Structural Beauty of Fractals: A Case Study on the Mandelbrot set and Julia sets

Andy Jingqian Xue

LivableCityLAB, Urban Governance and Design Thrust, Society Hub
The Hong Kong University of Science and Technology (Guangzhou)
Email: andy.j.xue@connect.hkust-gz.edu.cn

(Daft: March 2024)

Abstract
Fractals are widely renowned for their aesthetic appeal, often considered among the most visually captivating patterns. Among these, the Mandelbrot Set and Julia Sets are particularly notable for their marvelous visual effects and self-similarity across various scales. This kind of beauty is not just a matter of personal opinion but an objective fact, and that living structure is the underlying notion that evokes the sense of beauty. This tutorial offers step-by-step instructions for computing and analyzing the structural beauty of MS and JS. Based on the concept of living structure, this structural beauty or livingness (L) is determined by the substructures (S) and their inherent hierarchy. First, we will introduce the definition of the Mandelbrot set and Julia sets, along with the process used to generate them. Following this, we offer an interactive Python program for understanding MS and JS and creating images of them. The results allow for the further measurement of their structural beauty. The source code of this tutorial is available at: https://github.com/AndyXue957/beauty-of-fractals.

Keywords: Living structure, fractal geometry, scaling hierarchy, structural beauty of images, livingness, head/tail breaks

1. Introduction

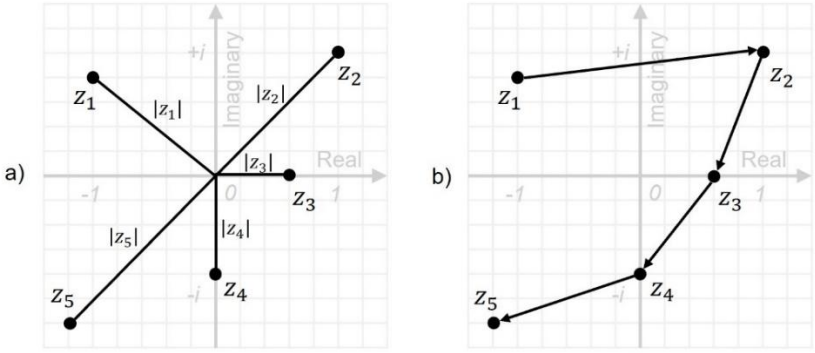
The Mandelbrot set (MS) and Julia sets (JS) are two-dimensional sets that exhibit marvelous visual effects but generated from a simple iterative equation. The recursive nature of both MS and JS identifies them as fractals, showing remarkable complex geometric shape and self-similarity (Mandelbrot 1982). The beauty of them is often said to be in the eye of the beholder, suggesting it varies from one individual to another. However, this commonly held thinking has long been challenged that beauty is partially objective while partially subjective. The discovery of living structure by Christopher Alexander (2002–2005) could be a milestone, demonstrating that the underlying structure–or living structure–is the reason for the sense of wellbeing and beauty evoked in the human heart and mind. Living structure to beauty is what temperature to warmth, and the livingness is defined by a mathematical structure composed of multiple organized and interconnected substructures with an inherent hierarchy of far more small substructures than large ones (Jiang 2019). This hierarchical structure exhibits a scaling relationship that navigates between scales and details, reflecting a long-tailed distribution and the non-linearity of its complex organization.

A fractal is perceived intuitively as a geometry that intricately blends complexity with aesthetic beauty. This complexity can be quantified through fractal dimension *D*, which is a statistical measure that provides insight into how detailed a fractal pattern is at various scales (Mandelbrot 1967). Unlike the dimensions we are familiar with (1D for a line, 2D for a plane, etc.), the fractal dimension of a shape can be non-integer, indicating that the fractal fills space in a way that is not completely one-dimensional or two-dimensional, but somewhere in between. For instance, a coastline might have a fractal dimension that suggests it is more complex than a straight line (1D) but does not fully occupy a two-dimensional area. The fractal dimension, to a certain extent, is able to quantify beauty through complexity. It primarily focuses on the quantitative relationships and density of geometric elements yet overlooks the objective and structural essence of beauty (Jiang and Yin 2014). Factually, a fractal is a living structure, and its structural beauty or livingness (L) is determined by the number of substructures (S) and their inherent scaling hierarchy (H). In other words, the more substructures there are and the higher the levels of hierarchy within it, the more structurally beautiful and complex a fractal becomes.

The remainder of this tutorial is structured as follows. Section 2 introduces the mathematical definition of MS and JS and along with the approach of computing structural beauty of fractals. Section 3 presents the specific guidance of generating MS and JS automatically and the understanding of two different statuses of a sequence. Section 4 presents the process of computing the substructures of MS and calculating the structural beauty of livingness. The source code including the above contents is available at: https://github.com/AndyXue957/beauty-of-fractals. Please follow the instructions provided in the repository for further information about configuration and installation.

2. Mandelbrot set, Julia sets and structural beauty of fractals

The Mandelbrot Set and Julia Sets are defined within the complex plane, a two-dimensional Cartesian coordinate system specifically designed for complex numbers $z = x + yi$, where the horizontal axis represents the real component x, and the vertical axis corresponds to the imaginary component y (Panel a and Panel b of Figure 1). Each complex number refers to a specific point in the complex plane and the absolute value or magnitude of a complex number is equivalent to the distance from the coordinate origin and that corresponding point itself. Taking point z_1(-1,0.8) as an example, it denotes the complex number of -1+0.8i, whose magnitude is $\sqrt{(-1)^2 + (0.8)^2} \approx 1.3$. For a sequence as $\{z_1, z_2, z_3, z_4, z_5\}$, it is represented as an evolving trajectory or an orbit on the complex plane (Panel b of Figure 1). This sequence of complex numbers might exhibit various statuses, which is the fundamental principle behind the generation of MS and JS. Due to computational limitations, not all points on the complex plane can be accurately represented. Consequently, the complex plane is stored in computers as an image, or a raster composed of discrete grids of points. For a complex plane with an extent of x: -1 to 1 and y: -1 to 1, and a raster size of 20×20, the resolution of this image is 0.1. This setup results in points such as (-1.0, 0.8), (0, 0.6), (-1.2, -1.2) … that are the multiples of 0.1.



c)

n	1	2	3	4	5
z_n	-1 + 0.8i	1 + 1i	0.6	-0.8i	-1.2 - 1.2i
$\lvert z_n \rvert$	$\sqrt{(1)^2+(0.8)^2}$	$\sqrt{(1)^2+(1)^2}$	$\sqrt{(0.6)^2}$	$\sqrt{(0.8)^2}$	$\sqrt{(1.2)^2+(1.2)^2}$

Figure 1: Illustration of complex numbers and sequence in the complex plane
(Note: A complex plane is represented by a two-dimensional Cartesian coordinate system of complex numbers. The sequence of complex numbers $\{z_1, z_2, z_3, z_4, z_5\}$ provided in Panel c is plotted individually with corresponding magnitudes in the complex plane in Panel a, whereas it is plotted successively from z_1 to z_5 in Panel b.)

Both MS and JS are defined through the iterative processing of complex numbers in the context of a simple mathematical formula. For MS, each point c in the complex plane is iterated through the equation $z_{n+1} = z_n^2 + c$, where z starts at $z_0 = 0$ and c is the complex number associated with the current point being evaluated. The point c belongs to MS if, after numerous iterations, the value of z does not approach infinity. In other words, the point c does not belong to MS if, the corresponding sequence $\{z_1, z_2, z_3, \ldots, z_N\}$ gets arbitrarily large to infinity. Therefore, the outcomes of iterating a sequence starting from a given value of c are binary: the sequence either diverges to infinity or remains bounded within a limited range (see Section 3 for details). JS, on the other hand, are defined similarly but with a crucial difference: instead of varying c for each point on the complex plane, c is fixed for the entire set, and the initial value

of z_0 is varied across each point in the plane. The resulting set of points, for which $\{z_1, z_2, z_3, \ldots, z_N\}$ does not diverge to infinity under iteration, forms a Julia Set. Each different value of c generates a unique Julia Set, producing an infinite variety of intricate patterns. In conclusion, MS is mapped on the complex plane using various complex numbers c with a fixed initial value of z_0, while each JS is plotted with various initial values z_0 and a fixed value of c. For each distinct c, there corresponds a unique JS, illustrating that there are multiple Julia Sets rather than a single entity.

Living structure is what underlies the notion of structural beauty or livingness (Jiang and de Rijke 2023). The concept of living structure is universal, thereby a fractal is regarded as a living structure. As mentioned above, the structural beauty of fractals (L) is determined by the substructures (S) and their inherent hierarchy (H). Mathematically, it is defined by the multiplication of the number of substructures and their levels of hierarchy:

$$L = S \times H \tag{1}$$

Substructures represent various types of components depending on the context. For instance, in the analysis of structural beauty or complexity of boundaries, a substructure could be defined as a bend determined by three vertices (Ma and Jiang 2018). On the other hand, when focusing on an image, a substructure could be interpreted as a figure or ground according to the Gestalt theory. The level of the inherent hierarchy where there are far more small substructures than large ones is computed by head/tail breaks (Jiang 2013). It quantitatively measures the extent to which the principle of "far more small things than large ones" recurs within the distribution, thereby revealing a scaling hierarchy. By iteratively dividing the data into "head" (comprising larger values) and "tail" (comprising smaller values) based on the mean value at each level, the method uncovers and categorizes the intrinsic hierarchical structure of the data.

3. Generating binary Mandelbrot set and Julia sets

As mentioned in Section 2, MS and JS are defined in a set of complex numbers that the corresponding iterative sequence $\{z_1, z_2, z_3, \ldots, z_N\}$ does not diverge to infinity. The concept opposite to divergence is boundedness. More specifically, a sequence is said to be bounded if there exists a limit to how far its elements can be from the origin in the complex plane. For MS and JS, this limit is conventionally set to 2, which means that if the absolute value of any point z_n in the sequence exceeds 2, the sequence is considered to diverge. Taking c=-i and z_0=0 as an example, the corresponding sequence is {i, -1+i, 0, i, -1+i, …} whether its absolute values are $\{1, \sqrt{2}, 0, 1, \sqrt{2}, 0, \ldots\}$, thereby c=-i belongs to MS. On the other hand, for c=1, the sequence is {0, 1, 2, 5, 26, …}, which is tend to infinity, therefore c=1 does not belong to MS. Following the below steps for better understanding of the two statuses (Figure 2):

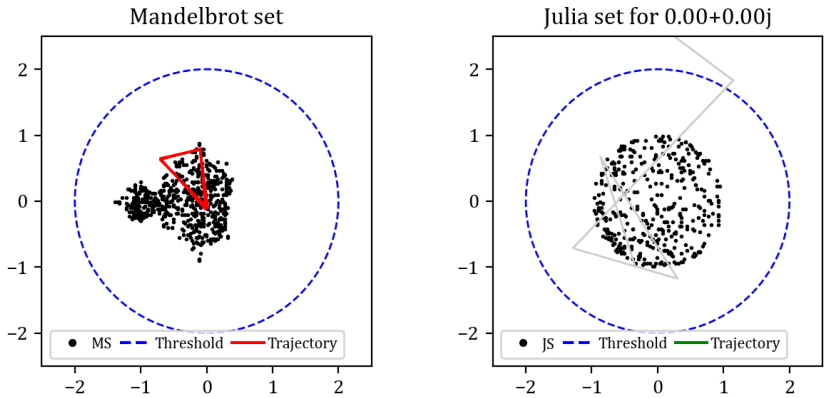


Figure 2: User interface of painting MS and JS with the trajectory for the sequence of current point (Note: The blue circle with a radius of 2 is the threshold boundary determining whether a sequence diverges. This means that if a point in the trajectory exits this boundary, the sequence is considered divergent, and the originating complex number does not belong to the specific set. Bounded trajectories are colored in red or green, while divergent trajectories are depicted in gray. Black points represent the complex numbers that belong to MS or JS.)

1) Run *s3_trajectory.py* and select a value of complex number c on the left complex plane to draw different JS by setting parameter *c_pre* in line 8. The trajectory of sequence for MS will be displayed on the left side while JS on the right side with a threshold boundary to determine the divergence of a sequence or trajectory.

2) Click and drag a point from any location to another space within the complex plane. If the current point belongs to MS—in other words, if the corresponding sequence remains bounded within the threshold circle—it will be marked as a black point on the visualization. By going through all the points in the complex plane, you can gradually reveal the approximate shapes of MS and JS.

MS or JS are computed and visualized as a raster or in an image format. Each point or pixel can exist in only two states: either it belongs to the set, or it does not. Consequently, the images of MS and JS are binary, characterized by values of either 0 (indicating non-membership) or 1 (indicating membership). MS is visualized on the complex plane of c, thereby each number of c corresponds to a distinct JS as mentioned in Section 2. Following the below steps for generating and saving image files of MS and JS:

1) Run *s3_binary.py* and the PNG format image of MS will be automatically saved under the root folder as *ms_16.png*. The complex plane for MS is displayed on the lefthand side (Panel a in Figure 3), whereas JS is shown on the righthand side (Panel b in Figure 3).

2) Click and drag a point (Panel a in Figure 3) from any location to another space within the complex plane of MS. Each point corresponds to a different JS. Click "save" button to save the current image of JS such as *-0.10_0.79.png* under the root folder.

3) Adjust different sizes of the images where you can see MS and JS at different resolutions (Panel c in Figure 3). Change the parameters *width* and *height* in the source code and try to discover the difference by yourself. Since MS is symmetrical about the y-axis and contains graphical elements on the y-axis itself, the grid size must be an odd number to display MS's symmetry fully and accurately. For instance, if the range of the y-axis is -2 to 2 but the grid size is set to 40, the minimum discernible unit would have a spacing of 0.1. This means the actual center point would not fall precisely on 0, thus failing to accurately display its symmetry.

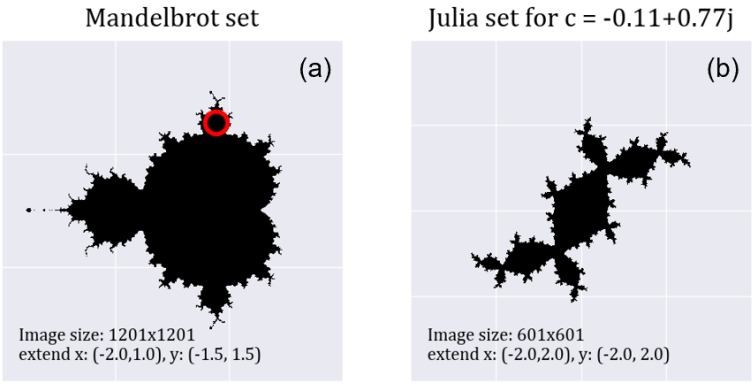


Figure 3: Images of MS and the corresponding JS for a certain point (in red), with key parameter settings for generating these images in the source code
(Note: The red circle in Panel denotes the specific complex number c=-0.11+0.77i for Panel b. The extent for MS is x: -2 to 1 and y: -1.5 to 1.5, while the extent for JS is x: -2 to 2 and y: -2 to 2. The image size of both MS and JS is 601×601)

MS and JS are generated through an iterative process, and in this tutorial, we will focus on MS to elaborate how the iterative generation works. The determination of whether a sequence diverges or remains bounded is not fixed but changes as the iteration progresses of $z_{n+1} = z_n^2 + c$. For the point c=1+i starting from z_0=0, during the first three iterations, the sequence is {-0.5+i, -1.25, 1.0625+i}, with its corresponding absolute values approximately {1.12, 1.25, 1.46}, remaining bounded within the

threshold of 2. However, at the fourth iteration, $z_4 \approx -0.37+3.13i$ with an absolute value of 3.147, exhibiting a tendency to diverge since the iterative equation involves squaring the complex numbers. This indicates that during the iterative process, certain points in the complex plane may transition from being bound to diverging, and the boundary of MS gradually shrinks and narrows down, eventually approaching a steady state. Following the below steps for generating iterating process of MS and save them as image files (Figure 4):

1) Choose five specific iterations for interpreting the iterative process, iterations of 1, 2, 4, 8 and 16 are recommended for better comparison. You can select other settings for further exploration on the evolution of MS.

2) Change the parameter *max_iter* at line 9 in script *s3_binary.py* and run for each setting of iteration. In total you should run 5 times, but it won't take long.

3) Eventually you will generate 5 PNG format images under the root folder, and they are named as *ms_1.png*, *ms_2.png*, … *ms_16.png*, respectively.

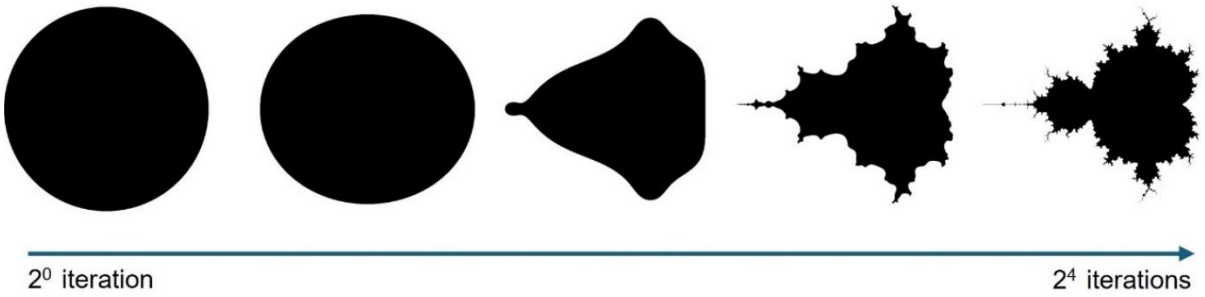


Figure 4: Illustration of iterative process of MS at five different iterations
(Note: These images display the evolution of iterations from left to right, starting with 1, then 2, 4, 8, and finally 16. As the iterations increase, the shape of MS progressively shrinks until it stabilizes.)

Through these experiments, we've discovered that the MS and JS possess astonishing visual effects and are iteratively evolved into complex shapes. They are processed and digitized into an image format, which encompasses properties such as image size. Based on it, in the next section we will elaborate how to quantify or compute the structural beauty of these fractals.

4. Quantifying structural beauty of MS and JS

Structural beauty or livingness (L) is defined by the multiplication of the number of substructures (S) and their hierarchy (H). In Section 4 we focus on the structural beauty of the boundary of MS or JS, which is very much like the coastline (Mandelbrot 1967). The bend each determined by three vertices is considered as a substructure. Therefore, the scaling hierarchy is that there are far more short bends than large ones. In this tutorial we only compute the upper part as they are symmetrical. Following the below steps for computing the structural beauty of MS:

1) Select the PNG format image generated during Section 3. For finer details in MS, we recommend enlarging the image size twenty times and only upper half to 12001×6001 and extent of x: -2 to 2 and y: 0 to 2. In this Section we will take MS at iteration 16 as an example (fifth MS in Figure 4).

2) Up to this point, we have only a binary image of the MS, and the aim of this section is to compute the structural beauty of MS's boundary. To achieve this, we first need to obtain the boundary file of MS. Open the *ms_16.png* in ArcMap or other GIS software and adjust the symbology to unique value where pixel of value 0 in white and pixels of value 255 in black (Panel a in Figure 5).

3) Use *Tools->Surface->Counter* tool to obtain the polyline vector which contains boundary information (Panel b and c in Figure 5). Set the *Contour interval* as 255 and other parameters as

default. A sample contour file is available in the *sample_data* folder for those who do not have ArcGIS installed.

After obtaining the vector contour of MS, we can compute the substructures by generating the bends of the boundary recursively and apply head/tail breaks 2.0 for quantifying their inherent scaling hierarchy. Based on the results, we can calculate the degree of structural beauty or livingness determined by the substructures. Follow the below steps to obtain the results and further analysis:

1) Open *s4_beauty.py* and change the parameters of *input_path* and *output_path* to specify the locations of the input MS contour shapefile and the output polyline shapefile for the identified bends, respectively. For example, set *input_path='contour16.shp'* and *input_path='bends16.shp*.

2) Run *s4_beauty.py* and the Python program will identify the bends and compute the structural beauty automatically.

3) Check whether the polyline shapefile is successfully output as *bends16.shp*. If so, open it in ArcMap (Panel e in Figure 5).

4) The statistics of substructures and their inherent hierarchy computed by head/tail breaks 2.0 (Jiang 2019) will be printed on the command window (Panel e in Figure 5).

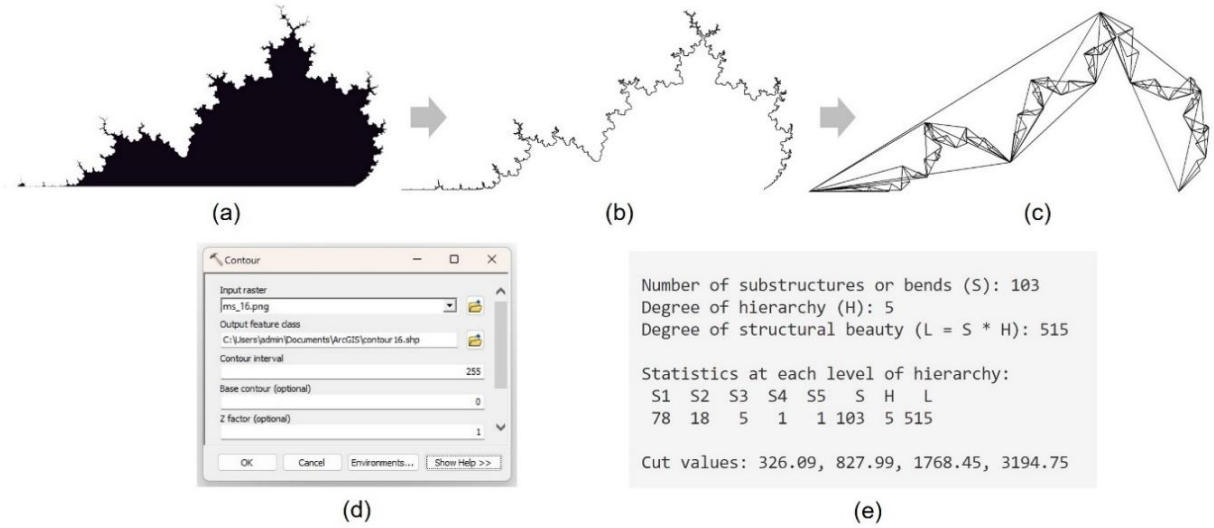


Figure 5: Illustration of obtaining bends and their statistics from MS in raster format
(Note: Image of the upper half of MS in Panel a can be generated following the steps in Section 3. The boundary of MS (Panel b) can be obtained using *Contour* tool (Panel d) in *Spatial Analyst Tools->Surface.* The contour lines can be transformed into bends (Panel c) through *s4_beauty.py* and their statics of structural beauty (Panel e) will be output in the terminal of your Python environment. The cut values represent the division distance value of two categories using head/tail breaks 2.0.)

By following the same guidance, you can try to compute the bends of JS and MS at different iterations. The scaling hierarchy of far more small bends than large ones can be visualized though spectral colormap in ArcMap. In this Section we provide a case study on MS at 4th iteration and 16th iteration. Follow the below steps to compute the statistics of substructures and their levels of hierarchy:

1) Right click the bend feature and select *properties*. Choose *Symbology->Quantities->Graduated color*. Select *value* as *class* and set the classes of number the same as H (Panel e in Figure 5).

2) Use *spectral* colormap which is originally from red to blue for better visualization. Right click the headings of the table for colors and *Reverse Sorting*. The visualization indicates that there are far more small bends than large bends, in other words, there are far more blue bends than red bends.

As the results demonstrate (Figure 6), MS at the 16th iteration (Panel d in Figure 6) exhibits a higher level of hierarchy(H) than at the 4th iteration (Panel a), with the former displaying 5 levels of hierarchy and the latter only 3. Correspondingly, the number of bends or substructures (S) in the 16th iteration is significantly greater, in total to (78+18+5+1+1) = 103 bends or substructures. In contrast to the 4th iteration, it has a total of (7+1+1) = 9 bends (Panel b and d). As mentioned above, the structural beauty (L) of a fractal is defined by the multiplication of the number of substructures and their hierarchy. Therefore, the structural beauty of these MS is (78+18+5+1+1) × 5 = 515 and (7+1+1) × 3 = 27, indicating that MS at late iteration is much structurally beautiful or living than the early one. Substructures generated from both two MS could be well fitted by the power law, displaying a linear shape in log-log plot (Panel c and f), while a heavy-tailed distribution in linear scale plot.

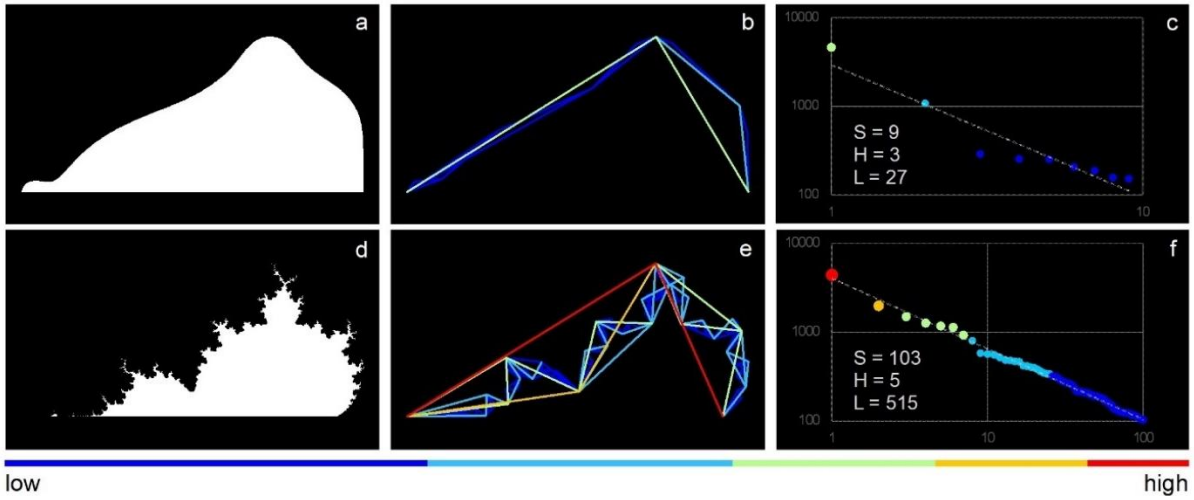


Figure 6: Comparison between MS after 4 iterations and 16 iterations
(Note: Panel a and Panel d show the images of MS at the 4th iteration and 16th iteration, respectively. Generated from the images, Panel b and Panel e show the scaling hierarchy of substructures for the MS boundaries, where there are far more small bends or substructures than large ones. Substructures at different level of hierarchy is colored using *spectral* colormap in ArcMap from lowest (1) to highest level (5). Panel c and Panel f depict the log-log plot of the rank-size relationship, and both panels exhibit a linear shape, indicative of a long-tailed distribution when viewed on a linear scale. S=number of substructures, H=levels of hierarchy, L=degree of structural beauty or livingness of fractals.)

JS are regarded as a subset of MS because each point in the complex plane of MS corresponds to a unique JS, among which many captivating shapes are yet to be uncovered. By computing the structural beauty or livingness of all JS, it is possible to create a map of MS based on the varying degrees of livingness. This process results in a beautifully colored MS, showcasing the diverse and intricate beauty embedded within its structure. Additionally, interpreting the livingness of MS and JS at each iteration allows for an insightful exploration into how a fractal evolves recursively, becoming increasingly complex and visually appealing in terms of its shape.

5. Conclusion

A fractal is a geometric entity characterized by a complex structure and pattern, which generates an appealing visual impact through its inherent complexity. This aesthetic appeal or beauty is not subjective or subject to individual interpretation but originates from its underlying living structure consisting of interconnected smaller substructures. The living structure is quantifiable and defined in a scaling hierarchy, where there are far more substructures than large ones. By examining MS and JS as the most representative examples of fractals, quantitative analysis reveals that the boundaries of these fractals undergo evolution, enhancing their structural beauty through iteration. This recursive process initially generates large-scale substructures, but as iteration progresses, it increasingly focuses on creating finer, small-scale substructures. It implies that the more substructures the more beautiful, and the more levels of hierarchy or scales the more beautiful a fractal is. The discovery of living structure establishes a

connection between complexity and beauty, providing a means to quantitatively describe and visualize complexity with the inherent hierarchy.

Acknowledgement
I would like to thank Prof. Bin Jiang for his patient guidance on this tutorial. The structure-based approach to computing the beauty of fractals is inspired by the work on computing the scaling hierarchy of the coast of Britain (Ma and Jiang, 2018). The implementation of the source code was facilitated by the assistance of ChatGPT-4.

References:

Alexander C. (2002–2005), *The Nature of Order: An essay on the art of building and the nature of the universe*, Center for Environmental Structure: Berkeley, CA.

Mandelbrot B. B. (1982), *The Fractal Geometry of Nature*, W. H. Freeman and Co.: New York.

Mandelbrot B. B. (1967), How long is the coast of Britain? Statistical self-similarity and fractional dimension, *Science*, 156(3775), 636–638.

Jiang B. (2019), Living structure down to earth and up to heaven: Christopher Alexander, *Urban Science*, 3(3), 96.

Jiang B. and Yin J. (2014), Ht-index for quantifying the fractal or scaling structure of geographic features, *Annals of the Association of American Geographers*, 104(3), 530–541.

Jiang B. and de Rijke C. (2023), Living images: A recursive approach to computing the structural beauty of images or the livingness of space, *Annals of the American Association of Geographers*, 113(6), 1329–1347.

Jiang B. (2013), Head/tail breaks: A new classification scheme for data with a heavy-tailed distribution, *The Professional Geographer*, 65(3), 482–494.

Jiang B. (2019), A recursive definition of goodness of space for bridging the concepts of space and place for sustainability, *Sustainability*, 11(15), 4091.

Ma D. and Jiang B. (2018), A smooth curve as a fractal under the third definition, *Cartographica: The International Journal for Geographic Information and Geovisualization*, 53(3), 203–210.

Further readings:

Devaney R. (2006), Unveiling the Mandelbrot set, In *Plus*, https://fractal.org/Unveiling-Mandelbrot-set.pdf.

Julia set (2024), In *Wikipedia*, https://en.wikipedia.org/wiki/Julia_set.

Mandelbrot set (2024), In *Wikipedia*, https://en.wikipedia.org/wiki/Mandelbrot_set.

Numberphile (2014), The Mandelbrot Set – Numberphile, In *YouTube*, https://www.youtube.com/watch?v=NGMRB4O922I.

Numberphile2 (2014), Filled Julia Set, In *YouTube*, https://www.youtube.com/watch?v=oCkQ7WK7vuY.

Ramer-Douglas-Peucker algorithm (2024), In *Wikipedia*, https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm.

The Mathemagicians' Guild (2020), The Mandelbrot Set Explained, In *YouTube*, https://www.youtube.com/watch?v=7MotVcGvFMg.

The Mathemagicians' Guild (2021), Julia Sets, and how they relate to The Mandelbrot Set, In *YouTube*, https://www.youtube.com/watch?v=dctJ7ISkU-4.

Appendix A: Documentation for the source code

The source code for this tutorial is hosted on GitHub at https://github.com/AndyXue957/beauty-of-fractals and is written for Python 3.11. The requirement of Python packages is provided in *requirement.txt*. The repository comprises various components essential for the study of fractals and their structural beauty, including:

- *sample_data*
  - *contour16.tif*: Boundary polyline generated from the Mandelbrot set after 16 iterations.
- *s3_trajectory.py*: A Python script to generate the trajectory of points on complex plane.
- *s3_binary.py*: A Python script to generate binary Mandelbrot set and Julia sets.
- *s4_beauty.py*: A python script to compute structural beauty of the contour lines of boundaries.
- *requirements.txt*: Requirements of python packages in this program.

Here are some key instructions and insights for the program. For *s3_trajectory.py*:

- Julia Set Initialization: In line 8, initialize the Julia set's c values on the right complex plane. A default value of c=0 produces a circular Julia set.

- Trajectory Computation: The function *compute_trajectory* (line 13-25) calculates trajectories differently for points in the Mandelbrot set (initial z=0, c is the click position) compared to those in the Julia set (z is the click position, c is predefined). The iterative formula $z_{n+1} = z_n^2 + c$ is applied to compute each subsequent point z_n.

For *s3_binary.py*:

- Iteration and Image Settings: Set the maximum iterations for generating the Mandelbrot and Julia sets (line 9), adjusting image sizes and display extents (lines 12-16) for detailed visualization.

- Mandelbrot and Julia Set Generation: Functions *generate_mandelbrot* (line 28-50) and *generate_julia* (line 54-73) define the complex plane and compute each point's divergence, marking non-divergent points to highlight the set's boundaries.

For *s4_beauty.py*:

- Input and output files are specified (line 9-10), along with a tolerance parameter defining the minimum bend size to preserve (line 11).

- Bend Identification: Function *recursive_bend_identification* (line 60-81) utilizes a recursive method to preserve significant bends, splitting the polyline at the farthest point from a direct line in its initial iteration.

- Structural Beauty Calculation: Line 118-120 computes the statistical measures of structural beauty, including the number of bends and their hierarchical scaling determined by the head/tail breaks method (function *head_tail_breaks* in line 14-44).

For detailed understanding and execution instructions, the source code is annotated with comments that provide deeper insights into each process. The results, including the structural beauty statistics, are displayed in the command window, offering a comprehensive analysis of fractals' geometric and aesthetic properties.

Appendix B: Identification of substructures of a boundary

A fractal is a living structure, consisting of interconnected substructures with inherent scaling hierarchy where there are far more small substructures than large ones. This structure is the underlying reason that evokes the feeling of wellbeing and beauty from people's heart and mind. Therefore, precise definition of these substructures is crucial for quantifying structural beauty. Technically, a substructure for the boundary of the Mandelbrot set or the Julia sets is defined as a curve or bend represented by three vertices (Panel a in Figure B1), or alternatively, by the two lines connecting these vertices (Panel b in Figure B1. The bends can be identified recursively through iterative end-point fit algorithm (Douglas and Peucker 1973), also well known as Douglas-Peucker algorithm. This algorithm begins by identifying the largest bends and continues until all the smallest bends are detected. The magnitude of a bend is quantitatively determined by the shortest distance from the arc of the bend to the straight line joining the bend's endpoints, which means the longer the perpendicular distance is, the larger a bend is. Larger bends are the first parts of the boundary to be identified, while smaller bends become visible with more detailed observation. Thus, the living structure of the boundaries for the Mandelbrot set and the Julia sets should consist of far more small bends than large bends.

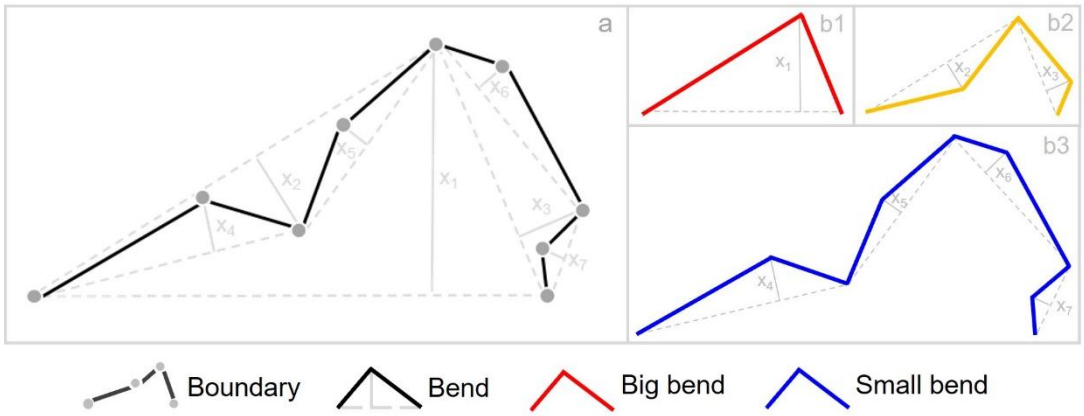


Figure B1: Illustration of the identifying process of substructures or bends of a boundary (Note: A bend is determined by three vertices within the boundary (Panel a) and two connected lines between these vertices (Panel b). Each bend, represented as x_1, x_2, ... x_n, is labeled with perpendicular distances, where larger distances are highlighted in red and smaller ones in blue. The characteristics of far more small bends or substructures than large ones (head percentage threshold as 45%) recurs twice for this boundary where bends: $x_1 > x_2+x_3 > x_4+x_5+x_6+x_7$, resulting in three levels of hierarchy as displayed in Panel b1 to b3.)

For example, considering a boundary that consists of 9 vertices (Panel a in Figure B1), the identification begins its process by identifying two base points that are furthest apart on the boundary to form a baseline (demonstrated as dashed lines). From there, it searches for the point on the boundary that is farthest from this baseline, considering this point as a significant bend as its maximum deviation (solid grey lines). The boundary is then split into two segments at this maximum deviation point, one segment running from one base point to the deviation point and the other from the deviation point to the other base point. The algorithm recursively applies the same process to each segment, searching for points with the maximum deviation from the newly formed baselines and splitting the segments further if necessary. This recursive division continues until the deviation of points from the baseline in all segments is below a predefined threshold, which is the *tolerance* parameter in *s4_beauty*. The first iteration identifies x_1. The second iteration identify x_2 and x_3 from x_1, while the last iteration brings x_4, x_5 from x_2, and x_6, x_7 from x_3. Each bend could be represented by the two connected lines between three vertices (Panel b1 to b3 in Figure B1). The magnitudes of the bends can be classified using the head/tail breaks (Jiang 2013) with the break percentage of 45% and the results show that the heavy-tailed distribution of far more small bends (colored in blue) than large ones (colored in red) recurs twice as $x_1 > x_2+x_3 > x_4+x_5+x_6+x_7$.