|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**  **TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**  **Ảnh có chứa Đồ họa, biểu tượng, hình mẫu, Phông chữ  Nội dung do AI tạo ra có thể không chính xác.**  **BÀI TẬP TRÊN LỚP**  **MÔN ĐỒ HỌA MÁY TÍNH – CS105.P22**  **BÀI TIỂU LUẬN FRACTAL**  **Giảng viên: Cáp Phạm Đình Thăng**  **Sinh viên thực hiện:**   |  |  | | --- | --- | | **MSSV** | **Họ tên** | | **Đinh Hữu Phước** | **22521150** | | **Nguyễn Trần Khương An** | **22520026** | | **Huỳnh Yến Nhi** | **22521035** | | **Trần Quang Đạt** | **22520236** |   **Tp. Hồ Chí Minh, 4/2025** |

**NHẬN XÉT CỦA GIÁO VIÊN**

**.........................................................................................................................................**

**.........................................................................................................................................**

**.........................................................................................................................................**

**.........................................................................................................................................**

**.........................................................................................................................................**

**.........................................................................................................................................**

**.........................................................................................................................................**

**.........................................................................................................................................**

**.........................................................................................................................................**

**.........................................................................................................................................**

**.........................................................................................................................................**

**.........................................................................................................................................**

**......................................................................................................................................**

**.........................................................................................................................................**

**.........................................................................................................................................**

**......................................................................................................................................... .........................................................................................................................................**

**.........................................................................................................................................**

**Mục lục**

[**I. Tìm hiểu về cách vẽ bông tuyết Vankoch (Koch Snowflake)** 5](#_Toc195136125)

[1. Giới thiệu 5](#_Toc195136126)

[2. Ý tưởng tổng quát 5](#_Toc195136127)

[3. Đặc điểm hình học 5](#_Toc195136128)

[4. Ứng dụng 6](#_Toc195136129)

[5. Source code 6](#_Toc195136130)

[6. Giải thích mã JavaScript 7](#_Toc195136131)

[7. Hình ảnh minh họa 8](#_Toc195136132)

[**II. Tìm hiểu về cách vẽ đảo Minkowski** 8](#_Toc195136133)

[1. Giới thiệu 8](#_Toc195136134)

[2. Ý tưởng tổng quát 8](#_Toc195136135)

[3. Tính chất hình học 9](#_Toc195136136)

[4. Ứng dụng 9](#_Toc195136137)

[5. Source code 9](#_Toc195136138)

[6. Giải thích đoạn mã 11](#_Toc195136139)

[7. Hình ảnh minh họa 12](#_Toc195136140)

[**III. Tìm hiểu về cách vẽ tam giác Sierpinski (Triangles) và Hình vuông Sierpinski (Carpet)** 13](#_Toc195136141)

[1. Tam giác Sierpinski (Sierpinski Triangle) 13](#_Toc195136142)

[1.1 Giới thiệu 13](#_Toc195136143)

[1.2 Ý tưởng tổng quát 14](#_Toc195136144)

[1.3 Đặc điểm toán học 14](#_Toc195136145)

[1.4 Source code 14](#_Toc195136146)

[1.5 Phân tích đoạn mã 15](#_Toc195136147)

[2. Hình vuông Sierpinski (Sierpinski Carpet) 15](#_Toc195136148)

[2.1 Giới thiệu 15](#_Toc195136149)

[2.2 Ý tưởng tổng quát 15](#_Toc195136150)

[2.3. Đặc điểm toán học 16](#_Toc195136151)

[2.4 Source code 16](#_Toc195136152)

[2.5 Phân tích đoạn mã 16](#_Toc195136153)

[3. So sánh Sierpinski Triangle và Carpet 17](#_Toc195136154)

[4. Hình ảnh minh họa 17](#_Toc195136155)

[**IV. Tìm hiểu và trình bày tập Mandelbrot và Julia Set.** 18](#_Toc195136156)

[1. Tập Mandelbrot 18](#_Toc195136157)

[1.1 Định nghĩa 18](#_Toc195136158)

[1.2 Hình dạng và đặc điểm 18](#_Toc195136159)

[2. Julia Set 18](#_Toc195136160)

[2.1 Định nghĩa 18](#_Toc195136161)

[2.2 Hình dạng và đặc điểm 19](#_Toc195136162)

[3. Ứng dụng và ý nghĩa 19](#_Toc195136163)

[4. Kết luận 19](#_Toc195136164)

[5. Hình ảnh minh họa 20](#_Toc195136165)

[**VII Source code** 20](#_Toc195136166)

[**VIII. Tài liệu tham khảo** 20](#_Toc195136167)

## 

## **I. Tìm hiểu về cách vẽ bông tuyết Vankoch (Koch Snowflake)**

## **1. Giới thiệu**

Bông tuyết Von Koch, hay Koch Snowflake, là một trong những hình dạng fractal cổ điển đầu tiên được mô tả. Nó được tạo thành từ ba cạnh của một tam giác đều, mỗi cạnh trải qua quá trình chia nhỏ lặp đi lặp lại để tạo thành các chi tiết ngày càng phức tạp. Mỗi lần lặp (iteration) làm tăng độ chi tiết của cạnh, nhưng tổng chiều dài lại tăng vô hạn – một đặc trưng của fractal.

## **2. Ý tưởng tổng quát**

Quy trình tạo hình bắt đầu từ một tam giác đều. Với mỗi cạnh, ta thực hiện quy trình sau:

1. Chia cạnh thành 3 phần bằng nhau.
2. Tạo một tam giác đều nhỏ nhô ra ngoài từ đoạn giữa.
3. Thay thế đoạn giữa bằng hai cạnh của tam giác vừa tạo.
4. Lặp lại quy trình trên với tất cả các cạnh mới sinh ra.

## **3. Đặc điểm hình học**

* Với mỗi lần lặp, số cạnh tăng gấp 4 lần, độ dài mỗi cạnh giảm còn 1/3.
* Chu vi tăng dần và tiến tới vô hạn.
* Diện tích hội tụ về một giá trị hữu hạn.
* Có tính chất tự đồng dạng (self-similarity) rõ rệt – đặc trưng của fractal.

## **4. Ứng dụng**

* Thể hiện các quy luật hình học tự nhiên: cấu trúc tuyết, mô hình đường bờ biển, phân dạng tự nhiên.
* Minh họa trong lý thuyết hỗn loạn, toán học fractal và đồ họa máy tính.
* Sử dụng để rèn luyện tư duy đệ quy trong lập trình và phân tích hình học.

## **5. Source code**

|  |
| --- |
| function drawKochSnowflake(ctx, level) {    const size = 400;    const height = size \* Math.sqrt(3) / 2;    const centerX = 400;    const centerY = 400;    const A = { x: centerX, y: centerY - height / 2 };    const B = { x: centerX - size / 2, y: centerY + height / 2 };    const C = { x: centerX + size / 2, y: centerY + height / 2 };    ctx.beginPath();    ctx.moveTo(A.x, A.y);    drawKoch(ctx, A, B, level);    drawKoch(ctx, B, C, level);    drawKoch(ctx, C, A, level);    ctx.stroke();  }  function drawKoch(ctx, p1, p2, level) {    if (level === 0) {      ctx.lineTo(p2.x, p2.y);      return;    }    const dx = (p2.x - p1.x) / 3;    const dy = (p2.y - p1.y) / 3;    const pa = { x: p1.x + dx, y: p1.y + dy };    const pb = { x: p1.x + 2 \* dx, y: p1.y + 2 \* dy };    const baseAngle = Math.atan2(dy, dx);    const angle = baseAngle + Math.PI / 3;    const length = Math.hypot(dx, dy);    const peak = {      x: pa.x + Math.cos(angle) \* length,      y: pa.y + Math.sin(angle) \* length    };    drawKoch(ctx, p1, pa, level - 1);    drawKoch(ctx, pa, peak, level - 1);    drawKoch(ctx, peak, pb, level - 1);    drawKoch(ctx, pb, p2, level - 1);  } |

## **6. Giải thích mã JavaScript**

**6.1. Hàm drawKochSnowflake(ctx, level)**

* Xác định kích thước và vị trí tam giác đều ban đầu.
* Gọi đệ quy hàm drawKoch() cho ba cạnh: AB, BC, và CA.
* ctx là đối tượng canvas 2D để vẽ, còn level là số cấp độ lặp (mức fractal).

**6.2. Hàm đệ quy drawKoch(ctx, p1, p2, level)**

Hàm này vẽ một cạnh fractal từ điểm p1 đến p2:

* Cơ sở đệ quy (level = 0): vẽ đoạn thẳng từ p1 đến p2.
* Đệ quy:
  + Tính hai điểm chia cạnh thành ba phần: pa, pb.
  + Tính góc giữa đoạn p1 → p2.
  + Xác định điểm đỉnh của tam giác đều mới peak nhô ra ngoài.
  + Gọi đệ quy cho bốn đoạn con:
    - p1 → pa
    - pa → peak
    - peak → pb
    - pb → p2

## **7. Hình ảnh minh họa**

|  |  |  |
| --- | --- | --- |
| **Level = 2** | **Level = 3** | **Level = 5** |
|  |  |  |

## **II. Tìm hiểu về cách vẽ đảo Minkowski**

## **1. Giới thiệu**

Đảo Minkowski là một hình fractal hình học được tạo ra bằng cách biến đổi từng cạnh của hình vuông (hoặc bất kỳ đa giác nào) thành một đường có hình vuông nhô ra theo quy tắc lặp lại. Hình này còn gọi là "Minkowski sausage" hay "coastline fractal", do nó mô phỏng cách đường bờ biển có thể phức tạp và dài hơn khi ta đo ở độ phân giải cao hơn.

## **2. Ý tưởng tổng quát**

Bắt đầu từ một hình vuông, ta thay mỗi cạnh bằng một đường gấp khúc gồm nhiều đoạn nhỏ hơn, trong đó có các hình vuông nhô ra ngoài và vào trong.

Với mỗi cấp độ level, ta:

1. Chia cạnh thành 4 phần bằng nhau.
2. Giữ nguyên đoạn đầu và cuối.
3. Thêm hai hình vuông nhỏ:
   * Một nhô ra ngoài theo hướng vuông góc.
   * Một nhô vào trong, đối xứng với cái nhô ra.
4. Thay thế đoạn cạnh ban đầu bằng chuỗi các đoạn này.
5. Lặp lại quy trình trên cho tất cả các cạnh và tất cả cấp độ tiếp theo.

## **3. Tính chất hình học**

* Mỗi lần lặp tăng số đoạn cạnh và độ chi tiết.
* Chu vi của đảo Minkowski tăng theo cấp số nhân và có thể tiến tới vô hạn khi level tăng.
* Diện tích hội tụ (giống như Koch snowflake).
* Có tính tự đồng dạng và tính fractal rõ rệt.

## **4. Ứng dụng**

* Mô phỏng địa lý (đường bờ biển, địa hình phức tạp).
* Mô hình hóa trong đồ họa máy tính và nghệ thuật số.
* Minh họa về cách đo chiều dài phụ thuộc vào độ phân giải đo (liên quan đến "paradox of coastline length").

## **5. Source code**

|  |
| --- |
| function drawMinkowskiIsland(ctx, level) {    const size = 300;    const startX = 250;    const startY = 250;    const square = [      { x: startX, y: startY + size },  // Bottom-left      { x: startX + size, y: startY + size },  // Bottom-right      { x: startX + size, y: startY },  // Top-right      { x: startX, y: startY }          // Top-left    ];    ctx.beginPath();    ctx.moveTo(square[0].x, square[0].y);    for (let i = 0; i < 4; i++) {      const from = square[i];      const to = square[(i + 1) % 4];      const path = generateMinkowskiEdge(from, to, level);      for (let j = 1; j < path.length; j++) {        ctx.lineTo(path[j].x, path[j].y);      }    }    ctx.closePath();    ctx.stroke();  }  function generateMinkowskiEdge(p1, p2, level) {    if (level === 0) return [p1, p2];    const dx = p2.x - p1.x;    const dy = p2.y - p1.y;    const length = Math.sqrt(dx \* dx + dy \* dy);    const dirX = dx / length;    const dirY = dy / length;    const segment = length / 4;    const points = [];    // Đoạn 1 – giữ nguyên    const A = {      x: p1.x + dirX \* segment,      y: p1.y + dirY \* segment    };    points.push(...generateMinkowskiEdge(p1, A, level - 1));    // Đoạn 2 – hình vuông nhô ra xuống/phải    const B1 = {      x: A.x + dirX \* segment,      y: A.y + dirY \* segment    };    const normalDir = {      x: -dirY, // vuông góc với (dirX, dirY)      y: dirX    };    const B2 = {      x: A.x + normalDir.x \* segment,      y: A.y + normalDir.y \* segment    };    const B3 = {      x: B2.x + dirX \* segment,      y: B2.y + dirY \* segment    };    points.push(...generateMinkowskiEdge(A, B2, level - 1));    points.push(...generateMinkowskiEdge(B2, B3, level - 1));    points.push(...generateMinkowskiEdge(B3, B1, level - 1));    // Đoạn 3 – hình vuông nhô ra lên/trái    const flipNormal = {      x: -normalDir.x,      y: -normalDir.y    };    const C1 = {      x: B1.x + flipNormal.x \* segment,      y: B1.y + flipNormal.y \* segment    };    const C2 = {      x: C1.x + dirX \* segment,      y: C1.y + dirY \* segment    };    const C3 = {      x: B1.x + dirX \* segment,      y: B1.y + dirY \* segment    };    points.push(...generateMinkowskiEdge(B1, C1, level - 1));    points.push(...generateMinkowskiEdge(C1, C2, level - 1));    points.push(...generateMinkowskiEdge(C2, C3, level - 1));    // Đoạn 4 – giữ nguyên    points.push(...generateMinkowskiEdge(C3, p2, level - 1));    return points;  } |

## **6. Giải thích đoạn mã**

**6.1. Hàm chính drawMinkowskiIsland(ctx, level)**

* Xác định vị trí và kích thước ban đầu của hình vuông.
* Gọi hàm generateMinkowskiEdge() cho từng cạnh để tạo đường fractal.
* Dùng ctx.lineTo() để vẽ các đoạn fractal lên canvas.

**6.2. Hàm generateMinkowskiEdge(p1, p2, level)**

Hàm này đệ quy sinh ra các đoạn fractal từ đoạn thẳng p1 → p2.

Trường hợp cơ sở:

* Nếu level == 0, trả về đoạn thẳng [p1, p2].

Trường hợp đệ quy:

* Tính hướng và chiều dài đoạn ban đầu.
* Tính vector pháp tuyến để xây các hình vuông nhô ra/vào.
* Chia đoạn thành 4 phần:
  1. Giữ nguyên đoạn đầu.
  2. Tạo hình vuông nhô ra.
  3. Tạo hình vuông nhô vào.
  4. Giữ nguyên đoạn cuối.

Mỗi đoạn con lại được gọi đệ quy cho tới khi đạt level = 0.

Chi tiết hình vuông nhô ra:

Dùng vector vuông góc với hướng cạnh để tính điểm nhô ra khỏi đường thẳng ban đầu. Sau đó tạo 3 đoạn cạnh vuông nhỏ từ điểm A → B2 → B3 → B1.

Hình vuông nhô vào:

Tương tự, nhưng dùng vector pháp tuyến đảo chiều để nhô vào.

## **7. Hình ảnh minh họa**

|  |  |
| --- | --- |
| **Level = 2** |  |
| **Level = 3** |  |
| **Level = 5** |  |

## **III. Tìm hiểu về cách vẽ tam giác Sierpinski (Triangles) và Hình vuông Sierpinski (Carpet)**

## **1. Tam giác Sierpinski (Sierpinski Triangle)**

## **1.1 Giới thiệu**

Tam giác Sierpinski là một trong những fractal hình học nổi tiếng nhất, được xây dựng bằng cách lặp lại thao tác chia nhỏ một tam giác đều thành các tam giác con. Fractal này có tính chất tự đồng dạng (self-similar), nghĩa là toàn bộ hình trông giống chính nó ở mọi cấp độ thu nhỏ.

## **1.2 Ý tưởng tổng quát**

* Bắt đầu từ một tam giác đều.
* Chia tam giác đó thành 4 tam giác nhỏ bằng cách nối trung điểm các cạnh.
* Bỏ tam giác nhỏ ở giữa (hình ngược).
* Tiếp tục áp dụng quy tắc này cho 3 tam giác còn lại.
* Lặp lại quy trình này theo số cấp độ (level).

## **1.3 Đặc điểm toán học**

* Mỗi lần lặp tăng số tam giác theo quy luật: 3n3^n3n
* Diện tích **giảm dần và tiến tới 0** khi level → ∞
* Đường biên trở nên vô hạn – một đặc trưng của fractal.
* Có **dimension fractal** bằng:

|  |
| --- |
|  |

## **1.4 Source code**

|  |
| --- |
| function drawSierpinskiTriangle(ctx, p1, p2, p3, level) {      if (level === 1) {        ctx.beginPath();        ctx.moveTo(p1.x, p1.y);        ctx.lineTo(p2.x, p2.y);        ctx.lineTo(p3.x, p3.y);        ctx.closePath();        ctx.fill();        return;      }        // Trung điểm mỗi cạnh      const m1 = midpoint(p1, p2);      const m2 = midpoint(p2, p3);      const m3 = midpoint(p3, p1);        drawSierpinskiTriangle(ctx, p1, m1, m3, level - 1);      drawSierpinskiTriangle(ctx, m1, p2, m2, level - 1);      drawSierpinskiTriangle(ctx, m3, m2, p3, level - 1);    }      function midpoint(p1, p2) {      return {        x: (p1.x + p2.x) / 2,        y: (p1.y + p2.y) / 2      };    } |

## **1.5 Phân tích đoạn mã**

Hàm **drawSierpinskiTriangle(ctx, p1, p2, p3, level)**

* Tham số:
  + ctx: ngữ cảnh canvas để vẽ.
  + p1, p2, p3: ba đỉnh của tam giác.
  + level: mức đệ quy (cấp độ fractal).
* Hoạt động:
  + Nếu level === 1, vẽ tam giác đặc.
  + Nếu level > 1:
    - Tìm trung điểm của 3 cạnh.
    - Tạo 3 tam giác nhỏ và gọi đệ quy cho từng tam giác

## **2. Hình vuông Sierpinski (Sierpinski Carpet)**

## **2.1 Giới thiệu**

Hình vuông Sierpinski, hay Sierpinski Carpet, là một fractal được xây dựng bằng cách chia hình vuông thành 9 phần bằng nhau và bỏ phần ở giữa, rồi lặp lại thao tác trên từng ô còn lại.

## **2.2 Ý tưởng tổng quát**

* Bắt đầu với một hình vuông lớn.
* Chia thành 9 ô vuông nhỏ (3×3).
* Bỏ ô trung tâm.
* Lặp lại quy trình với 8 ô còn lại.
* Cứ như vậy đến cấp độ đệ quy cuối.

## **2.3. Đặc điểm toán học**

* Số ô được giữ lại sau mỗi cấp: 8n8^n8n
* Diện tích giảm dần:

|  |
| --- |
|  |

* Khi n → ∞, diện tích tiến đến 0, nhưng chiều dài biên tiến tới vô hạn.
* Fractal dimension:

|  |
| --- |
|  |

## **2.4 Source code**

|  |
| --- |
| function drawSierpinski(ctx, x, y, size, level) {      if (level === 1) {        ctx.fillRect(x, y, size, size);        return;      }      const newSize = size / 3;      for (let dx = 0; dx < 3; dx++) {        for (let dy = 0; dy < 3; dy++) {          if (dx === 1 && dy === 1) continue; // bỏ ô trung tâm          drawSierpinski(ctx, x + dx \* newSize, y + dy \* newSize, newSize, level - 1);        }      }    } |

## **2.5 Phân tích đoạn mã**

Hàm **drawSierpinski(ctx, x, y, size, level)**

* **Tham số:**
  + x, y: tọa độ góc trên trái của hình vuông hiện tại.
  + size: kích thước cạnh hình vuông.
  + level: cấp độ fractal.
* **Hoạt động:**
  + Nếu level === 1, vẽ hình vuông đặc tại vị trí (x, y) có kích thước size.
  + Nếu level > 1, chia hình vuông thành 9 phần (3x3 lưới):
    - Bỏ ô giữa (dx = 1, dy = 1)
    - Gọi đệ quy cho 8 ô còn lại.

## **3. So sánh Sierpinski Triangle và Carpet**

|  |  |  |
| --- | --- | --- |
| **Thuộc tính** | **Sierpinski Triangle** | **Sierpinski Carpet** |
| **Hình ban đầu** | Tam giác đều | Hình vuông |
| **Số mảnh sau mỗi lần lặp** | 3 | 8 |
| **Quy tắc loại bỏ** | Tam giác giữa | Ô vuông giữa |
| **Dimension fractal** | Xấp xỉ 1.585 | Xấp xỉ 1.893 |

## **4. Hình ảnh minh họa**

|  |  |  |
| --- | --- | --- |
| **Level = 3** |  |  |
| **Level = 5** |  |  |
| **Level = 7** |  |  |

## **IV. Tìm hiểu và trình bày tập Mandelbrot và Julia Set.**

## **1. Tập Mandelbrot**

### **1.1 Định nghĩa**

Tập Mandelbrot là tập hợp các điểm **c∈C** sao cho dãy sau không tiến ra vô cực khi lặp đi lặp lại:

|  |
| --- |
|  |

* Nếu dãy **zn** không phân kỳ, thì **c** thuộc tập Mandelbrot.
* Nếu dãy phân kỳ (nghĩa là ∣zn∣→∞) thì **c** không thuộc tập **Mandelbrot**.

### **1.2 Hình dạng và đặc điểm**

* Tập Mandelbrot nằm trong mặt phẳng phức và có hình dạng tổng thể giống như **một hình trái tim (cardioid)** kết hợp với một vòng tròn.
* Bao quanh nó là vô số nhánh phụ nhỏ hơn có hình dạng tương tự toàn bộ – thể hiện tính **tự đồng dạng (self-similarity).**
* Biên giới của tập Mandelbrot có cấu trúc fractal, nghĩa là không mịn và có độ chi tiết vô hạn – khi phóng to lên bao nhiêu lần, vẫn thấy các chi tiết mới xuất hiện.

## **2. Julia Set**

### **2.1 Định nghĩa**

Julia Set cũng bắt nguồn từ hàm lặp:

|  |
| --- |
|  |

Tuy nhiên, khác với Mandelbrot, ở đây:

* **c** là **hằng số cố định**.
* Ta thay đổi **giá trị khởi tạo z0​** trong mặt phẳng phức **C**.
* Tập Julia của một giá trị c là tập các điểm z0 sao cho dãy zn **không phân kỳ**.

**Mối liên hệ với Mandelbrot:**

* Nếu c **thuộc tập Mandelbrot**, thì Julia Set tương ứng sẽ **liên thông** (connected).
* Nếu c **không thuộc Mandelbrot**, thì Julia Set tương ứng sẽ là **một tập rời rạc** (disconnected), giống như **bụi (dust-like fractal)**.

### **2.2 Hình dạng và đặc điểm**

* Mỗi giá trị **c** khác nhau sẽ tạo ra **một Julia Set khác nhau** – tức là có vô số Julia Sets tương ứng với các giá trị ccc.
* Các Julia Set có cấu trúc fractal: thường **đối xứng, phức tạp**, và mang tính nghệ thuật cao.
* Julia Set thể hiện tính nhạy cảm với điều kiện ban đầu – đặc trưng của các hệ động lực học phi tuyến (chaotic systems).

## **3. Ứng dụng và ý nghĩa**

* Trong khoa học máy tính: Sinh ảnh fractal, đồ họa máy tính.
* Trong toán học: Nghiên cứu về hệ động lực học, hỗn loạn, fractals.
* Trong nghệ thuật: Tạo hình ảnh mang tính đối xứng và trừu tượng.
* Mô phỏng sự tăng trưởng, lan truyền hoặc phân bố trong tự nhiên.

## **4. Kết luận**

Tập Mandelbrot và Julia không chỉ là đối tượng toán học đẹp về mặt hình ảnh mà còn sâu sắc về mặt lý thuyết. Chúng minh họa cho sự phức tạp có thể sinh ra từ những quy tắc đơn giản, và là cầu nối giữa hình học, đại số và hệ động lực học. Những khám phá về hai tập này không chỉ thay đổi cách chúng ta nhìn về hình học mà còn tác động tới nhiều lĩnh vực khác như khoa học máy tính, vật lý, và thậm chí cả nghệ thuật.

## **5. Hình ảnh minh họa**

|  |  |
| --- | --- |
| **Mandelbrot** | **Julia Set** |
|  |  |

## **VII. Source code**

**Link:** <https://github.com/DatTran0509/CS105-DoHoaMayTinh/tree/main/TieuLuan_FourGuy?fbclid=IwY2xjawJjh8dleHRuA2FlbQIxMAABHu2Nx4_oc_-jD6lmKCtxY9T2Kg9vtGD5_XqAjGHQ0YlI5gicR2-He45ViBPu_aem_TgnaKCzt09U5VCqoPlHFAQ>

## **VIII. Tài liệu tham khảo**

### <https://adammurray.link/webgl/#basic-examples>

### <http://www.shodor.org/interactivate/activities/KochSnowflake/>

### <http://www.shodor.org/interactivate/activities/SierpinskiTriangle/>