# Graph-Theoretic Boundary Analysis of Classical Fractals Using Edge Pixel Adjacency

Douglas C. Youvan

doug@youvan.com

April 5, 2025

Fractals are mathematical objects known for their self-similar structure and intricate boundaries that repeat across scales. They appear both in theoretical contexts—such as the Mandelbrot and Julia sets—and in geometric constructions like the Sierpinski Triangle and Carpet. Traditionally, fractals are visualized using color gradients based on escape-time iterations or recursive geometric patterns. While these renderings reveal aesthetic and recursive complexity, they often obscure the local boundary structure of the fractal itself. In this study, we introduce a graph-theoretic approach to analyzing fractal boundaries using edge pixel adjacency. By extracting edge pixels from binary representations of fractals and constructing graphs based on 8-connected neighborhood relationships, we reveal how boundary components connect—or fragment—at a fixed resolution. We apply this method to five classical fractals and visualize their five largest connected edge components. The resulting structures provide insight into the differences between deterministic and chaotic fractals, offering a reproducible, scalable framework for analyzing local topological structure. This method complements traditional visualizations by focusing on boundary connectivity, revealing patterns not readily apparent through iteration count or color mapping alone.

Keywords: fractal geometry, edge detection, graph theory, pixel adjacency, Mandelbrot set, Julia set, Sierpinski triangle, Sierpinski carpet, Burning Ship fractal, binary image analysis, connected components, fractal boundary, graph-based visualization, image topology, computational geometry, local structure, 8-connected neighborhood, structural fragmentation, deterministic fractals, chaotic fractals. A collaboration with GPT-4o. 

**Abstract**

This study presents a graph-theoretic framework for analyzing the boundary structure of classical fractals through edge pixel adjacency. At a fixed resolution of 1000×1000 pixels, we examine five well-known fractals—the Mandelbrot set, a connected Julia set, the Sierpinski Carpet, the Sierpinski Triangle, and the Burning Ship fractal—by extracting their binary edge maps and converting them into undirected graphs based on 8-connected pixel neighborhoods. Within each graph, we identify and visualize the five largest connected components to assess local and global connectivity of the boundary structure.

Our results reveal marked differences in edge connectivity and fragmentation that correspond to the underlying generative principles of each fractal. Deterministic fractals such as the Sierpinski Carpet and Triangle display substantial fragmentation, with multiple disconnected boundary regions arising from their recursive geometric voids. In contrast, chaotic fractals like the Mandelbrot and Julia sets maintain a dominant, highly connected edge component even in discrete form. The Burning Ship fractal exhibits mixed characteristics, with a large core component and irregular peripheral fragmentation.

The proposed method offers a reproducible, scalable complement to traditional fractal visualizations by focusing explicitly on boundary topology rather than global form or color encoding.

## 1. Introduction

Fractals occupy a central position in modern mathematics and computational modeling due to their unique ability to describe complex, self-similar structures across scales. Introduced formally by Benoît Mandelbrot in the late 20th century, fractals have since become widely recognized in both theoretical and applied contexts, appearing in disciplines as diverse as physics, biology, computer graphics, and signal processing. Their recursive structure and non-integer dimensionality allow them to model natural phenomena such as coastlines,

vascular systems, turbulence, and crystal growth—domains where traditional Euclidean geometry often falls short.

The visual representation of fractals has historically focused on two main approaches. One, typically applied to sets generated by iterative complex functions (e.g., the Mandelbrot or Julia sets), involves escape-time algorithms where pixel values are color-coded according to how quickly points escape to infinity. The other, often used for deterministic fractals such as the Sierpinski Triangle or Carpet, is based on recursive geometric removal and rendered as binary patterns. While visually striking, both methods tend to emphasize global appearance or iteration depth at the expense of fine-grained boundary structure.

This work is motivated by the observation that fractal boundaries—where the transition from interior to exterior occurs—harbor significant topological information that is not well-captured in traditional renderings. In particular, these boundaries can be viewed as discrete graphs constructed from adjacent edge pixels, where each pixel acts as a node and connections are defined through local adjacency. By framing the problem in terms of graph theory, we gain access to a wide array of analytical tools capable of characterizing fragmentation, connectivity, and local structure.

The central goal of this study is to analyze and compare the edge connectivity of five classical fractals by rendering them as binary images at a fixed resolution, extracting their boundary pixels, and constructing undirected adjacency graphs based on 8-connected pixel neighborhoods. We then identify and visualize the five largest connected components of each edge graph. This consistent and resolution-independent framework allows for a direct comparison of how different fractal types organize their boundary structures, revealing distinctions between chaotic and deterministic forms that are not easily observed through traditional visualization techniques.

## 2. Methodology

This section outlines the procedures used to generate binary fractal images, detect edge pixels, construct adjacency graphs, and extract and visualize the largest connected components from those graphs. The approach is designed to be consistent across all fractal types, enabling direct comparison of their boundary structures.

### 2.1 Fractals Studied

We selected five classical fractals that are well-known for their visual complexity and mathematical richness. Each represents a distinct class of fractal generation—either deterministic and geometric or chaotic and escape-time-based—thereby offering a meaningful basis for comparison.

**Mandelbrot Set**

The Mandelbrot set consists of all complex numbers $c \in \mathbb{C}$ for which the recursive sequence $z_{n+1} = z_n^2 + c$, starting from $z_0 = 0$, remains bounded. It is one of the most studied and visually iconic fractals, with a complex boundary structure that exhibits infinite self-similarity and embedded miniature copies of itself.

**Julia Set ($c = -0.4 + 0.6i$)**

Julia sets are closely related to the Mandelbrot set but are generated by fixing a specific complex value $c$ and iterating $z_{n+1} = z_n^2 + c$ for each point $z_0$ in the complex plane. The connectivity and shape of the resulting set depend heavily on the value of $c$. We chose $c = -0.4 + 0.6i$, which yields a connected Julia set with a richly textured boundary.

**Sierpinski Carpet**

The Sierpinski Carpet is a deterministic fractal generated by recursively removing the central third of a square from a grid of nine equal parts. This process is repeated on each remaining sub-square, creating a self-similar structure with an increasingly porous interior and a highly fragmented boundary.

**Sierpinski Triangle**

The Sierpinski Triangle is another geometric fractal, constructed by recursively removing the central triangle from an equilateral triangle and repeating this operation on the three resulting sub-triangles. The result is a structure composed entirely of sharp angular segments and voids, exhibiting both self-similarity and significant boundary sparsity.

**Burning Ship Fractal**

The Burning Ship fractal is defined similarly to the Mandelbrot set but with the real and imaginary parts of the complex number taken as absolute values before squaring. Specifically, the iteration is given by $z_{n+1} = (|\operatorname{Re}(z_n)| + i|\operatorname{Im}(z_n)|)^2 + c$.

This results in a structure with sharp vertical ridges and horizontal striations, giving it an appearance reminiscent of a ship in flames.

## 2.2 Image Generation

Each of the five fractals was rendered at a fixed resolution of 1000×1000 pixels. For the Mandelbrot, Julia, and Burning Ship fractals, the standard escape-time algorithm was employed: each point in the complex plane was iterated up to a maximum of 100 iterations, and points that did not escape were marked as belonging to the set. For the Sierpinski Triangle and Carpet, recursive geometric algorithms were used to apply hole-removal operations over multiple levels of iteration.

Once generated, each image was converted into a binary array, where pixels belonging to the fractal were marked with a value of 1 (black) and all others with a value of 0 (white). This binary format served as the input for edge detection and subsequent graph construction.

---

### 2.3 Edge Detection and Graph Construction

To isolate the boundary structure of each fractal, we applied a pixel-level edge detection algorithm based on 8-connected neighborhood analysis. For each pixel marked as part of the fractal (value = 1), we examined its surrounding 3×3 grid. If any of the neighboring pixels had a value of 0 (indicating exterior space), the central pixel was marked as an edge pixel.

Each edge pixel was then treated as a node in an undirected graph. Edges were formed between nodes that were directly adjacent (up to 8 directions), preserving the spatial relationships of the boundary in the image domain. This resulted in a graph whose topology directly reflects the local connectivity of the fractal boundary. All graphs were constructed and processed using the open-source Python library NetworkX, which provides efficient tools for building and analyzing large-scale networks.

---

### 2.4 Component Ranking and Visualization

Once the adjacency graph was constructed for each fractal, we applied NetworkX's built-in algorithm to identify all connected components—subsets of nodes in which any two nodes are linked by a path of edges. These components were then ranked by size, defined as the number of edge pixels (nodes) in each component.

For each fractal, the five largest connected components were selected for visualization. To enhance interpretability, a consistent color scheme was applied: the largest component is shown in black, followed by red, blue, green, and purple for the second through fifth components, respectively. Each set of components

was plotted using a scatter plot in image coordinates, where pixel positions were preserved, and the y-axis was flipped to match standard image orientation.

This visualization method highlights both dominant structural regions and peripheral fragments of the fractal boundary, allowing for intuitive comparisons of edge organization across fractal types.

## 3. Results

The edge graphs extracted from the 1000×1000 binary images of five classical fractals reveal diverse patterns of connectivity, fragmentation, and structural organization. For each fractal, we identified and visualized the five largest connected components derived from the graph of 8-connected edge pixels. The results are presented below, organized by fractal type.

---

### 3.1 Mandelbrot and Julia Sets

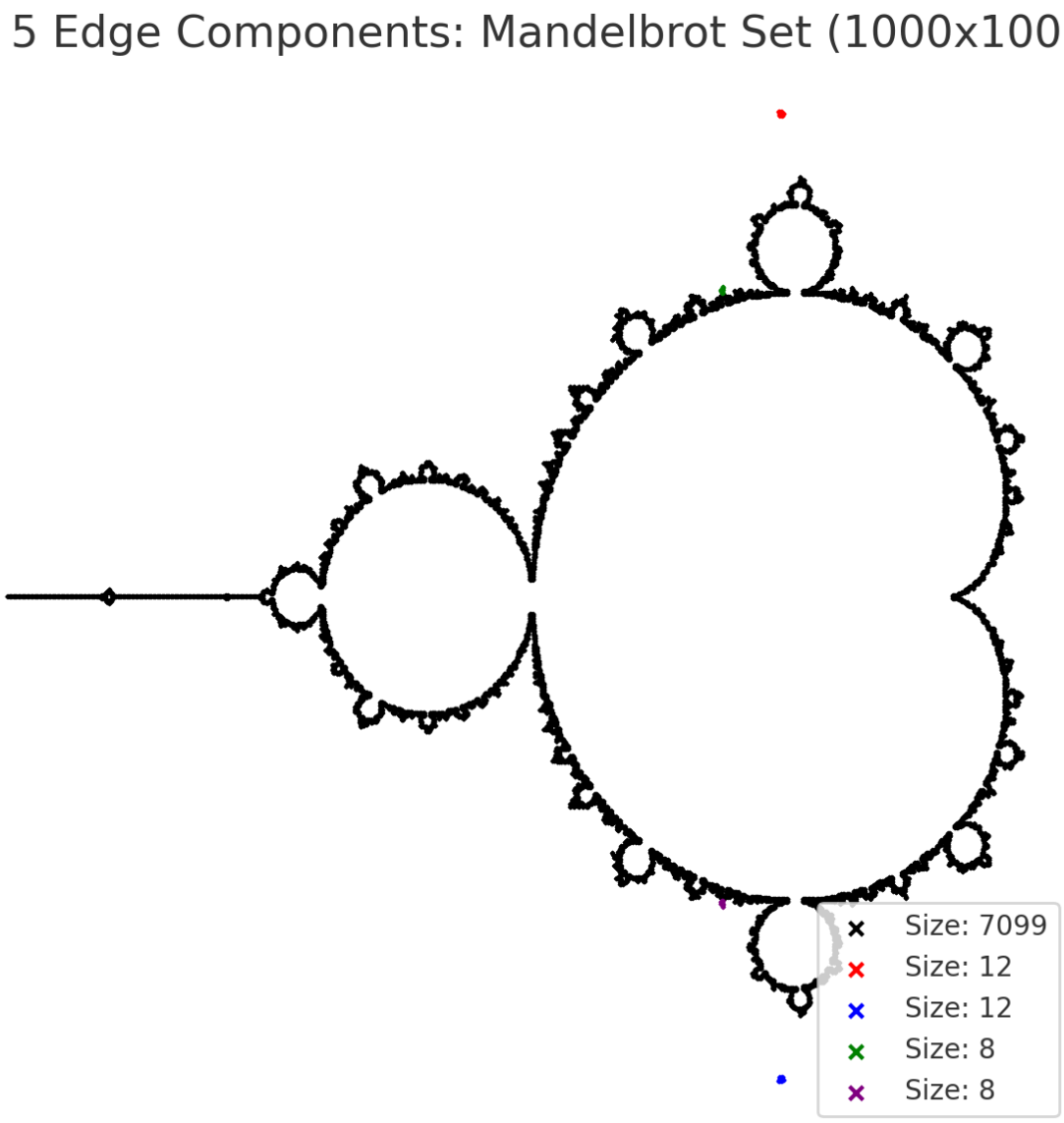
The Mandelbrot and Julia sets, both generated via escape-time dynamics, exhibit similar structural traits when viewed through the lens of edge graph analysis. In both cases, the topological dominance of a single, large boundary component is striking. This primary component, rendered in black, accounts for the majority of edge pixels and forms a coherent, closed structure surrounding the interior of the set.

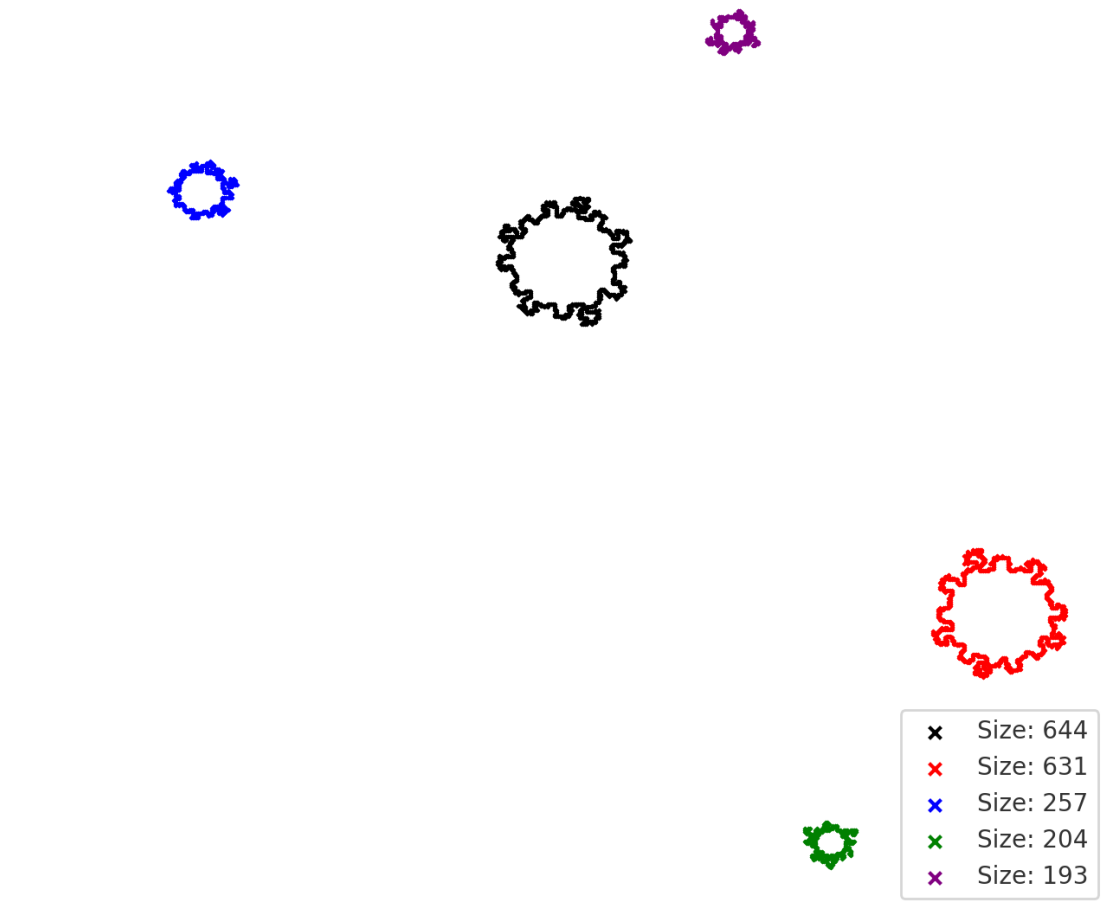
In the Mandelbrot set, the central cardioid and surrounding circular lobes remain fully connected in the edge graph. Smaller satellite features—such as miniature Mandelbrot replicas—are captured as peripheral components, appearing in red, blue, green, and purple. These secondary components are sparse and spatially separated, consistent with the fractal's recursive embeddings and disconnected "islands" in the complex plane.

The Julia set (with $c=-0.4+0.6i$) shows a similarly unified edge boundary. The largest component tightly encircles the fractal's interior in a branching, tree-like formation. A few minor components appear at the outermost edges of the image,

possibly representing disconnected filaments or noise at the resolution limit, but they are negligible in both size and structural significance.

These results confirm that chaotic fractals with continuous escape boundaries tend to preserve large-scale edge connectivity, even under discrete sampling.






---

### 3.2 Sierpinski Carpet and Triangle

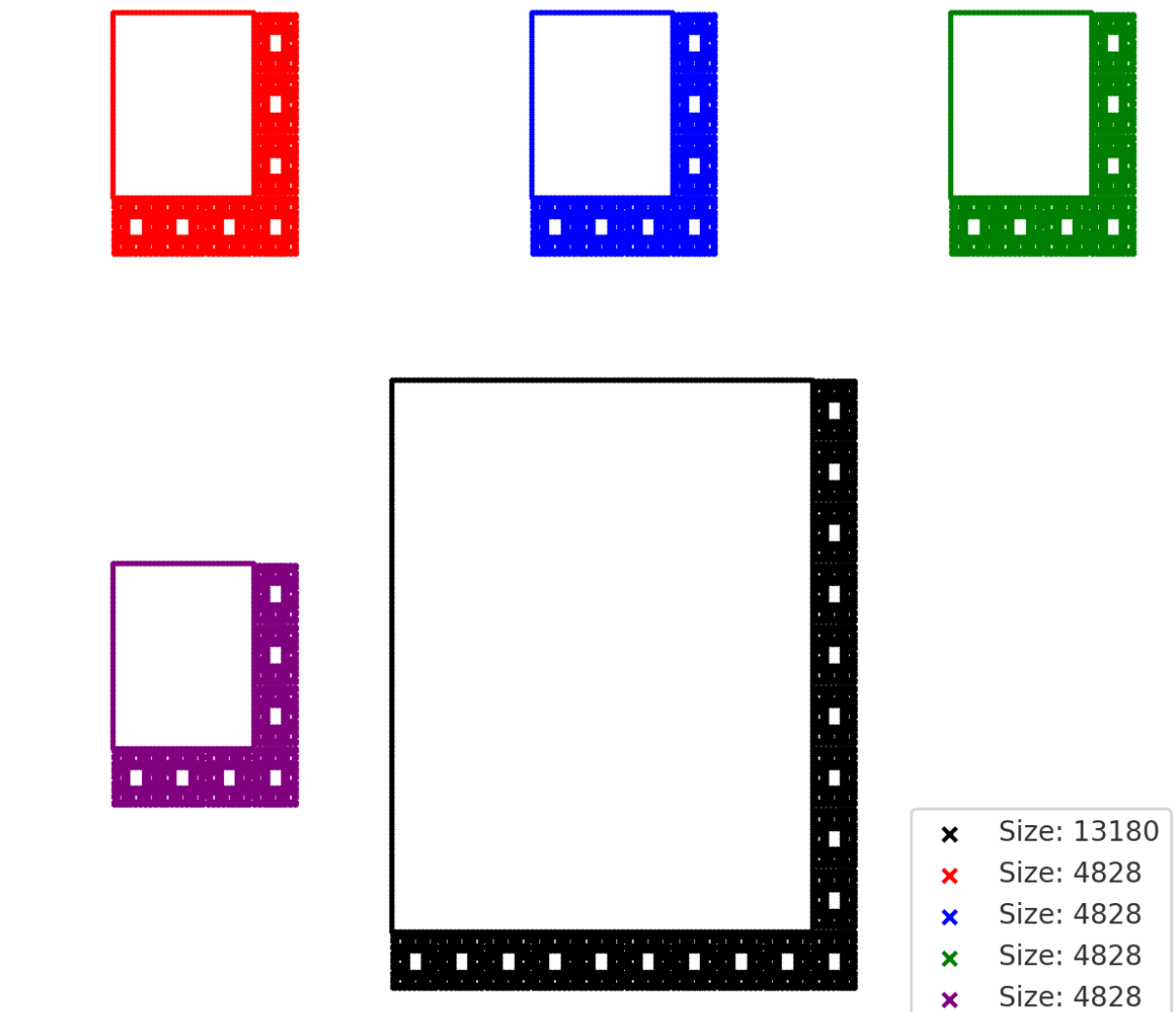
The edge graphs of the Sierpinski Carpet and Sierpinski Triangle reveal a dramatically different structural pattern compared to the chaotic fractals. As deterministic fractals constructed through recursive geometric removal, both exhibit heavy fragmentation of their boundary structures, even at 1000×1000 resolution.

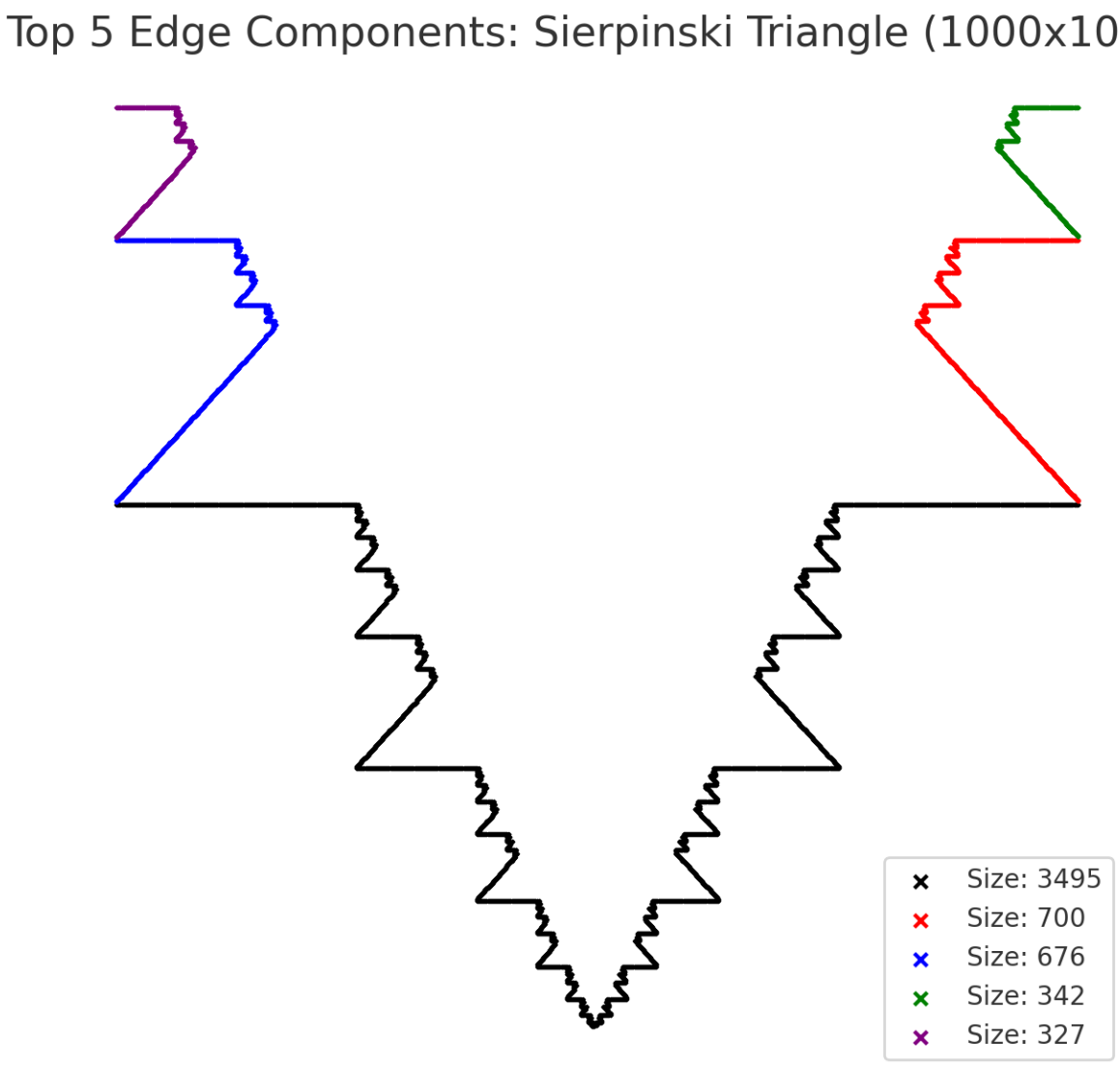
In the Sierpinski Carpet, the recursive removal of central squares results in multiple fully disconnected edge regions. Notably, entire square voids are left without surrounding edge pixels if they are completely embedded within filled

regions. This leads to the formation of several large but spatially isolated edge components, each corresponding to a different part of the recursive construction. The largest component is typically centered within the image, while others occupy corners and outer ring segments.

The Sierpinski Triangle, with its angular structure and recursive subtractions, produces a scattered and irregular edge graph. The sharp tips and thin connective arms that theoretically link parts of the triangle are insufficiently resolved at this pixel density, resulting in disconnected components. Each of the five largest components occupies a distinct triangular sub-region, corresponding to different levels in the recursion hierarchy.

Together, these two deterministic fractals highlight how recursive void generation leads to true topological fragmentation, which is faithfully captured through edge graph analysis.


Top 5 Edge Components: Sierpinski Carpet (1000x1000)
Size: 13180
Size: 4828
Size: 4828
Size: 4828
Size: 4828

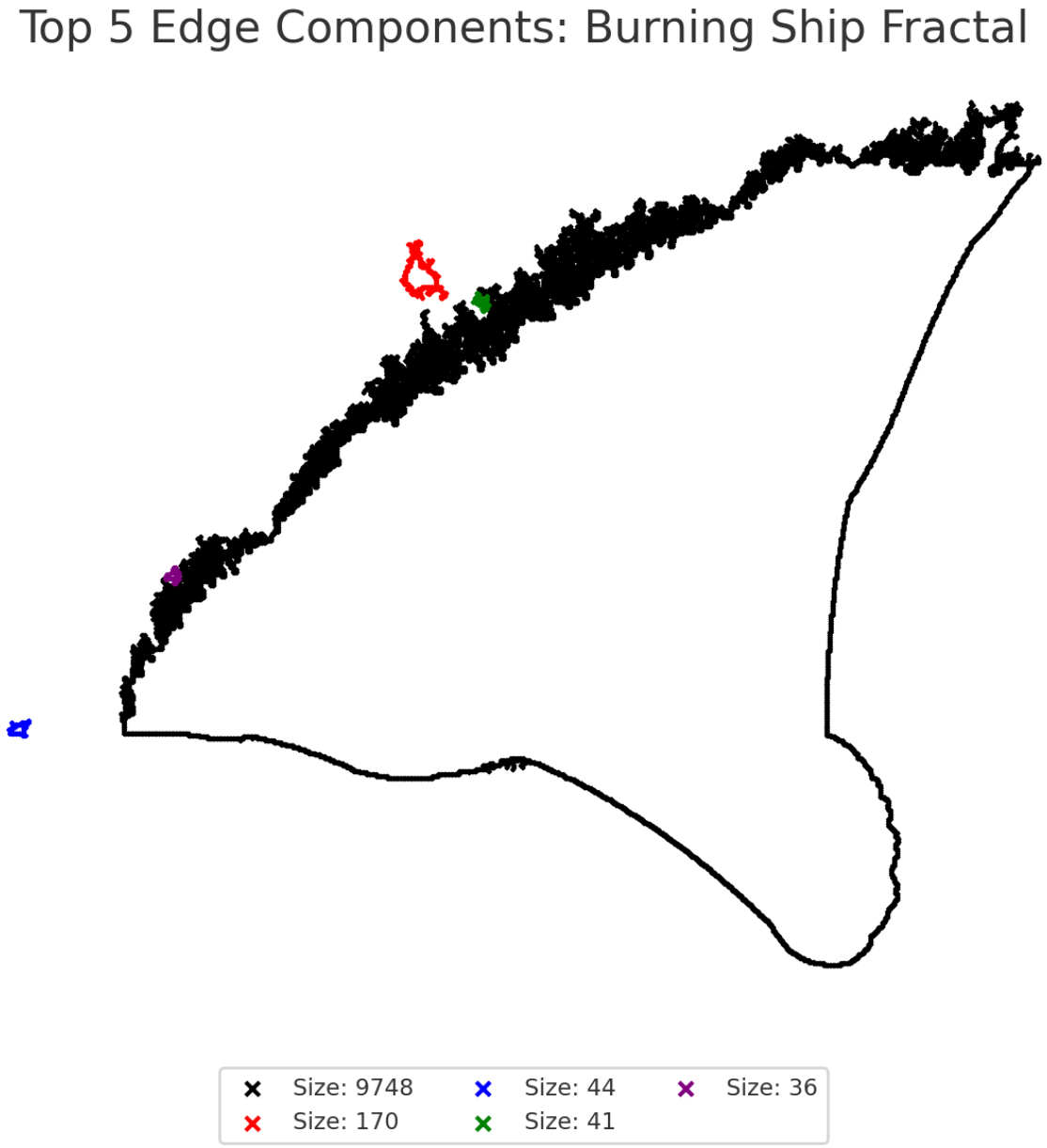



### 3.3 Burning Ship Fractal

The Burning Ship fractal presents a hybrid case, exhibiting traits of both chaotic and deterministic systems. Its edge graph reveals a large, asymmetric core component, accompanied by a collection of irregularly shaped secondary fragments. Unlike the smooth perimeter of the Mandelbrot or Julia sets, the primary boundary in the Burning Ship is jagged and uneven, characterized by vertical ridges, horizontal striations, and sharp bends.

The largest component spans the central mass of the fractal and retains connectivity across its vertically stretched shape. The secondary components are more irregularly placed and vary significantly in shape and size. Some appear as extended filaments, while others are compact and isolated.

This pattern reflects the nonlinear folding and absolute-value operations used in the fractal's definition, which produce a visual structure that is highly sensitive to initial conditions. The result is an edge graph that is partly unified but also exhibits meaningful peripheral disintegration.




---

These five visualizations collectively demonstrate the utility of edge graph analysis in revealing structural differences among fractals that may appear similar under traditional visualization techniques. By isolating the most connected regions of the boundary and highlighting fragmentation, this method offers a consistent and topology-sensitive lens for comparing diverse fractal forms.

## 4. Discussion

The edge graph analysis presented in this study reveals consistent and interpretable differences in boundary structure across the five fractals examined. These differences are strongly linked to the generative logic of each fractal—whether deterministic or chaotic—and are made visible through a uniform methodology that emphasizes local connectivity.

Deterministic fractals such as the Sierpinski Carpet and Sierpinski Triangle exhibit high degrees of fragmentation, even at a resolution of 1000×1000 pixels. Their construction relies on the recursive removal of specific geometric regions, which introduces sharp voids and angular gaps that disrupt continuity in the boundary. As a result, their edge graphs are composed of multiple large but disconnected components, each corresponding to isolated substructures within the fractal. These disconnections are not artifacts of resolution but are instead intrinsic to the fractals' recursive definitions. In such cases, traditional visualizations may emphasize symmetry and repetition, but the edge graph reveals the true topological separation between regions.

In contrast, chaotic fractals like the Mandelbrot Set and the Julia Set demonstrate remarkably coherent edge structures, with a single large component dominating the graph. Despite the complexity of their appearance, the boundary remains topologically connected throughout the image space. This suggests a form of global continuity that persists even under pixel discretization. The few smaller components that do exist are spatially distant or peripheral, likely corresponding to isolated lobes or miniature embedded structures.

The Burning Ship fractal occupies an intermediate space. While it retains a dominant central edge component, its jagged and asymmetric shape leads to irregular fragmentation along its periphery. This mixed behavior reflects the non-standard dynamics of its generation, which include nonlinearity and directional bias.

Importantly, this graph-theoretic approach isolates boundary behavior that is often obscured in traditional visualizations, where iteration counts or recursive

symmetry may dominate the viewer's perception. By focusing exclusively on edge connectivity, the method removes color and depth as interpretive cues and instead foregrounds the topological structure of the fractal boundary itself.

Beyond visualization, the edge graph provides a foundation for fine-grained spatial analysis. Metrics such as the number of connected components, graph diameter, or clustering could be used to classify fractals or detect transitions in structural organization. This opens the possibility of using edge topology not merely for interpretation, but for quantitative classification of fractal types—an area with potential relevance in both theoretical studies and applied pattern recognition tasks involving fractal-like forms.

In summary, the edge graph method serves as a lens through which structural distinctions emerge between deterministic and chaotic fractals, grounded not in aesthetics but in connectivity. This reorientation from global form to local adjacency allows for a richer understanding of how fractals organize their boundaries—and how those boundaries behave under discretization.

## 5. Conclusion

This study demonstrates that edge graph analysis at a fixed resolution provides a scalable, reproducible, and focused method for investigating the boundary complexity of classical fractals. By representing edge pixels as nodes in an 8-connected adjacency graph, we isolate and visualize the local topological structure of fractal boundaries in a way that is both rigorous and intuitive.

Unlike traditional rendering techniques that rely on iteration counts, coloring schemes, or recursive patterning, this method makes no assumptions about internal dynamics or aesthetic presentation. Instead, it emphasizes connectivity, fragmentation, and spatial organization at the pixel level—features that often remain hidden in standard visualizations. This allows for a direct and unambiguous comparison of structural behavior across fractal types.

The results clearly distinguish between deterministic fractals, which exhibit high degrees of boundary fragmentation, and chaotic fractals, which maintain coherent

and globally connected edge structures. These observations are consistent across the five fractals studied and suggest that boundary graphs can serve as meaningful descriptors of fractal class and behavior.

Beyond visualization, the method opens avenues for further quantitative analysis. Graph-based metrics—such as component size distribution, clustering, or spectral properties—could provide additional insights into the geometric and topological properties of fractals. As such, edge graph analysis stands as a valuable complement to existing approaches, offering a minimalist yet powerful perspective on fractal form.

**References**


1. Mandelbrot, B. B. (1982). *The Fractal Geometry of Nature*. W. H. Freeman and Company.
   — The foundational text introducing the concept of fractals and their self-similarity across scales.

2. Peitgen, H.-O., Jürgens, H., & Saupe, D. (2004). *Chaos and Fractals: New Frontiers of Science* (2nd ed.). Springer.
   — A comprehensive reference on the generation, visualization, and mathematical properties of fractals.

3. Falconer, K. (2003). *Fractal Geometry: Mathematical Foundations and Applications* (2nd ed.). Wiley.
   — A rigorous mathematical treatment of fractals, fractal dimensions, and measure theory.

4. Barnsley, M. F. (1988). *Fractals Everywhere*. Academic Press.
   — Covers deterministic fractal constructions such as the Sierpinski Triangle and Carpet.

5. Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4th ed.). Pearson.
   — An authoritative guide on image binarization, edge detection, and pixel connectivity.

6. Newman, M. E. J. (2010). *Networks: An Introduction*. Oxford University Press.
   — Essential reading for graph-theoretic methods, including component analysis and adjacency networks.

7. Tricot, C. (1995). *Curves and Fractal Dimension*. Springer.
   — A practical introduction to methods such as box-counting for analyzing fractal boundaries.

8. Feder, J. (1988). *Fractals*. Springer.
   — Provides accessible insight into physical and natural examples of fractals.

9. NetworkX Developers. (2023). *NetworkX: Network Analysis in Python*. https://networkx.org
— Documentation and tools used for constructing and analyzing edge pixel adjacency graphs.

10. van der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). *The NumPy Array: A Structure for Efficient Numerical Computation*. *Computing in Science & Engineering*, 13(2), 22–30.
— Describes the array-processing library used for fractal image generation and edge detection.

## Appendix A: Full Python Code

The following Python code provides a complete, reproducible implementation of the methods described in this paper. It includes all necessary steps for generating binary fractal images at a resolution of 1000×1000, extracting edge pixels using 8-connected adjacency, constructing graph representations of the boundaries, and visualizing the five largest connected components.

This code requires the following Python packages:

- numpy
- matplotlib
- networkx

All figures in this study were generated using these modules without additional dependencies.

---

### A.1 Imports and Global Parameters

python

CopyEdit

```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx


RESOLUTION = 1000
MAX_ITER = 100
PLOT_COLORS = ['black', 'red', 'blue', 'green', 'purple']
```

---

**A.2 Fractal Generators**

Mandelbrot Set

python

CopyEdit

```
def generate_mandelbrot_image(size=1000, max_iter=100):
    xmin, xmax = -2.0, 1.0
    ymin, ymax = -1.5, 1.5
    image = np.zeros((size, size), dtype=np.uint8)


    for i in range(size):
        for j in range(size):
            x = xmin + (xmax - xmin) * i / size
            y = ymin + (ymax - ymin) * j / size
            c = complex(x, y)
            z = 0
            for _ in range(max_iter):
                z = z*z + c
                if abs(z) > 2:
                    break
            else:
                image[j, i] = 1
    return image
```

Julia Set

python

CopyEdit

```
def generate_julia_image(c=-0.4+0.6j, size=1000, max_iter=100):
    xmin, xmax = -2.0, 1.0
    ymin, ymax = -1.5, 1.5
    image = np.zeros((size, size), dtype=np.uint8)

    for i in range(size):
        for j in range(size):
            x = xmin + (xmax - xmin) * i / size
            y = ymin + (ymax - ymin) * j / size
            z = complex(x, y)
            for _ in range(max_iter):
                z = z*z + c
                if abs(z) > 2:
                    break
            else:
                image[j, i] = 1
    return image
```

Burning Ship Fractal

python

CopyEdit

```
def generate_burning_ship_image(size=1000, max_iter=100):
```

```
    xmin, xmax = -2.0, 1.5
    ymin, ymax = -2.0, 1.5
    image = np.zeros((size, size), dtype=np.uint8)

    for i in range(size):
        for j in range(size):
            x0 = xmin + (xmax - xmin) * i / size
            y0 = ymin + (ymax - ymin) * j / size
            x, y = 0.0, 0.0
            iteration = 0
            while x*x + y*y <= 4 and iteration < max_iter:
                x, y = abs(x), abs(y)
                xtemp = x*x - y*y + x0
                y = 2*x*y + y0
                x = xtemp
                iteration += 1
            if iteration == max_iter:
                image[j, i] = 1
    return image
```

Sierpinski Carpet

python

CopyEdit

```
def generate_sierpinski_carpet(size=1000):
```

```
    image = np.ones((size, size), dtype=np.uint8)

    def remove_carpet(x, y, s):
        if s < 3:
            return
        step = s // 3
        image[y+step:y+2*step, x+step:x+2*step] = 0
        for dx in range(3):
            for dy in range(3):
                if dx == 1 and dy == 1:
                    continue
                remove_carpet(x + dx*step, y + dy*step, step)

    remove_carpet(0, 0, size)
    return image
```

Sierpinski Triangle

python

CopyEdit

```
def generate_sierpinski_triangle(size=1000):
    image = np.ones((size, size), dtype=np.uint8)

    def remove_triangle(x, y, s):
        if s < 2:
```

```
        return
    h = s // 2
    for i in range(h):
        image[y + i, x + h - i : x + h + i + 1] = 0
    remove_triangle(x, y, h)
    remove_triangle(x + h, y, h)
    remove_triangle(x + h//2, y + h, h)

remove_triangle(0, 0, size)
return image
```

---

**A.3 Edge Pixel Detection**

```
python
CopyEdit
def extract_edge_pixels(binary_image):
    edge_image = np.zeros_like(binary_image)
    for i in range(1, binary_image.shape[0] - 1):
        for j in range(1, binary_image.shape[1] - 1):
            if binary_image[i, j] == 1:
                neighbors = binary_image[i-1:i+2, j-1:j+2]
                if np.any(neighbors == 0):
                    edge_image[i, j] = 1
    return edge_image
```

**A.4 Graph Construction and Component Extraction**

python

CopyEdit

```
def build_edge_graph(edge_image):
    coords = np.argwhere(edge_image == 1)
    coord_to_index = {tuple(coord): idx for idx, coord in enumerate(coords)}
    G = nx.Graph()

    for idx, (i, j) in enumerate(coords):
        G.add_node(idx)
        for di in [-1, 0, 1]:
            for dj in [-1, 0, 1]:
                if di == 0 and dj == 0:
                    continue
                ni, nj = i + di, j + dj
                if (ni, nj) in coord_to_index:
                    neighbor_idx = coord_to_index[(ni, nj)]
                    G.add_edge(idx, neighbor_idx)
    return G, coords

def extract_top_components(G, coords, num_components=5):
    top_components = sorted(nx.connected_components(G), key=len,
reverse=True)[:num_components]
```

```
    return [np.array([coords[node] for node in comp]) for comp in
top_components]
```

---

**A.5 Visualization**

```
python

CopyEdit

def plot_colored_components(component_list, title, size=1000):

    plt.figure(figsize=(6, 6))

    for coords, color in zip(component_list, PLOT_COLORS):

        plt.scatter(coords[:, 1], size - coords[:, 0] - 1, s=1, c=color, label=f'Size:
{len(coords)}')

    plt.title(f"Top 5 Edge Components: {title}")

    plt.axis('off')

    plt.legend(markerscale=5, loc='lower right')

    plt.tight_layout()

    plt.show()
```

---

**A.6 Example Usage**

```
python

CopyEdit

# Example for Mandelbrot

image = generate_mandelbrot_image()

edges = extract_edge_pixels(image)

G, coords = build_edge_graph(edges)
```

```
top_components = extract_top_components(G, coords)
plot_colored_components(top_components, "Mandelbrot Set")
```

---