Sierpinski Figures Documentation

Welcome to the documentation for the Sierpinski Figures project. This project provides tools for generating and visualizing Sierpinski fractals in both 2D and 3D spaces.

Getting Started

Core Components

3D Sierpinski Pyramid

The Sierpinski Pyramid, also known as the Sierpinski Tetrahedron, is a 3D fractal formed by recursively subdividing a tetrahedron into smaller tetrahedrons.

Implementation

The implementation uses the midpoint displacement method to generate the points of the Sierpinski Pyramid.

Example Usage

```python
from sierpinski.shapes_3d import SierpinskiPyramid

# Create a pyramid with 4 iterations
pyramid = SierpinskiPyramid(iterations=4)
fig = pyramid.plot_3d()
fig.show()  # Opens interactive 3D visualization
fig.write_html("pyramid.html")  # Save as interactive HTML
```

Visualization

The 3D Sierpinski Pyramid is visualized using Plotly. The points are plotted iteratively, and the process can be animated to show the formation of the fractal.

Algorithm Explanation

1. **Vertices Definition**: The vertices of the initial tetrahedron are defined as points `A`, `B`, `C`, and `D`.
2. **Starting Point**: A starting point inside the tetrahedron is defined.
3. **Plot Initialization**: The initial tetrahedron and plots for the current point, line to vertex, and trajectory are set up.
4. **Frame Generation**: For each frame:

    - A random vertex is chosen.
    - The current point is updated to the midpoint between the current point and the chosen vertex.
    - The points and frames are updated with the new current point and line to the vertex.
5. **Layout and Animation**: The layout for the 3D plot is defined, and the animation is created using Plotly's `go.Figure` and `go.Frame`.

Mathematical Background

The Sierpinski Pyramid is constructed by recursively removing the central tetrahedron from each subdivided tetrahedron. This process continues indefinitely, creating a self-similar fractal pattern.

2D Sierpinski Triangle

The Sierpinski Triangle is a fractal that is formed by recursively subdividing an equilateral triangle into smaller equilateral triangles.

Implementation

The implementation uses the midpoint displacement method to generate the points of the Sierpinski Triangle.

Example Usage

```
from sierpinski.shapes_2d import SierpinskiTriangle

# Create a triangle with 6 iterations
triangle = SierpinskiTriangle(iterations=6)
fig = triangle.plot()
fig.show()  # Opens in browser or notebook
fig.write_html("triangle.html")  # Save as interactive HTML
```

Visualization

The 2D Sierpinski Triangle is visualized using matplotlib. The points are plotted iteratively, and the process can be animated to show the formation of the fractal.

Algorithm Explanation

1. **Vertices Definition**: The vertices of the initial equilateral triangle are defined as points `A`, `B`, and `C`.
2. **Starting Point**: A random starting point inside the triangle is generated using a convex combination of the vertices.
3. **Plot Initialization**: The initial triangle and plots for the current point, line to vertex, and trajectory are set up.
4. **Animation Initialization**: The `init` function initializes the plots.
5. **Update Function**: The `update` function is called for each frame of the animation. It:
    - Chooses a random vertex.
    - Updates the current point plot.
    - Updates the line plot to the chosen vertex.
    - Calculates the midpoint between the current point and the chosen vertex.
    - Updates the scatter plot with the trajectory.
6. **Animation Creation**: The `FuncAnimation` function creates the animation using the `update` function and the frames.

Mathematical Background

The Sierpinski Triangle is constructed by recursively removing the central triangle from each subdivided triangle. This process continues indefinitely, creating a self-similar fractal pattern.

Theory: Convergence of the Chaos Game to the Sierpinski Triangle

Introduction

The chaos game is a simple iterative process that remarkably generates the Sierpinski triangle. This document provides a formal mathematical proof of why this occurs.

Definitions

Definition 1: The Chaos Game

Let T be an equilateral triangle with vertices v_1, v_2, v_3. The chaos game consists of: 1. Selecting an initial point p_0 2. Iteratively generating points p_n by: - Randomly selecting a vertex v_i - Setting p_n as the midpoint between p_{n-1} and v_i

Definition 2: Sub-triangle at Prefix x

For a ternary string x, a sub-triangle T(x) with vertices $\{v_1(x), v_2(x), v_3(x)\}$ is defined recursively: - T(ε) is the base triangle (empty prefix) - T(x‖i) is the triangle formed by $v_i(x)$ and the midpoints of its adjacent edges

Definition 3: Sierpinski Triangle

The Sierpinski triangle S(x) at prefix x is defined recursively as: S(x) = T(x) ∪ S(x‖1) ∪ S(x‖2) ∪ S(x‖3)

Mathematical Representation

Tridrant Notation

Any Sierpinski sub-triangle can be uniquely identified using a ternary string representing its path: - Each digit (1,2,3) represents which tridrant was chosen - Empty string ε represents the base triangle - String concatenation (x‖i) represents selecting tridrant i in sub-triangle x

For example: “132” represents: 1. First tridrant of base triangle 2. Third tridrant of resulting sub-triangle 3. Second tridrant of that sub-triangle

Core Lemmas

Lemma 1: Invariance Property

For any point p ∈ S and vertex v chosen by the chaos game, the resulting point p’ also lies in S.

Proof: Let x be the shortest prefix such that p ∈ S(x). When moving halfway to vertex v: 1. T(x) maps to T(v‖x) 2. By definition, T(v‖x) ⊂ S ∴ p’ ∈ S

Example: Consider a point p = (0.25, 0.25) in the Sierpinski triangle: - If v_1 = (0, 0) is

chosen, p' = (0.125, 0.125) - If $v_2 = (1, 0)$ is chosen, p' = (0.625, 0.125) - If $v_3 = (0.5, 0.866)$ is chosen, p' = (0.375, 0.558) All resulting points remain within the fractal.

Lemma 2: Distance Convergence

For any point $p \notin S$, let p* be its closest point in S. After n steps: $d(p_n, p_n) \leq (1/2)^n d(p_0, p_0)$

Proof: Each iteration halves the distance between corresponding points: $d(p_{n+1}, p_{n+1}) = \|(p_n + v)/2 - (p_n + v)/2\| = d(p_n, p_n^*)/2$

Example: Start with $p_0 = (0.4, 0.4) \notin S$ - Initially $d_0 = 0.1$ (distance to nearest point in S) - After 3 iterations: $d_3 \leq 0.0125$ - After 10 iterations: $d_{10} \leq 0.0001$

Lemma 3: Complete Coverage

For any point $q \in S$ and $\varepsilon > 0$, the probability of the chaos game visiting a point within ε of q approaches 1.

Proof Sketch: 1. Any point in S can be represented by an infinite ternary sequence 2. The chaos game generates random ternary sequences 3. By the law of large numbers, all finite prefixes occur with probability 1

Example: To reach point $q = (0.333, 0.289) \in S$ within $\varepsilon = 0.01$: 1. Need sequence: v_1, v_3, v_2, v_1, v_3 2. Probability $\approx (1/3)^5$ for this exact sequence 3. Multiple sequences lead to ε-neighborhood

Convergence Properties

Rate of Convergence

For a point p_0 starting distance d from S: - After n steps: $d_n \leq d/2^n$ - To reach within ε: need approximately $\log_2(d/\varepsilon)$ steps - Typical convergence in 600 steps for $\varepsilon \approx 10^{-3}$

Additional Theoretical Results

Hausdorff Dimension

The Sierpinski triangle has Hausdorff dimension: $D = \log(3)/\log(2) \approx 1.585$

Proof Outline: 1. Triangle splits into 3 self-similar copies 2. Each copy scaled by factor 1/2 3. D satisfies $3 = (2)^D$

Extended Properties

Non-equilateral Triangles

The proof holds for any triangle because: - Arguments are topological, not geometric - Only relative positions matter - Result is topologically equivalent but geometrically skewed

Modified Ratios

For ratio $\lambda \neq 1/2$: - New fractal shapes emerge - Convergence still occurs but to different attractors - $\lambda > 1/2$: More "stretched" towards vertices - $\lambda < 1/2$: More "compressed" towards center

Conclusion

The chaos game converges to the Sierpinski triangle because: 1. Points on S remain on S

(Lemma 1) 2. Points off S converge exponentially to S (Lemma 2) 3. All points in S are eventually approximated (Lemma 3)

Extensions

The proof framework extends to: - Non-equilateral triangles (via topological equivalence) - Different ratios $\lambda \neq 1/2$ (producing modified fractals) - Other polygons (generating different recursive structures)

Implementation Notes

Numerical Considerations

- Due to floating-point arithmetic, points should be considered equal if their distance is below a small epsilon (typically 1e-10)
- After approximately 20 iterations, points off S become visually indistinguishable from points on S

Interactive Demonstrations

The theoretical results can be verified using our interactive tools: - Use `chaos_game.py` to visualize Lemma 1's invariance property - `convergence_visualizer.py` demonstrates the exponential convergence from Lemma 2 - `coverage_analyzer.py` provides empirical evidence for Lemma 3

Numerical Analysis

Convergence Rate

For practical implementations: - First 10 iterations: Rapid convergence towards S - Next 100 iterations: Fine structure development - Beyond 1000 iterations: Detail refinement

Error Bounds

For a starting point p_0 with distance d_0 from S: - After n iterations: $d_n \leq d_0/2^n$ - To achieve precision ε: $n \geq \log_2(d_0/\varepsilon)$

References

1. Barnsley, M. F. (1988). Fractals Everywhere. Academic Press.
2. Chaganty, A. T. (2020). "Why does the chaos game converge to the Sierpinski triangle?" Retrieved from https://arun.chagantys.org/technical/2020/04/28/chaos-game.html
3. Gleick, J. (1987). Chaos: Making a New Science. Viking Press.
4. Sierpiński, W. (1915). "Sur une courbe dont tout point est un point de ramification." Comptes Rendus de l'Académie des Sciences, 160, 302-305.

Usage Guide

Quick Start

Installation

This guide provides detailed instructions on how to use the Sierpinski Figures project to

generate and visualize fractals.

2D Sierpinski Triangle

Creating a Sierpinski Triangle

```python
from sierpinski.shapes_2d import SierpinskiTriangle

# Create a triangle with 6 iterations
triangle = SierpinskiTriangle(iterations=6)
fig = triangle.plot()
fig.show()  # Opens in browser or notebook
fig.write_html("triangle.html")  # Save as interactive HTML
```

Visualizing the Triangle

The 2D Sierpinski Triangle is visualized using matplotlib. The points are plotted iteratively, and the process can be animated to show the formation of the fractal.

Algorithm Explanation

1. **Vertices Definition**: The vertices of the initial equilateral triangle are defined as points `A`, `B`, and `C`.
2. **Starting Point**: A random starting point inside the triangle is generated using a convex combination of the vertices.
3. **Plot Initialization**: The initial triangle and plots for the current point, line to vertex, and trajectory are set up.
4. **Animation Initialization**: The `init` function initializes the plots.
5. **Update Function**: The `update` function is called for each frame of the animation. It:
   - Chooses a random vertex.
   - Updates the current point plot.
   - Updates the line plot to the chosen vertex.
   - Calculates the midpoint between the current point and the chosen vertex.
   - Updates the scatter plot with the trajectory.
6. **Animation Creation**: The `FuncAnimation` function creates the animation using the `update` function and the frames.

3D Sierpinski Pyramid

Creating a Sierpinski Pyramid

```python
from sierpinski.shapes_3d import SierpinskiPyramid

# Create a pyramid with 4 iterations
pyramid = SierpinskiPyramid(iterations=4)
fig = pyramid.plot_3d()
fig.show()  # Opens interactive 3D visualization
fig.write_html("pyramid.html")  # Save as interactive HTML
```

Visualizing the Pyramid

The 3D Sierpinski Pyramid is visualized using Plotly. The points are plotted iteratively, and the process can be animated to show the formation of the fractal.

Algorithm Explanation

1. **Vertices Definition**: The vertices of the initial tetrahedron are defined as points `A`, `B`, `C`, and `D`.
2. **Starting Point**: A starting point inside the tetrahedron is defined.
3. **Plot Initialization**: The initial tetrahedron and plots for the current point, line to

vertex, and trajectory are set up.
4. **Frame Generation**: For each frame:
    - A random vertex is chosen.
    - The current point is updated to the midpoint between the current point and the chosen vertex.
    - The points and frames are updated with the new current point and line to the vertex.
5. **Layout and Animation**: The layout for the 3D plot is defined, and the animation is created using Plotly's `go.Figure` and `go.Frame`.

Additional Features

Plotly Visualization

- Interactive 3D rotation and zoom
- Hover information showing coordinates
- Export to HTML for interactive web viewing
- Multiple color schemes and styling options
- Camera position control
- Screenshot capabilities

Saving Figures

Both 2D and 3D figures can be saved as interactive HTML files using the `write_html` method. This allows you to share the visualizations easily.

Customization

You can customize the appearance of the figures by modifying the parameters in the `plot` and `plot_3d` methods. This includes changing colors, sizes, and other styling options.