



Article

Elementary Fractal Geometry 7: An Algorithm Visualizing the Magnification Flow

Christoph Bandt 

Institute of Mathematics, University of Greifswald, 17487 Greifswald, Germany; bandt@uni-greifswald.de

Abstract

Self-similar sets are defined as compact attractors. Their interior structure, however, is revealed only by successive magnification. This requires the use of a computer. We describe a new and very simple algorithm which performs an unlimited magnification of self-similar sets. In the case of integer data, it works without numerical error propagation. In other words, the program visualizes the scenery flow introduced by Bedford and Fisher, and Hochman. An open-source MATLAB implementation of the program is provided as a starting point and invitation to computational work in fractals.

Keywords: self-similar; scenery flow; fractal algorithm; visualization

MSC: 28A80; 52-08; 68U05

1. Introduction

Self-similar sets. A compact non-empty set A in Euclidean $\mathbb{R}^d$ is called self-similar if it is a finite union of pieces A_j which are geometrically similar to A [1]. Using contractive similitudes f_j with $A_j = f_j(A)$, Hutchinson (cf. [2–5]) has characterized these sets as unique solutions of the equation

$$A = \bigcup_{j=1}^m f_j(A). \quad (1)$$

The set $F = \{f_1, \dots, f_m\}$ is called an iterated function system or IFS, and A is the attractor of F . A contractive similitude from Euclidean $\mathbb{R}^d$ to itself fulfils $|f(x) - f(y)| = r|x - y|$ where the constant $r < 1$ is called the factor of f . We assume that all maps in F have the same factor r . In that case, we can write $f_j = g^{-1}h_j$ where g^{-1} is a contractive linear map and $h_j(x) = s_j(x) + v_j$ is an isometry of $\mathbb{R}^d$. Here, s_j denotes a linear map and v_j a translation vector. With this notation, Equation (1) is turned into the so-called numeration from

$$g(A) = \bigcup_{j=1}^m h_j(A). \quad (2)$$

The maps h_j and the matrix of g are often considered as the digits and base of a numeration system. For the decimal system, we have $g(x) = 10x$ and $h_j(x) = x + j - 1$ for $j = 1, \dots, 10$, and $A = [0, 1]$. We can assume that $v_1 = 0$. (Otherwise, replace v_k by $\tilde{v}_k = v_k - (g - s_k)(g - s_1)^{-1}v_1$ and consider the translated attractor $\tilde{A} = \varphi(A)$ with $\varphi(x) = x - (g - s_1)^{-1}v_1$, cf. ([6] Section VI.A).) That is, $f_1(0) = h_1(0) = 0$, so that zero is in A and can be taken as a ‘control point’ of the attractor. The idea of control points goes back to Thurston [7] and plays an important part in this paper.



Academic Editor: María A. Navascués

Received: 29 October 2025

Revised: 17 November 2025

Accepted: 18 November 2025

Published: 24 November 2025

Citation: Bandt, C. Elementary Fractal Geometry. 7. An Algorithm Visualizing the Magnification Flow. *Fractal Fract.* **2025**, *9*, 762. <https://doi.org/10.3390/fractalfract9120762>

Copyright: © 2025 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Motivation. With these assumptions and additional properties, a comprehensive theory of the measure and dimension of self-similar fractals was established [2,4,5,8]. Nevertheless, there are few concrete examples of self-similar sets in the literature. Most of them, like the Sierpiński gasket, fractal squares, and post-critically finite fractals, have a fairly simple structure, or, like Barnsley’s fern [3], approximate natural phenomena with an unclear mathematical structure. The lack of examples has led to the widespread belief that magnifications of a self-similar set all look more or less the same.

This is not true. In the paper [6] with Mekhontsev in 2018, we have shown that even simple modifications of the Sierpiński gasket have a very rich interior structure which in the presence of a separation condition can be described by finite-state automata. Without a computer, this structure cannot be recognized. Mekhontsev [9] built the free IFStyle package with excellent graphics which visualizes and classifies thousands of examples within minutes. It has not been generally used, however, since it is sophisticated and not open source. There is a definitive need for transparent software in fractal geometry.

Moreover, all traditional magnification algorithms for self-similar sets are limited by two constraints:

- (i) *Bounded number of digits.* Suppose the diameter of A is one, and we use floating-point numbers with 16 decimal digits. Then, the magnification of A by the factor 10^{17} is not possible, since 10^{-17} equals zero in the computer.
- (ii) *Numerical error propagation.* Each magnification step is expressed by certain operations, such as multiplication. If magnification steps are iterated, the inaccuracies of the operations will accumulate.

Main contribution. In this note, we introduce the principle of virtual magnification, which solves problem (i) and speeds up the magnification procedure. The basic idea is to calculate with control points and directions of pieces instead of all their points. The principle works for arbitrary self-similar sets without the assumption of a separation condition.

Moreover, our algorithm will magnify without numerical error propagation and thus also solve problem (ii) when we have *integer data* in (2). That is, g and the s_j must be given by integer matrices, and the v_j must have integer coordinates. The s_j must belong to a finite group S of linear isometries such that $gS = Sg$. The v_j could be rational, since the multiplication of all v_j by the common denominator would change the size but not the shape of A . Most concrete examples in the literature fulfill these conditions.

The algorithm was implemented in MATLAB R2020b as an open-source program with 100 lines of code, which are appended as Supplementary Materials to this paper. The program was successfully tested with several other MATLAB versions as well as with Octave. It visualizes the interior structure of self-similar sets as a slow-motion film. For certain IFSs with integer data and open set condition, including most examples from the literature, you can run it almost as long as you want. Otherwise, a decisive constraint is the memory and speed of the computer. There is much room for improvement.

Exploration with our program shows that except for the most simple cases, self-similar attractors have a very rich local structure. An IFS does not just generate a compact set A , it also generates a collection of photos of small places in A . The theory of scenery flow and uniform scaling, discussed in Section 3, says that magnification at different spots will lead to the same infinite collection although in different order. Our program is a practical way to sample photos from an IFS, showing the importance of the scenery flow.

Contents of the paper. Our program is written for attractors A in the plane. Extensions are discussed in Section 7. The simplest examples come from $g = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$ and $s_j \in \{\pm 1, \pm s\}$ with $s = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$. While the program is based on matrix calculations, notation in the text will

often use complex numbers, such as $g = 2$, $s = i$, and $v = x + iy$ instead of $v = \begin{pmatrix} x \\ y \end{pmatrix}$. The IFS data for Figure 1 are

$$g(z) = 2, h_1(z) = iz, h_2(z) = iz - 1 + 3i, h_3(z) = z + 3 + 2i. \quad (3)$$

For this example, the function of the program will be explained in the next section. In Section 3, we discuss the well-known concept of scenery flow and our discrete version for self-similar sets. We then formulate our principle of virtual magnification and operate with control points and neighbor maps. In Section 5, we explain the structure of the program. Our implementation puts transparency and simplicity over the efficiency and quality of figures. In the two final sections, we discuss the example data, possible extensions of the program, and open problems. We hope that readers will feel encouraged to conduct their own experiments. The study of fractal structures requires explorative computer work.

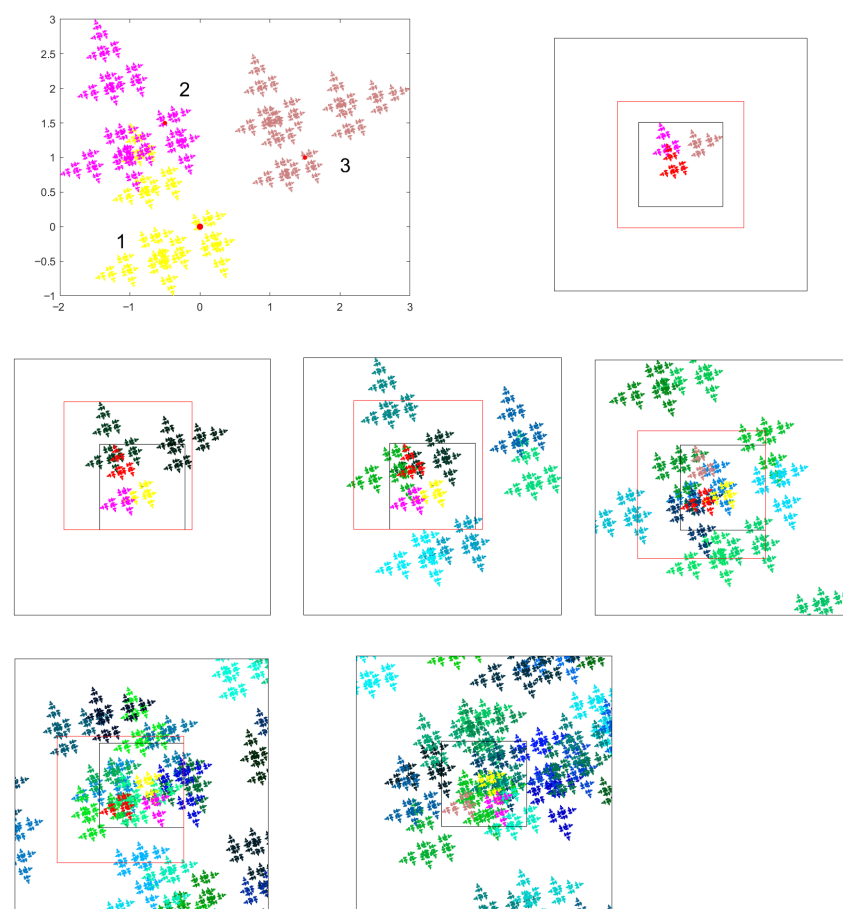


Figure 1. (Upper row) Global view of a self-similar set with control point at (0,0). The set A is the first reference piece which is placed in the small square in the middle of the screen with a control point at the center and subpieces colored yellow, pink, and brown. Then, one subpiece is chosen as the next reference piece and colored red. The red subpiece will go to the central piece with light colors, and the red frame will go to the outer frame in the next magnification. (Middle row) The new reference piece is in the small square, with colored subpieces; one of them is chosen as next reference and colored red. Other pieces are given dark shades of color, randomly chosen, without distinction of subpieces. Points outside the red frame will disappear in the next magnification. If we choose part 3 as a red subpiece, the direction of the reference piece will not change. Choice of part 1 or 2 will induce a rotation of the reference piece by 90° , according to (3). (Lower row) Our choice of subpieces 1, 3, 3, 2, 3 leads to an unexpectedly dense pattern where 45 neighbor pieces intersect the window.

2. A Preview of the Program

Our MATLAB program `scenery.m` and a number of data files `*.mat` are the Supplementary Materials of this paper. If you have access to MATLAB or the non-commercial version Octave, you can start it from one of the data files that contain the IFS matrices g and H . The example in Figure 1 is obtained from loading the file `B1-figure1.mat` and executing the command `scenery(g,H,6,5,w(1,:))`. Details will be given in Section 5.

First, the fractal A is shown in full scale in a coordinate system in the upper left of Figure 1. If there are three pieces, the first is yellow, the second pink, and the third brown. We also draw the control point 0 as a red spot. Since we assume $f_1(0) = 0$, the control point of A_1 agrees with that of A , while the control points of A_2, A_3 are $f_2(0), f_3(0)$, respectively. In the figure, A looks like a Cantor set, but the pieces A_1 and A_2 do intersect, and this is a place where magnification gives more information.

Next, A is moved to the inner subsquare of a square screen in such a way that the control point of A coincides with the center of the square. One of the pieces is selected and colored red. On the upper right of Figure 1 is A_1 , which was yellow before. The red subpiece is the central piece in the next magnification. The red square indicates the window of this magnification. On the left of the middle row, we have drawn A_1 as the central piece with three subpieces. The sets A_2, A_3 are now neighbor pieces with random shades of dark color. Note that the central piece has the same control point as A but another orientation. The brown subpiece of the central piece is already colored red in order to show that the third part A_{13} of A_1 is the next central piece. The piece A_{33} of the right neighbor is outside the red square and will not be seen in the next magnification.

In the picture in the middle, A_{13} is the central reference piece, and its third subpiece A_{133} is selected for the next step and colored red. The blue neighbors below the central piece are A_{12} and A_{11} . Only a small part of A_{11} is in the red square and will be visible in the next magnification. The choice of piece 3 does not change the orientation of the central piece since $s_3(z) = z$ in (3). There is a slight shift in the picture since the focus goes from the control point of the piece to the control point of its subpiece.

In the next step, piece 2 is selected, which rotates the central piece. For the last magnification, we have again chosen piece 3. Thus, the central pieces in the last row are A_{1332} and A_{13323} . In the last picture, 45 neighbors intersect the screen, and it is not clear anymore whether we have a Cantor set. This view could not be anticipated from the global picture on the upper left. In the present case, however, subsequent magnification again simplifies the structure and indicates that it is indeed a Cantor set.

3. Continuous and Discrete Magnification Flow

Furstenberg's concept of a photo. Now, let us treat the magnification process more formally. We first slightly modify Furstenberg's notion of a miniset [10]. Suppose we have an object A , a mapping $\varphi : A \rightarrow X$, and a subset $W \subset X$. Then, $\varphi(A) \cap W$ is called a *photo* of A on the screen W .

This is a very nice abstraction of the everyday concept of a photo. Usually, $X = \mathbb{R}^2$ is the plane, although with 3D printing, we could have three-dimensional photos and in theory also n -dimensional ones [10]. In this paper, W is a square. The object A in our examples is part of the plane. It could also be in some higher-dimensional space. The mapping φ represents the properties of the camera. In our case, φ is a similitude. Usually, $A \cap \varphi^{-1}(W)$ is a small part of A , which is shown in standard size as the photo $\varphi(A) \cap W$. So, φ will be an expanding similitude.

The scenery flow. Let $B_\varepsilon(c)$ denote the closed ball with center c and radius ε . For $c \in A$, the photos generated by

$$\varphi_\varepsilon : A \rightarrow B_1(0) \quad \text{with} \quad \varphi_\varepsilon(z) = (z - c)/\varepsilon \quad \text{for } \varepsilon > 0 \quad (4)$$

have been considered by many authors [10–19]. The photo is $\varphi_\varepsilon(A \cap B_\varepsilon(c))$, a subset of the unit ball $B_1(0)$, which serves as a circular window. Some authors, in particular Furstenberg, take square windows. For a fixed c and variable ε , which tends to 0, these photos form a film which is called the magnification flow or scenery flow of A at c .

It is appropriate to consider ε on a logarithmic scale, taking $\delta = -\log \varepsilon$ as the parameter. The values $\varepsilon = r^k, k = 1, 2, \dots$ then go to equidistant values $\delta = k \log \frac{1}{r}$. For self-similar sets, it was observed that on this logarithmic scale, the collection of photos of the scenery flow is independent of c in a measure-theoretic sense [11,15,19]. If we change the center point c , the magnification film will show the same pictures but in different order. This ‘uniform scaling property’ requires a limit $\varepsilon \rightarrow 0$ from Furstenberg minisets to Furstenberg microsets. It was recently proved for much more general measures and used to solve various problems in geometric measure theory [12,14,17,18]. Our interpretation from a somewhat applied viewpoint is that self-similar sets in the 1990s were seen as models of certain sets in nature, like a fern or a maple leaf [3], and now appear as models of scale-free texture given by a collection of photos which share certain characteristic features.

The discrete magnification flow. It is good to know that our magnification experiments are connected with contemporary fractal research. We need not go into detail, however, since computers do not allow indefinite magnification with $\varepsilon \rightarrow 0$ and $\delta \rightarrow \infty$. We consider a discrete version of the magnification flow. Instead of assuming a point c as the center for the magnification flow, we take a sequence $j_1, j_2, j_3, \dots$ of indices in $\{1, \dots, m\}$ and the corresponding nested sequence

$$A_{j_1} \supset A_{j_1 j_2} \supset A_{j_1 j_2 j_3} \supset \dots$$

of pieces of A of level 1, 2, 3, $\dots$, which in the limit would determine a unique point c of intersection. The center of magnification is slightly shifted at every step. When $A_{j_1 j_2 \dots j_k} = f_{j_1} f_{j_2} \dots f_{j_k}(A)$ with control point $c_k = f_{j_1} f_{j_2} \dots f_{j_k}(0)$ is taken as reference piece in our program, the corresponding photo is given by

$$\varphi^k : A \rightarrow W \quad \text{with} \quad \varphi^k(z) = (z - c_k)/r^k \quad \text{for } k = 1, 2, \dots \quad (5)$$

If we compare this with (4) for $\varepsilon = r^k$ where $\varphi_{r^k}(z) = (z - c)/r^k$, only the reference point is different. Without loss of generality, we can assume that $A \subset B_1(0)$. Moreover, 0 is the control point and $c \in A$. Then, the distance between 0 and c is at most one, and the distance between $c_k = f_{j_1} f_{j_2} \dots f_{j_k}(0)$ and $c \in A_{j_1 j_2 \dots j_k}$ is at most r^k . Consequently, the difference

$$\varphi^k(z) - \varphi_{r^k}(z) = (c - c_k)/r^k \quad \text{is the same vector for all } z \in A, \quad \text{with length } \leq 1.$$

Thus, corresponding photos of the continuous and the discrete scenery flow at c differ only by a translation of length at most one. In our implementation of the discrete flow illustrated in Figure 1, the translation vector is between two points in the small black square. By calculating our picture in the larger black square, we included the translated photo of the continuous scenery flow for the small square. In Figure 1, the ratio of the side lengths of the large and small squares is 3. It can be increased so that the translation vector appears even smaller. This requires additional computer time and memory.

To summarize, the photos of our discrete setting include the photos of the established scenery flow for $\varepsilon = r^k, k = 1, 2, \dots$. It is possible to interpolate our photos numerically

on the continuous scale of all ε . Thus, our method is indeed a visualization of the scenery flow—not for all k up to infinity, only up to some k_0 depending on the IFS and the capacity of the computer. The use of magnification factors $1/r^k$ is motivated by our algorithm, which will now be presented. There is a mathematical benefit, too: for a finite type IFS with the open set condition, our discrete magnification flow is given by a finite number of photos repeated in random order [20].

4. The Principle of Virtual Magnification

Traditional algorithms of magnification calculate the fractal at smaller places and then magnify to the size of the screen. The key idea of the present paper is to represent the data at the small place by a much simpler data structure: the control points of the reference piece and its neighbors in the small place. The configuration of these control points is magnified to standard size. Then, around each standardized control point, we draw the corresponding piece. For this, we need only one numerical copy of A , which is given as a cloud of N points.

Let us demonstrate this idea with Figure 1. The global point cloud on the upper left was calculated by a classical IFS algorithm [3]. It is then transformed to fit into the small black square W at the upper right with a control point at the center. This is our standard point cloud. No further IFS iteration is calculated. All pieces in subsequent magnifications are obtained as rotated and translated copies of the standard cloud.

The self-similarity of A allows the algorithm to determine the positions of the control points in a recursive way. We can forget the coordinates of the small place where we are. We only have to keep track of the relative positions of neighbor control points with respect to the control point of the reference piece. This requires much less work than the local calculation of small pieces of A .

When we have integer IFS data, the calculations of control points and corresponding directions involve fairly small integers. There will be no numerical errors. We may have numerical errors in the initial calculation of the standard point cloud, but these errors cannot accumulate. That is why we can magnify as long as the computer can handle the number of neighbors.

To formulate the principle of virtual magnification in technical terms, we need the concept of a neighbor map which has been used in various forms [6,7,21–25]. Suppose $u = u_1 \dots u_k$ and $w = w_1 \dots w_k$ are two words of length k from the alphabet $\{1, \dots, m\}$. Given the IFS $F = \{f_1, \dots, f_m\}$, there are similitudes $f_u = f_{u_1} f_{u_2} \dots f_{u_k}$ and $f_v = f_{v_1} f_{v_2} \dots f_{v_k}$ which generate the k -th level pieces $A_u = f_u(A)$ and $A_v = f_v(A)$ of the attractor A . The neighbor map from A_u to A_v then is

$$h(u, v) = f_u^{-1} f_v.$$

Note that $h(u, v)$ does not map A_u to A_v . This is accomplished by $k(u, v) = f_v f_u^{-1}$. The neighbor map $h = h(u, v)$ maps the full-size attractor A to a neighboring set $h(A)$ of the same size such that the following diagram commutes.

$$\begin{array}{ccc} A & \xrightarrow{h} & h(A) \\ f_u \downarrow & & \downarrow f_u \\ A_u & \xrightarrow{k} & A_v \end{array}$$

Thus, $h(A)$ has the same metric position with respect to A as A_v has with respect to A_u , up to similarity. The conjugating map is f_u . The following basic equation allows the calculation

of neighbor maps of level $k + 1$ from those of level k . For $i, j \in \{1, \dots, m\}$, the words $ui = u_1 \dots u_k i$ and v_j have a length of $k + 1$.

$$h(ui, v_j) = f_{ui}^{-1} f_{v_j} = f_i^{-1} f_u^{-1} f_v f_j = f_i^{-1} h(u, v) f_j. \quad (6)$$

Starting with the neighbor maps $f_i^{-1} f_j$ of the first level, we can now inductively determine all neighbor maps. This has been accomplished in other papers [6,21,26]. However, there are three differences.

- (i) In other papers, only proper neighbor maps have been considered—that is, neighbors $h(A)$ which intersect A . The definition above includes all potential neighbors. Below, we choose as actual neighbor maps those $h(x) = sx + t$ for which $|t| < C$ where C is a constant depending on A and our window W .
- (ii) In other papers, it was assumed that the IFS fulfills the open set condition or weak separation condition in order to keep the number of proper neighbors finite. For visualization, this assumption is not needed. The computer can perform only finitely many steps, anyway.
- (iii) The standardization forced by f_u^{-1} concerns size, position and direction. In our magnification flow, we change size and position but keep the direction of pieces. In the program, we work with neighbor maps and restore the direction afterwards.

We assumed that the f_j values are given by integer data, $f_j(x) = g^{-1}(s_j(x) + v_j)$ where the matrices g, s_j and the vectors v_j have integer entries, and the s_j belong to a finite group S of isometries with $gS = Sg$. Then, a straightforward calculation of $f_i^{-1} f_j$ and of $h(ui, v_j)$ from $h(u, v)$ by (6) shows that

Proposition 1. *For integer data, all neighbor maps have the form $h(x) = sx + t$ with some integer matrix $s \in S$ and some integer vector t , which is the control point of the neighbor $h(A)$.*

Since s describes the direction change from A to $h(A)$, a neighbor map describes what in geometry is called a flag: combined information on position and direction. The principle of virtual magnification says that instead of magnifying tens of thousands of points, we perform the calculation with tens of flags. Then, we transform one copy of our standard point cloud to each position and direction.

5. The Algorithm and Its Implementation

The algorithm uses only elementary matrix operations. Here are the main steps.

1. **Initialization.** We calculate a global point cloud x representing A by a classical IFS algorithm. We draw x on the whole screen and in the small square. Initial values of parameters include the iteration number $it = 1$ and the total number nit of iterations. The word u and the list of neighbor maps nb are empty. The reference piece is A .
2. **Choice of a subpiece.** There are different options. The simplest is to take a random i uniformly from $\{1, \dots, m\}$. We replace the reference piece A_u by its subpiece A_{ui} by putting $u = ui$, and we save the direction matrix D of the new reference piece.
3. **Calculation of neighbors.** For all neighbor maps h in the list nb , and $j = 1, \dots, m$, we calculate the new neighbor map $h' = f_i^{-1} h f_j$ as in (6) and add the neighbor maps $f_i^{-1} f_j$ with $j \neq i$. Then, we remove the neighbor maps for which $h'(A)$ cannot intersect the large square. This is our new list nb .
4. **Plotting the magnification.** The cloud of the reference piece with its new direction is Dx . The clouds of the reference pieces are $Dh(x)$ with h in nb . If $it = nit$, the job is completed. Otherwise, we increase it by one and go to 2.

We now discuss the implementation of the steps in the appended MATLAB program `scenery.m` and verify the correctness of the algorithm.

The program is run by loading an IFS data file into the workspace and typing `scenery(g,H,nit,np,3)`. Here, *nit* is the total number of magnifications and *np* is the length of pauses in seconds to study subsequent pictures. The last parameter determines the choice of subpieces in step 2. Various data files are provided. Each data file consists of an 2×2 integer matrix g and an $2 \times 3 \times m$ integer array H which contains the digits $h_1, \dots, h_m$, as described in Section 1. The mapping $h_j(x) = s_j x + v_j$ with the 2×2 matrix s_j and 2×1 vector v_j is represented by $H(:, :, j) = [s_j \ v_j]$. It is very simple to edit these data and produce your own data files.

Step 1 starts with the calculation of the cloud x by the procedure `g1oba.m`. A minimum size of $N = 30,000$ points is prescribed. This parameter can be adapted to the size of the screen and the capacity of the computer. Attractors A with larger Hausdorff dimensions require greater N values for graphical accuracy. We determine the minimum level L with $mL > N$ and take the actual number as $N = mL$. The point cloud x is the set of all $f_v(0)$ for words $v = v_1 \dots v_L$ from our alphabet $\{1, \dots, m\}$ —that is, all control points of pieces of level L . It is calculated by a double loop. See Barnsley [3] for related algorithms. As a matrix, x has dimension $2 \times N$.

The system automatically determines maximal and minimal coordinates of x and plots the global point cloud in a corresponding rectangle. We prefer a quadratic window with the control point 0 of A in the center. The maximum length a of a vector in x is called the radius of A . In other words, a is the root of $\max_{n=1}^N x(1,n)^2 + x(2,n)^2$. The small square in the middle of our screen will be $[-a, a]^2$. It will include the point cloud x describing A as well as any rotation of x around 0.

The big square representing the screen is $W = [-b, b]^2$ where the ratio $rat = b/a$ is a parameter which can be changed. We put $rat = 3$. Moreover, we draw the red square which goes to the big square in the next step. Its radius is b/R where $R = \sqrt{\det g}$ denotes the expanding factor of our IFS. The center of the red square is the control point of the subpiece which will be chosen in step 3. To ensure the correctness of step 3, the red square must be contained in W . Since the center of the red square is in the small black square, this condition is fulfilled if $b = rat \cdot a \geq a + b/R$ or $rat \geq R/(R - 1)$. So, rat should be taken larger than $R/(R - 1)$ but not too large because of the computational effort.

For step 2, the simplest choice of a random $i \in \{1, \dots, m\}$ is uniform. This is realized by the command `scenery(g,H,nit,np,0)`. Successive magnifications with uniform choice can be boringly repetitive, however. In particular, when the attractor contains islands as in Figure 1, smaller islands will be regularly visited. For that reason, we wrote the empirical procedure `greedychoice` which leads somewhat faster to interesting places with many neighbors. When the parameter w in the command `scenery(g,H,nit,np,w)` is larger than 0, subpieces with a greater number of neighbors will be favored. The weights will be the numbers of neighbors to the power of w . (The window for those neighbors is chosen smaller than the large square, however.) In our experiments, $w = 3$ performed well, as shown in Figure 2. A deterministic choice of subpiece i is also possible, taking the vector $w = i_1 i_2 \dots i_n$ as an input parameter of `scenery(g,H,nit,np,w)`. The program then produces n iterations, and in the case $nit > n$ continues with random choices according to $w = 3$. The vector of the actually chosen $i_1, i_2, \dots, i_{nit}$ is an output of the program `scenery`. Thus, an interesting magnification series can be repeated, extended and modified.

The direction matrix D of the reference piece is calculated recursively, starting with an initial unity matrix for A . Suppose the actual piece $A_u = f_u(A)$ has direction D and the subpiece $A_{ui} = f_u(f_i(A))$ was selected. Since $f_i = g^{-1}h_i$, the new direction is $Dg^{-1}s_i/R$.

Division by R is necessary to keep an isometry matrix. (For very large numbers of iterations, this calculation can be modified to minimize numerical errors.)

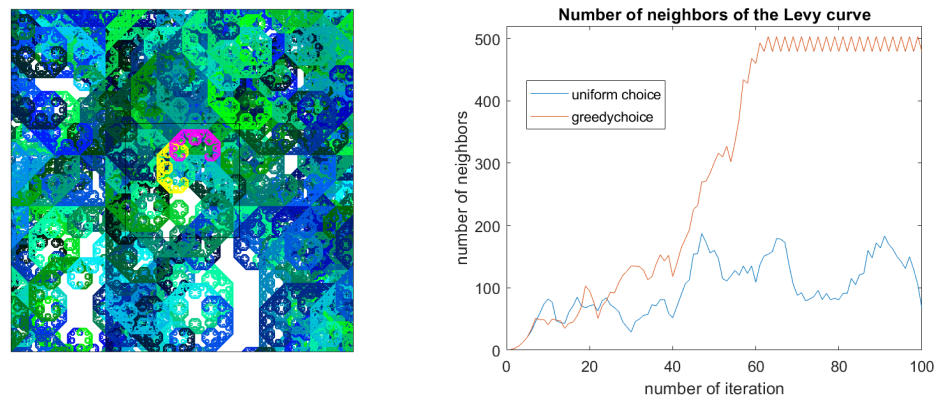


Figure 2. The Levy curve has a non-empty interior. Thus, after finitely many random magnifications, the screen should be filled. **(Left)** 50 magnifications and 270 neighbors were not enough to cover the neighborhood of the reference piece. The coloring is the same as in Figure 1. **(Right)** Even with the greedychoice procedure which favors dense neighborhoods, it regularly took more than 50 steps to reach the stage where about 500 neighbors cover the large square. With a uniform choice of subpieces, however, many more steps are needed.

Step 3 is performed by the procedure `nbmaps.m`. We select the neighbor maps h for A_u in such a way that they include all the sets which hit the screen W . It is here that we move from potential neighbor sets to those whose control point is in the vicinity.

Proposition 2. Suppose $A \subseteq B_\kappa(0)$ and $W \subseteq B_\lambda(0)$ and $h(x) = sx + t$ is a neighbor map with isometry matrix s and translation vector t , and $|t| > \kappa + \lambda$. Then, $h(A) \cap W = \emptyset$.

This is easy to see. Since $h(A) \subseteq B_\kappa(t)$, the existence of a point $y \in h(A) \cap W$ would contradict the triangle inequality for $0, y$, and t . To apply this to our squares, let $\kappa = \sqrt{2}a$ and $\lambda = \sqrt{2}b$. Then, the assumptions of the proposition are fulfilled for our attractor A and our screen W . Let t be the translation vector of a neighbor map h .

$$\text{If } |t| > \sqrt{2}(a + b) \text{ then } h(A) \text{ does not intersect the screen } W. \quad (7)$$

So, all neighbor maps with this condition are cancelled from our list in each step. In two dimensions, this works well, although some remaining neighbor maps miss the screen, too.

When A_u is drawn as a central piece, the list of neighbor maps $h = f_u^{-1}f_v$ comprises all neighbor sets A_v which intersect the screen W . Now, we select a subpiece A_{ui} and draw the red square which goes to W in the next step. Since the red square is contained in W by the choice of rat , the neighbor sets of A_{ui} must be of the form A_{vj} where A_v is a neighbor of A_u . So, we obtain the list of neighbor maps for A_{ui} by applying Equation (6) to all maps h in the list for A_u and $j = 1, \dots, m$. Again, we drop the neighbor maps which fulfill (7).

The number of remaining neighbors is saved in every step and is an output of the program. So, the full command is `z=scenery(g,H,nit,np,w)` where z is a $2 \times nit$ matrix. The upper line contains the indices $i_1, i_2, \dots$, and the lower line contains the number of neighbors at each step. They give insight into the change of density of pieces during magnification and on the computational effort, as shown in Figure 2.

Step 4 of our algorithm needs no explanation. It involves matrix multiplications and graphics operations which in the presence of many neighbors require a lot of time. It is possible to skip this step and perform only the neighbor calculations. One may like to see only a few magnifications on screen and evaluate the other pictures statistically. In our

version, step 4 is regularly performed. This allows us to correct the neighbor list and take only those neighbors for which the cloud includes points on our screen.

6. Remarks on the Program and the Data Files

A starting point for explorative work. MATLAB was designed 40 years ago as a toolbox for experiments with matrices with a very simple and intuitive language. This is exactly what we need here. The language has become a standard. Now, the system is slow, but it is still available at most universities, and there is a free version, Octave. Our procedures are small. You can easily change the data, the parameters or the whole procedure. If you prefer to program in another environment, like Julia, R, Python, C++, Maple etc., you can transfer our algorithm.

Instead of making the program fast and the graphics beautiful, we focused on simplicity, transparency, and accessibility. Our program contains no black box.

Six families of data. We recommend to proceed with the data in lexicographic order. The letter A marks simple examples for which the magnification process can be easily followed. They can be run with $nit = 6$ and $np = 5$ in order to gather some experience with the method. Later, you might prefer $np = 0$.

The relatives of the Sierpiński gasket, mostly taken from [6] and including Figure 1, are distinguished by the letter B. They all have three maps, expanding factor $R = 2$ and Hausdorff dimension $\frac{\log m}{\log R} = \frac{\log 3}{\log 2} \approx 1.58$. The IFS values from [6] were taken without change, and some of them have the control point 0 outside A. This does not matter, however, as long as 0 and A and thus all $f_j(0)$ belong to the small black square. These examples perform well with large nit values and can be easily modified.

The data files marked with the letter C use $g = 2 \pm i$ as an expanding map so that pieces are rotated by 26.6° in each magnification step. These IFSs contain four maps. Their Hausdorff dimension is $\frac{\log 4}{\log \sqrt{5}} \approx 1.72$. The symmetries are again 90° rotations. All IFSs designated by A, B, C are small integer data with the open set condition (OSC), permitting unlimited magnification. Since integer variables are not declared in our implementation, however, they are taken as floating-point numbers, which could result in significant error propagation for very large nit values. We checked that $nit = 1000$ causes no problems. When there is a number at the end of the file name, it denotes the number of proper neighbors and indicates in some way the complexity of the IFS.

The other examples show that the program works well even for non-integer data and IFSs without the open set condition. There are some restrictions, so $nit < 30$ is recommended for the beginning. The family marked by D is related to the hexagonal lattice and rotations by multiples of 60° . D1 up to D3 have expanding map $g = (5 \pm i\sqrt{3})/2$ with $R = \sqrt{7}$, and $m = 6$ mappings, which yields the Hausdorff dimension $\frac{\log m}{\log R} \approx 1.95$. They fulfill the OSC and provide very interesting, natural-looking patterns. D4 is a non-periodic tile with three pieces and $g = (3 + i\sqrt{3})/2$. In example D5, we have the same m and g but obviously not a tile. Here, two pieces of the second level coincide. To treat such ‘exact overlaps’ efficiently, the program has to be modified so that it counts multiple occurrences of the same neighbor piece only once. All these examples on the hexagonal lattice can be mapped into integer data by a linear coordinate transformation.

The three examples designated by the letter E come from a Pisot number of degree four and integer data projected from four-dimensional space, as explained in ([26] Section 4). They all have exact overlaps. The final group of examples with F are definitely not integer data. They have no exact overlaps and do probably not fulfill the OSC. For ‘fatfireworks’, we replaced the factor $g = 2$ of the ‘fireworks’ example in family B by $g = \sqrt{3}$. If this would fulfill the OSC, it must be a tile. Our program suggests that this is not the case. In F2 and F3, three incommensurable rotations were used together with both values of g . In the ‘fat’ case,

we have no tile and thus no OSC. The ‘slim’ case is hard to decide. In the final example, F4, we have replaced $g = 2$ by the affine map $g(x) = \begin{pmatrix} 2.3 & 0 \\ 0 & 2 \end{pmatrix} x$. The attractor is not self-similar, but the program calculates $R = \sqrt{2.3 \cdot 2}$. If there were no rotations, magnifications would show line fibers, since the contractions in horizontal and vertical directions differ. With 90° rotations, the scenery flow shows patterns such as in the self-similar examples for many iterations. This observation was unexpected.

Simple and more complex cases. For integer data IFSs in two dimensions, the finite type condition holds ([26] Section 1). This means that there are only finitely many proper neighbor maps. We can use Proposition 1 and (7) to estimate directly the number of possible neighbor maps $h(x) = sx + t$ which intersect our window W . According to (7), the integer vector t is in the circle with radius $\sqrt{2}(a + b)$ around 0, where a is the radius of A and $b = rat \cdot a$. For our value $rat = 3$, there are

$$\text{at most } 8(a + b)^2 = 128a^2 \text{ translation vectors } t \text{ and } |S| \cdot 128a^2 \text{ neighbor maps.} \quad (8)$$

Here, $|S|$ denotes the number of possible symmetries s , which is 4 for the data files with A, B, and C. So, we have 500 possible neighbor maps when the radius of A is 1 and 500 millions when the radius of A is 1000 (or when the radius is one, and the translations are not integers but decimals with three digits). The size of A matters a lot.

Although the upper bound in (8) need not coincide with the actual number of neighbor maps, it is a good rule that IFSs with small integers in the v_j lead to a small radius of A and are tractable. Our data files A, B, and C belong to this class. It is possible to determine all neighbor maps and to organize them in the neighbor graph. From this automaton, topological properties of A can be derived, as we have shown in [6,26]. Sometimes, even a neighborhood graph can be established which collects the connections between all possible photos [20]. However, from n given neighbors, we can theoretically compose 2^n neighborhoods. So, the neighborhood automaton can be huge compared to our computer’s memory. If the radius of A is large, even the neighbor automaton may be too large. To evaluate such cases correctly, we probably have to apply statistical methods.

The open set condition (OSC) is fulfilled if and only if all neighbor maps are different from the identity [2,4,26]. Thus, it could be desirable to display a warning or stop the program if $h(x) = x$ for some neighbor map. This can easily be accomplished. However, there are examples without an OSC where it can take hours to find such h values. On the other hand, there are examples with an OSC where a proof of the OSC may take hours. Our experience indicates that the question ‘OSC or not OSC?’ should be replaced by ‘low or high complexity?’ with a gradual change from one to the other. There are various ways to measure complexity. For our setting, a measurable parameter is the maximum number of neighbors on a ‘photo’.

7. How to Continue

Improvements of the program and the database. The graphics can be improved when we represent A as an image of $n_1 \times n_2$ pixels and add image intensities on a matrix. This graphical representation is necessary to study IFSs of the form (1) where the f_j values have different contraction factors or self-similar measures instead of self-similar sets. It is also desirable to have more interaction between users and the program. For larger examples, MATLAB has to be replaced by some faster system, like Julia.

Before you start programming, however, we urge you to test our simple package with your own IFS examples. Our data files are a tiny selection from a wide field. The command $H(:, :, j) = [a \ b \ c; d \ e \ f]$; will change the mapping h_j in (2) to $h_j(x) = \begin{pmatrix} a & b \\ d & e \end{pmatrix} x + \begin{pmatrix} c \\ f \end{pmatrix}$ or define a new mapping for $j = m + 1$. Experiments will slow down mainly because of an abundance of neighbors. There is an emergency stop if the number of neighbors exceeds

1000. This command at the end of the program can be removed or changed. In the case of many neighbors, one pragmatic way to modify the IFS is by $g=1.2*g$, which increases the expansion factor and makes the attractor thinner. Feel free to find better solutions. It is not forbidden to perform numerical experiments with IFSs given by decimal numbers without regard to integers or separation conditions, or similarity maps with different contraction factors, or affine contractions.

A change in the program is needed for IFSs with exact overlaps. Either identical neighbor maps must be eliminated, or, in the case of work with image intensities, must be counted with multiplicity. If this is settled, any complex Pisot number g is a possible expansion for an IFS, and this can be considered as integer data, since g is represented by an integer matrix G in a d -dimensional space. The neighbor calculations are performed in the d -dimensional integer lattice and then projected to the two-dimensional eigenspace of G , which corresponds to the eigenvalue g . The point cloud is drawn in this plane. This method, implemented in Mekhontsev's IFStile package [9], is discussed in ([26] Section 3).

New classes of spaces. While two-dimensional self-similar sets have been investigated since 1990 by many authors, there are almost no studies of the three-dimensional case. On one hand, it is difficult to draw 3D fractals (see, however, ref. [9] and Mandelbrot-type fractal demonstrations on diverse web sites). On the other hand, there are very few integer-valued matrices of similitudes or isometries on $\mathbb{R}^3$. Our program can easily be modified for the three-dimensional case. With $g(x) = 2x$ and the symmetry group of the cube, we have a very rich family of examples. It is much more general than the Sierpiński relatives, since we can go from $m = 3$ with dimension 1.58 up to $m = 7$ with dimension 2.8. We expect new geometrical insight from such examples, even for Cantor sets. The problem is to visualize and analyze the 3D constructions. When we use our point cloud method, we need less than twice the memory for the plane for the same number of neighbors, and we will project the three dimensional point clouds to the observer's plane. This is a first step. A good 3D representation must include light sources and ray tracing, and working with intensities instead of point clouds may require hardware beyond a PC.

A similar remark concerns the study of graph-directed IFSs. Concrete examples have been rarely studied, and their potential, in particular in 3D constructions, has not been exploited. They are implemented in [9]. We expect that in 3D, self-similar graph-directed constructions go much beyond ordinary IFSs. This question concerns mathematical exploration rather than computer programming.

Theoretical problems. So far, we have no rigorous method to decide whether a self-similar construction represents a Cantor set, although there are theoretical results [27]. This is an annoying situation especially in 3D where our eyesight can be helpless. An algorithm for disconnectedness probably has to use the neighbor graph and thus would be applicable only to relatively simple IFSs. However, one can also think about computational methods in case the set is given as a finite set of 3D pixels. If an attractor is neither connected nor a Cantor set, one would like to know the structure of its non-trivial connectivity components. In the case of self-similar sets, we conjecture that they are given by a graph-directed IFS.

In the family B of data files, there are four Cantor sets and four non-Cantor sets, which can be distinguished by magnification. We think the visualization can help understand Cantor sets and the structure of components.

All Cantor sets are homeomorphic by a classical theorem, but their metrical structure differs. By eyesight, we can distinguish the photos of different IFSs. So, it is a challenge to find rigorous methods to describe and classify Cantor sets with a fairly homogeneous structure from a metric viewpoint.

Other questions are related to our algorithm. As Figure 2 demonstrates, our empirical greedychoice procedure usually leads faster to dense patterns than the uniform choice of

subpieces. Is there a rigorous method which in self-similar constructions finds the densest patterns? This is connected with the dynamical interior of A , which is defined in [20] with the help of the neighborhood graph. In the case of tilings, it coincides with the topological interior. Lévy did search a long time for an interior point in his curve, which he found in 1937. As Figure 2 indicates, we needed more than 50 steps of random magnification to have the window completely filled with neighbor curves. What is the fastest algorithm to find an interior point in a complicated self-similar tile? Which methods can extract geometric information from the ‘photos’? While colleagues from science have their specific methods to describe soil, smoke and snow, we should have some methodology to understand simple mathematical model sets.

Supplementary Materials: The following supporting information can be downloaded at <https://www.mdpi.com/article/10.3390/fractalfract9120762/s1>, File S1 [fractalscenery.zip]: four MATLAB programs, 31 data files, readme and license file.

Funding: This research received no external funding.

Data Availability Statement: The program and data files described in the paper are available as Supplementary Materials alongside the published article.

Acknowledgments: The author gratefully acknowledges the fast and constructive work of the referees. He thanks Manuel Moran and Michael Barnsley for testing the first version of the program.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Moran, P.A.P. Additive functions of intervals and Hausdorff measure. *Math. Proc. Camb. Philos. Soc.* **1946**, *42*, 15–23. [\[CrossRef\]](#)
2. Bárány, B.; Simon, K.; Solomyak, B. *Self-Similar and Self-Affine Sets and Measures*; Mathematical Surveys and Monographs; American Mathematical Society: Providence, RI, USA, 2023; Volume 276.
3. Barnsley, M.F. *Fractals Everywhere*, 2nd ed.; Academic Press: Cambridge, MA, USA, 1993.
4. Bishop, C.J.; Peres, Y. *Fractal Sets in Probability and Analysis*; Cambridge University Press: Cambridge, UK, 2017.
5. Falconer, K.J. *Fractal Geometry: Mathematical Foundations and Applications*, 3rd ed.; J. Wiley & Sons: Hoboken, NJ, USA, 2014.
6. Bandt, C.; Mekhontsev, D. Elementary fractal geometry. New relatives of the Sierpiński gasket. *Chaos Interdiscip. J. Nonlinear Sci.* **2018**, *28*, 063104. [\[CrossRef\]](#) [\[PubMed\]](#)
7. Thurston, W.P. *Groups, Tilings, and Finite State Automata*; AMS Colloquium Lectures: Boulder, CO, USA, 1989.
8. Fraser, J.M. *Assouad Dimension and Fractal Geometry*; Cambridge University Press: Cambridge, UK, 2020; arXiv:2005.03763.
9. Mekhontsev, D. IFS Tile Finder, Version 2.60. 2021. Version 3.21 Now. Available online: <https://ifstile.com> (accessed on 28 October 2025).
10. Furstenberg, H. Ergodic fractal measures and dimension conservation. *Ergod. Theory Dyn. Syst.* **2008**, *28*, 205–222. [\[CrossRef\]](#)
11. Bandt, C. Local geometry of fractals given by tangent measure distributions. *Monatshefte Math.* **2001**, *133*, 265–280. [\[CrossRef\]](#)
12. Bárány, B.; Käenmäki, A.; Pyörälä, A.; Wu, M. Scaling limits of self-conformal measures. *arXiv* **2023**, arXiv:2308.11399. [\[CrossRef\]](#)
13. Bedford, T.; Fisher, A.M. Ratio geometry, rigidity and the scenery process for hyperbolic Cantor sets. *Ergod. Theory Dyn. Syst.* **1997**, *17*, 531–564. [\[CrossRef\]](#)
14. Fraser, J.M.; Pollicott, M. Uniform scaling limits for ergodic measures. *arXiv* **2015**, arXiv:1502.05610. [\[CrossRef\]](#)
15. Gavish, M. Measures with uniform scaling scenery. *Ergod. Theory Dyn. Syst.* **2011**, *31*, 33–48. [\[CrossRef\]](#)
16. Hochman, M. Dynamics on fractals and fractal distributions. *arXiv* **2010**, arXiv:1008.3731.
17. Hochman, M.; Shmerkin, P. Equidistribution from fractal measures. *Invent. Math.* **2015**, *202*, 427–479. [\[CrossRef\]](#)
18. Käenmäki, A.; Sahlsten, T.; Shmerkin, P. Dynamics of the scenery flow and geometry of measures. *Proc. Lond. Math. Soc.* **2015**, *110*, 1248–1280. [\[CrossRef\]](#)
19. Mörters, P.; Preiss, D. Tangent measure distributions of fractal measures. *Math. Ann.* **1998**, *312*, 53–93. [\[CrossRef\]](#)
20. Bandt, C. Elementary fractal geometry. 6. The dynamical interior of self-similar sets. *arXiv* **2025**, arXiv:2503.10430. [\[CrossRef\]](#)
21. Bandt, C.; Mesing, M. Self-affine fractals of finite type. In *Convex and Fractal Geometry*; Banach Center Publications; Institute of Mathematics of the Polish Academy of Sciences: Warsaw, Poland, 2009; Volume 84, pp. 131–148.
22. Lau, K.-S.; Ngai, S.-M.; Rao, H. Iterated function systems with overlaps and self-similar measures. *J. Lond. Math. Soc.* **2001**, *63*, 99–116. [\[CrossRef\]](#)
23. Loridant, B. Crystallographic number systems. *Monatshefte Math.* **2012**, *167*, 511–529. [\[CrossRef\]](#)

24. Scheicher, K.; Thuswaldner, J.M. Neighbors of self-affine tiles in lattice tilings. In *Fractals in Graz 2001*; Grabner, P., Woess, W., Eds.; Birkhäuser: Basel, Switzerland, 2003; pp. 241–262.
25. Strichartz, R.S.; Wang, Y. Geometry of self-affine tiles 1. *Indiana Univ. Math. J.* **1999**, *48*, 1–24. [[CrossRef](#)]
26. Bandt, C. Elementary fractal geometry. 3. Complex Pisot factors imply finite type. *Discret. Comput. Geom.* **2025**, *74*, 661–690. [[CrossRef](#)]
27. Luo, J.; Xiong, D.H. A criterion for self-similar sets to be totally disconnected. *Ann. Fenn. Math.* **2021**, *46*, 1155–1159. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.