows:

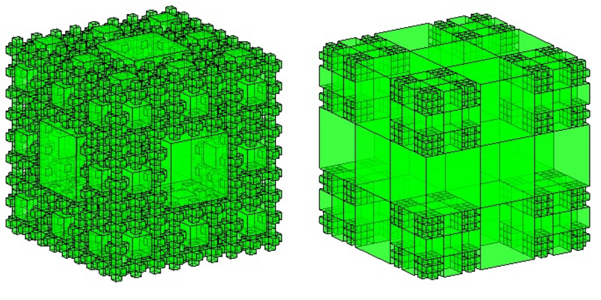


Figure 12: (a) Complementary set of Sierpinski sponge generated using 3 iterations;(b) Complementary set of 3D Cantor dust generated using 3 iterations.

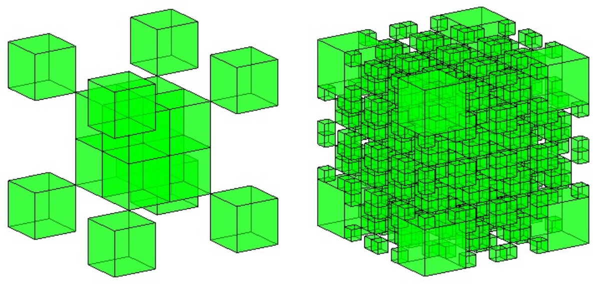


Figure 13: The first two steps for constructing General Sierpinski sponge $T(\mathcal{D} \setminus A_4)$.

Example 12. *Let*

$$\begin{aligned}A_4 = \{&(0,0,0), (\frac{3}{4},0,0), (0,\frac{3}{4},0), (\frac{3}{4},\frac{3}{4},0)\\ &(\frac{1}{4},\frac{1}{4},\frac{1}{4}), (\frac{1}{4},\frac{2}{4},\frac{1}{4}), (\frac{2}{4},\frac{1}{4},\frac{1}{4})(\frac{2}{4},\frac{2}{4},\frac{1}{4})\\ &(\frac{1}{4},\frac{1}{4},\frac{2}{4}), (\frac{1}{4},\frac{2}{4},\frac{2}{4}), (\frac{2}{4},\frac{1}{4},\frac{2}{4})(\frac{2}{4},\frac{2}{4},\frac{2}{4})\\ &(0,0,0), (\frac{3}{4},0,0), (0,\frac{3}{4},0), (\frac{3}{4},\frac{3}{4},0)\}.\end{aligned}$$

Then we can obtain the images of first two steps for constructing $T(\mathcal{D} \setminus A_4)$(see Figure 13) by running the following instructions:

$$\textit{Sierpinskicube2}(X, Y, Z, W, [0\,0\,0], 1, h), h = 1, 2,$$

where

$$\begin{aligned}X &= [0\,1\,1\,0; 1\,1\,1\,1; 1\,1\,1\,1; 0\,1\,1\,0]\\ Y &= [1\,1\,1\,1; 1\,0\,0\,1; 1\,0\,0\,1; 1\,1\,1\,1]\\ Z &= [1\,1\,1\,1; 1\,0\,0\,1; 1\,0\,0\,1; 1\,1\,1\,1]\\ W &= [0\,1\,1\,0; 1\,1\,1\,1; 1\,1\,1\,1; 0\,1\,1\,0].\end{aligned}$$

Remark 8. *In order to get the better visual effect for application, we also can integrate the above two source programs as needed.*

6 Conclusions

Geometric construction of fractal image with IFS begins with original image and some successive transformation are applied over the image. This paper presents the application of theory of IFS in geometric modelling of general Sierpinski fractals and their geometric constructions base on Matlab environment. We can obtain the fine structure of each step which generates 2D general Sierpinski fractal 180° rotation structure with by using deterministic algorithm. Using random iteration algorithm, we can obtain close approximation of fractal image faster than using deterministic algorithm and need lesser computer memory. Since IFS can be defined for any dimension, we can extend our techniques to three dimensional objects. We obtain the geometric modelling of general Sierpinski sponges with no rotation structure and their geometric construction. This enables also generation of more beautiful images. But the geometric modelling of geometric construction of general Sierpinski sponge with 180° rotation structure and general Sierpinski carpet(resp. gasket, sponge) with other rotation structure are difficult for us now. That is what we are going to do next.

Acknowledgements: This research was supported by China National Research Foundation(CSC scholarship), Chinese Universities Scientific Fund (grant No. Z109021203), Teaching Reform Project of Northwest A&F University (grant No. JY1501001-1).

References:

[1] A. Garg, A. Negi, A. Agrawal and B. Latwal, *Geometric modelling of complex objects using iterated function system*, International Journal of Scientific & Technology Research, Volume 3, Issue 6, 2014(6), pp. 1-8.

[2] M.-F. Barnsley, S.-G. Demko, *Iterated Function Systems and the Global Construction of Fractals*, Proc. Roy. Soc. London, Ser.A 399, 1985, pp. 243-275.

[3] G. David and S.Semmes, *Fractured fractals and broken dreams: Self-similar geometry through metric and measure*, Oxford Lecture Series in Mathematics and its Applications, vol. 7. The Clarendon Press Oxford University Press, New York (1997).

[4] F. Deng and F.-L. Xi, *An Application of Lsystem and IFS in 3D Fractal Simulation*, WSEAS Transactions on Systems. 4, 2008, pp. 352-361.

[5] K.-J. Falconer, *Techniques in Fractal Geometry*, John Wiley & Sons, Ltd., Chichester, 1997.

[6] K.-J. Falconer, *Fractal Geometry - Mathematical Foundations and Applications*, John Wiley & Sons, Ltd., Chichester, England, 1990.

[7] S. Demko, L. Hodges and D. Naylor, *Construction of fractal objects with iterated function systems*, San Francisco, July, pp. 22-26.

[8] Y.-X. Gui and W.-X. Li, *A generalized multifractal spectrum of the general sierpinsk carpets*, J. Math. Anal. Appl. 348, 2008, pp. 180-192.

[9] J.-E. Hutchinson, *Fractals and self-similarity*, Indiana Univ. Math. J., 30 (1981), pp. 713-747.

[10] B.-G. Jia, *Bounds of Hausdorff measure of the Sierpinski gasket*, J. Math. Anal. Appl. 330, 2007, pp. 1016-1024.

[11] K.-S. Lau, J.-J. Luo and H. Rao, *Topological structure of fractal squares*, Proceedings of the Edinburgh Mathematical Society. 155(1), 2013, pp. 73-86.

[12] Q.-H. Liu, L.-F. Xi and Y-.F. Zhao, *Dimensions of intersections of the Sierpinski carpet with lines of rational slopes*, Proceedings of the Edinburgh Mathematical Society. 50, 2007, pp. 411-448.

[13] Mathworks, *Matlab:The language of technical computing*, version 6.5, 2002.

[14] Slawomir S.Nikiel, *True-colour images and iterated function systems*, Computer & Graphics, 5(22), 1998, pp. 635-640.

[15] T.-D. Taylor, *Connectivity properties of Sierpinski relatives, Fractals*, 19(4), 2011, pp. 481-506.

[16] H.-P. Wang, *Research on dynamic simulation method of plants based on arithmetic of IFS*, J.Changchun Inst.The.(Nat.Sci.Edi),6(2), 2005, pp. 49-52.

[17] Z.-X. Wen, Z.-Y. Zhu and G.-T. Deng, *Lipschitz equivalence of a class of general Sierpinski carpets*, J. Math. Anal. Appl., 385 (2012), pp. 16-23.

[18] L.-F. Xi, and Y. Xiong, *Self-similar sets with initial cubic patterns*, C. R. Math. Acad. Sci. Paris, 348 (2010), pp.15-20.

[19] W.-Q. Zeng and X.-Y. Wang, *Fractal theory and its computer simulation*, Northeastern university press, Shenyang, China, 2001.

[20] Y. Fisher, E.W. Jacobs, R. D. Boss, *Fractal image compression using iterated transformes*, NOSC Techical Report, Naval Ocean System Center, San Diego, 1995.

[21] Z.-L. Zhou and F. Li, *A new estimate of the Hausdorff measure of the Sierpinski gasket*, Nonlinearity. 13(3), 2000, pp. 479-491.

[22] Z.-Y. Zhu and E.-M. Dong, *Simulation of Sierpinski-type fractals and their geometric constructions in Matlab environment*, Wseas Transactions on Mathematics, 12(10), 2013, pp. 2224-2880.

[23] Z.-Y. Zhu and E.-M. Dong, *Lipschitz equivalence of fractal triangles*, J. Math.Anal.Appl., 433(2016), pp. 1157-1176.

[24] Z.-Y. Zhu, *Lipschitz equivalence of totally disconnected general Sierpinski triangles*, Fractals, Vol. 23, No. 2(2015), pp. 1550013 (14 pages).

[25] Z.-Y.Zhu, Y. Xiong and L.-F. Xi, *Lipschitz equivalence of self-similar sets with triangular pattern*, Sci. China. Ser. A, 54(2011), pp. 1019-1026.