**Fractal Image Compression Process Explanation**

## **Introduction**

Fractal Image Compression (FIC) is a lossy compression technique that exploits self-similarity within images. Instead of storing pixel values, FIC stores a set of mathematical transformations that can recreate the image

**Encoding (Compression) Process:**

Step-by-Step:

**1.Divide the Image into Range Blocks**:

* **Range Blocks(R)**: Divide the original image into small non-overlapping blocks (e.g., 4×4 or 8×8 pixels). These are the target blocks we want to encode.
* **Domain Blocks(D)**: Create a pool of larger blocks (typically 2× the size of range blocks) using overlapping sections of the image. These will serve as "source material" for our transformations

.**2. Search for Best Match**:

For each range block R, find the domain block D that can be transformed to most closely resemble R.

The transformation involves:

* **Geometric Transformation**: Scaling (reduction from domain to range size), rotation (0°, 90°, 180°, 270°), and flipping (horizontal, vertical, or none)
* **Affine Transformation**: Contrast scaling (s) and brightness adjustment (o)

**3.Calculate Transformation**:

Record the transformation required to map the domain block to the range block:

* **Position of domain block**
* **Rotation** (0°, 90°, 180°, 270°)
* **Flip** (horizontal or vertical)
* **Brightness and contrast adjustments**

### 4.Storing the Compressed Data

For each range block, store:

* **Range Block ID**: Position/identifier of the range block
* **Domain Block ID**: Position/identifier of the matching domain block
* **Transformation Code**: Which rotation and/or flip was applied
* **Contrast Factor (s)**: The scaling factor for intensity values
* **Brightness Offset (o)**: The brightness adjustment value

This text file becomes the **only data needed to recreate the image**.

The text file example:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **RangeBlockID** | **DomainBlockID** | **Rotation (°)** | **Flip** | **Contrast** | **Brightness** |
| 0 | 5 | 0 | None (N) | 1.0 | 0 |
| 1 | 12 | 90 | Horizontal | 1.2 | -15 |
| 2 | 7 | 180 | Vertical | 0.9 | 10 |
| 3 | 20 | 270 | None (N) | 1.1 | -5 |

## **Understanding the Contrast Factor (s)**

The contrast factor (s) controls how the intensity values of the domain block are scaled before being applied to the range block:

* **s = 1.0**: Use the domain block's intensity exactly as is
* **s = 0.5**: Use the domain block at half intensity (darker)
* **s = 2.0**: Use the domain block at double intensity (brighter)
* **s = -1.0**: Use the domain block's inverted intensity (like a negative)

This factor allows us to match regions that have similar patterns but different intensity distributions. For example, a shadowed area might match a brightly lit area using a lower contrast factor.

## **Detailed Example: How Contrast Factor Works**

Consider these example blocks:

**Domain Block D (8×8):**

50 50 50 50 50 50 50 50

50 50 50 50 50 50 50 50

50 50 80 80 80 80 50 50

50 50 80 80 80 80 50 50

50 50 80 80 80 80 50 50

50 50 80 80 80 80 50 50

50 50 50 50 50 50 50 50

50 50 50 50 50 50 50 50

**Range Block R (4×4) we want to match:**

100 100 160 160

100 100 160 160

100 100 160 160

100 100 160 160

We can see that R appears to be a similar pattern to D but with double the intensity. In this case:

1. The domain block is downsampled to 4×4 (by averaging groups of 2×2 pixels):

50 50 50 50

50 80 80 50

50 80 80 50

50 50 50 50

1. The best transformation involves:

* No rotation or flip (geometric transformation)
* s = 2.0 (to double the intensity)
* b= 0 (no brightness offset needed)

1. The stored data would be:

RangeBlockID: X

DomainBlockID: Y

Transformation: None

Contrast: 2.0

Brightness: 0

When decompressing, we would take the domain block, apply s = 2.0 to all its values (doubling them), and use it to reconstruct the range block.

**Decoding (Decompression) Process:**

The goal is to **reconstruct the image** from the text-based transformation data.

#### Step-by-Step:

1. **Start with a Blank or Random Image**:

* Use a blank canvas or random noise image as the starting point.

### 2. Iterative Reconstruction

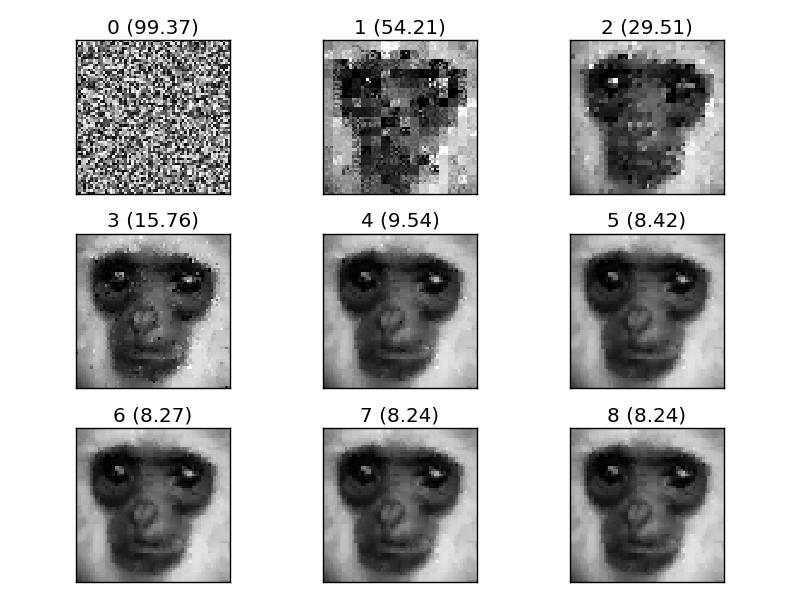
For each iteration:

* Process each range block using its stored transformation data
* Find the corresponding domain block
* Apply the geometric transformation (rotation/flip)
* Apply the contrast scaling and brightness offset
* Replace the range block with this transformed result

### 3. Repeat Until Convergence

* Continue this process for multiple iterations (typically 8-16)
* With each iteration, the image becomes clearer and closer to the original
* The process eventually converges due to the mathematics of iterated function systems

**Example :**

Here you can see a image output after each iteration   


**Key Takeaways**

* **Fractal compression** doesn’t store pixels — it stores **instructions** on how to recreate the image from its own parts.
* Compression is **slow** (due to searching and matching), but decompression is **fast**.
* Useful for images with lots of **self-similarity**, like natural textures.