Motilal Nehru National Institute of Technology, Allahabad

Project on

FRACTALS

Course: MCA I Semester – *Programming and Problem Solving*

Submitted To: Prof. M.M. Gore

Team Leader: Dibyendu Das (2025CA031)

Course: MCA 1st Semester

Subject: *Programming and Problem Solving*

Team Members:

- Dibyendu Das (Leader)
- Manish Kumar (Co-Leader)
- Amit Singhad
- Divyajeet
- Jitendra Singh Parmar
- Mohd Sufiyan
- Piyush Gautam
- Rajdeep Gupta
- Rajeev Tirkey
- Shri Krishan
- Smruti Ranjan Behera
- Sumit Raj Chaudhary

FRACTALS

Exploring the infinite complexity of the Mandelbrot, Julia, and Tricorn sets through escape-time algorithms

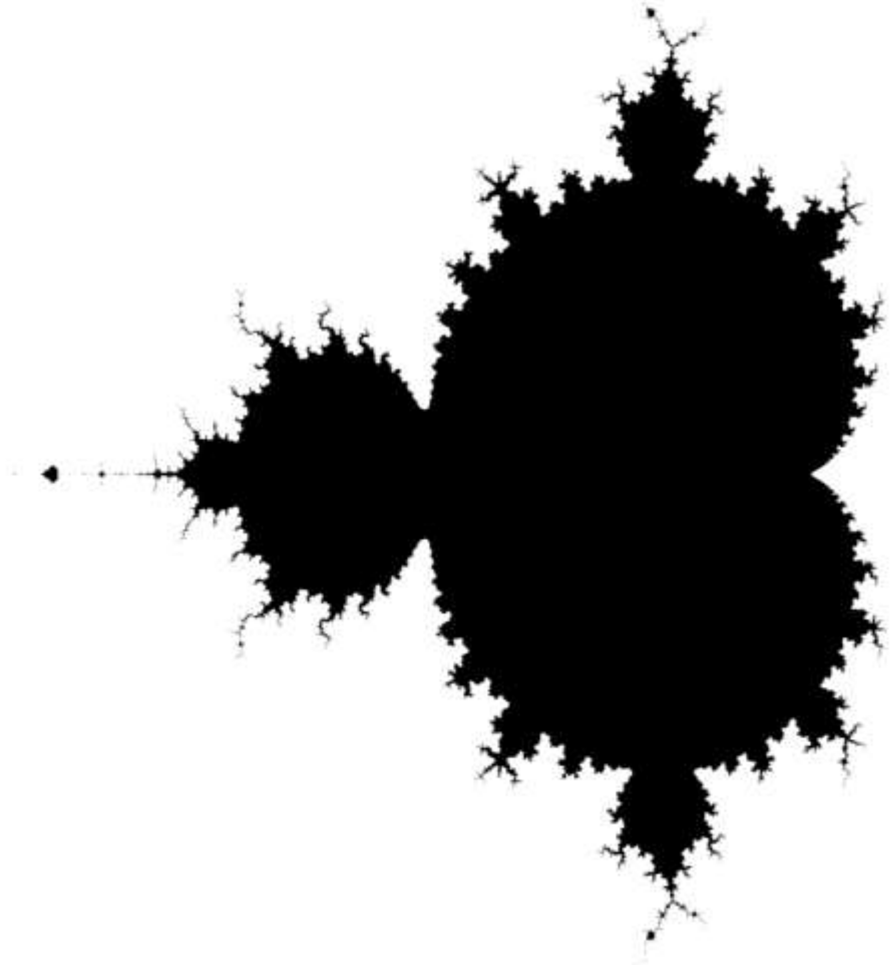
Fractals

Fractals are infinitely complex patterns that are self-similar across different scales. They are created by repeating a simple process over and over in an ongoing feedback loop.

Fractals are infinitely complex patterns that are self-similar across different scales. They are created by repeating a simple process over and over in an ongoing feedback loop.

Key Properties

- Self-similarity: Patterns repeat at every scale
- Infinite detail: Zooming reveals endless complexity
- Generated from simple equations
- Appears throughout nature and mathematics



Mathematical Foundation: The Mandelbrot Set

The Iteration Rule

At each point *c* in the complex plane, we iterate:

$$z_{n+1} = z_n^2 + c$$

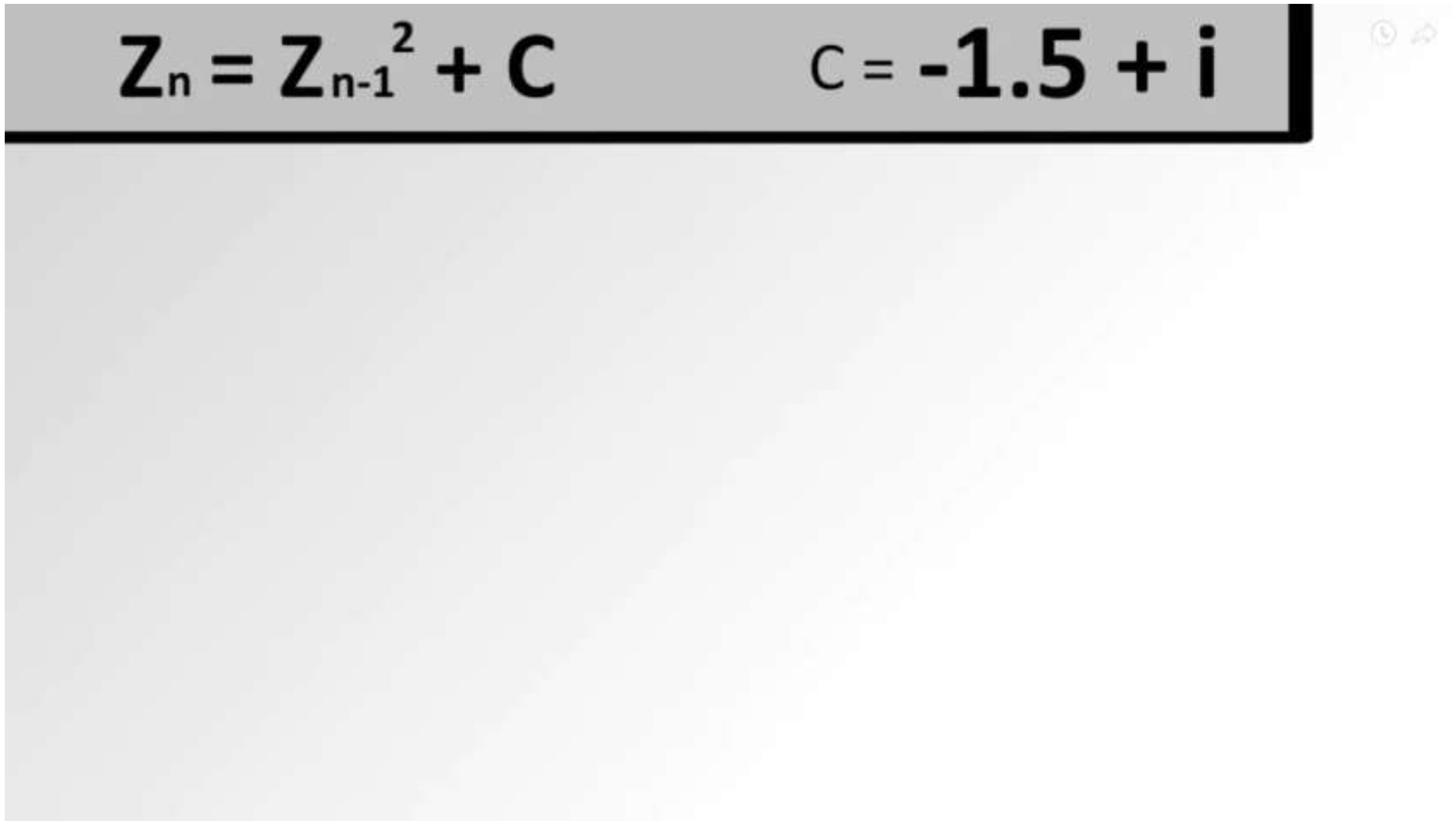
Starting condition: *z_0 = 0*

The Escape Criterion

A point belongs to the Mandelbrot set if its orbit remains bounded—meaning */z_n/* never exceeds 2, no matter how many iterations we perform.

Points that escape yield beautiful boundary patterns.

Math (Mandelbrot)



$Z_n = Z_{n-1}^2 + C$ $C = -1.5 + i$

$Z_n = Z_{n-1}^2 + C$ C = $-1.5 + i$

$Z_0 = 0$

$Z_1 = Z_0^2 + C$

$Z_n = Z_{n-1}^2 + C$ $\quad$ $C = -1.5 + i$

$Z_0 = 0$

$Z_1 = 0^2 + C$

$Z_n = Z_{n-1}^2 + C$ C = $-1.5 + i$

$Z_0 = 0$

$Z_1 = 0^2 + C$

$Z_n = Z_{n-1}^2 + C$ $\qquad$ $C = -1.5 + i$

$Z_0 = 0$

$Z_1 = 0^2 + -1.5 + i = -1.5 + i$

$Z_n = Z_{n-1}^2 + C$ C = $-1.5 + i$

$Z_0 = 0$

$Z_1 = 0^2 + -1.5 + i = -1.5 + i$

$Z_2 = Z_1^2 + C$

$Z_n = Z_{n-1}^2 + C$ $C = -1.5 + i$

$Z_0 = 0$

$Z_1 = 0^2 + -1.5 + i = -1.5 + i$

$Z_2 = (-1.5 + i)^2 + C$

$Z_n = Z_{n-1}^2 + C$ $C = -1.5 + i$

$Z_0 = 0$

$Z_1 = 0^2 + -1.5 + i = -1.5 + i$

$Z_2 = (-1.5 + i)^2 + -1.5 + i$

$Z_n = Z_{n-1}^2 + C$ $\quad C = -1.5 + i$

$Z_0 = 0$

$Z_1 = 0^2 + -1.5 + i = -1.5 + i$

$Z_2 = (-1.5 + i)^2 + -1.5 + i = -0.25 - 2i$

$Z_n = Z_{n-1}^2 + C$ $\quad$ $C = -1.5 + i$

$Z_0 = 0$

$Z_1 = 0^2 + -1.5 + i = -1.5 + i$

$Z_2 = (-1.5 + i)^2 + -1.5 + i = -0.25 - 2i$

$Z_3 = Z_2^2 + C$

$Z_n = Z_{n-1}^2 + C$ $\quad$ $C = -1.5 + i$

$Z_0 = 0$

$Z_1 = 0^2 + -1.5 + i = -1.5 + i$

$Z_2 = (-1.5 + i)^2 + -1.5 + i = -0.25 - 2i$

$Z_3 = (-0.25 - 2i)^2 + C$

$Z_n = Z_{n-1}^2 + C$ $C = -1.5 + i$

$Z_0 = 0$

$Z_1 = 0^2 + -1.5 + i = -1.5 + i$

$Z_2 = (-1.5 + i)^2 + -1.5 + i = -0.25 - 2i$

$Z_3 = (-0.25 - 2i)^2 + -1.5 + i$

$Z_n = Z_{n-1}^2 + C$ $C = -1.5 + i$

$Z_0 = 0$

$Z_1 = 0^2 + -1.5 + i = -1.5 + i$

$Z_2 = (-1.5 + i)^2 + -1.5 + i = -0.25 - 2i$

$Z_3 = (-0.25 - 2i)^2 + -1.5 + i = -5.4375 + 2i$

$Z_n = Z_{n-1}^2 + C$ $C = -1.5 + i$

$Z_4 = (-5.4375 + 2i)^2 + -1.5 + i$

$Z_5 = (24.07 - 20.75i)^2 + -1.5 + i$

$Z_6 = (147.13 - 997.76i)^2 + -1.5 + i$

$Z_7 = (9.74e{+}5 - 2.94e{+}5\ i)^2 + -1.5 + i$

$Z_8 = (8.6e{+}11 + 5.7e{+}11\ i)^2 + -1.5 + i$

$Z_n = Z_{n-1}^2 + C$ $\quad C = -1.5 + i$

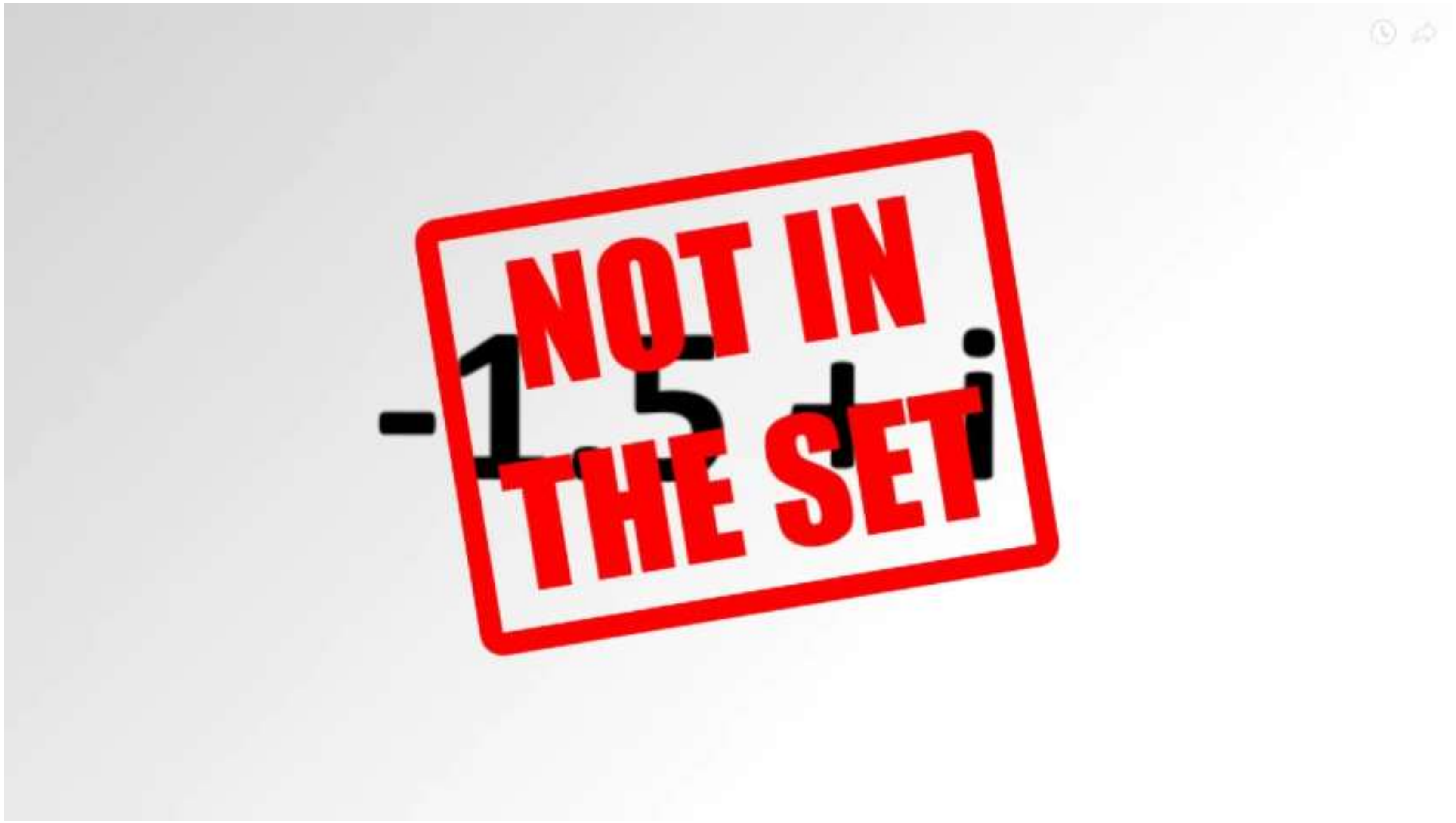
$Z_4 = (-5.4375 + 2i)^2 + -1.5 + i = 24.07 - 20.75i$

$Z_5 = (24.07 - 20.75i)^2 + -1.5 + i = 147.13 - 997.76i$

$Z_6 = (147.13 - 997.76i)^2 + -1.5 + i = 9.74e{+}5 - 2.94e{+}5\,i$

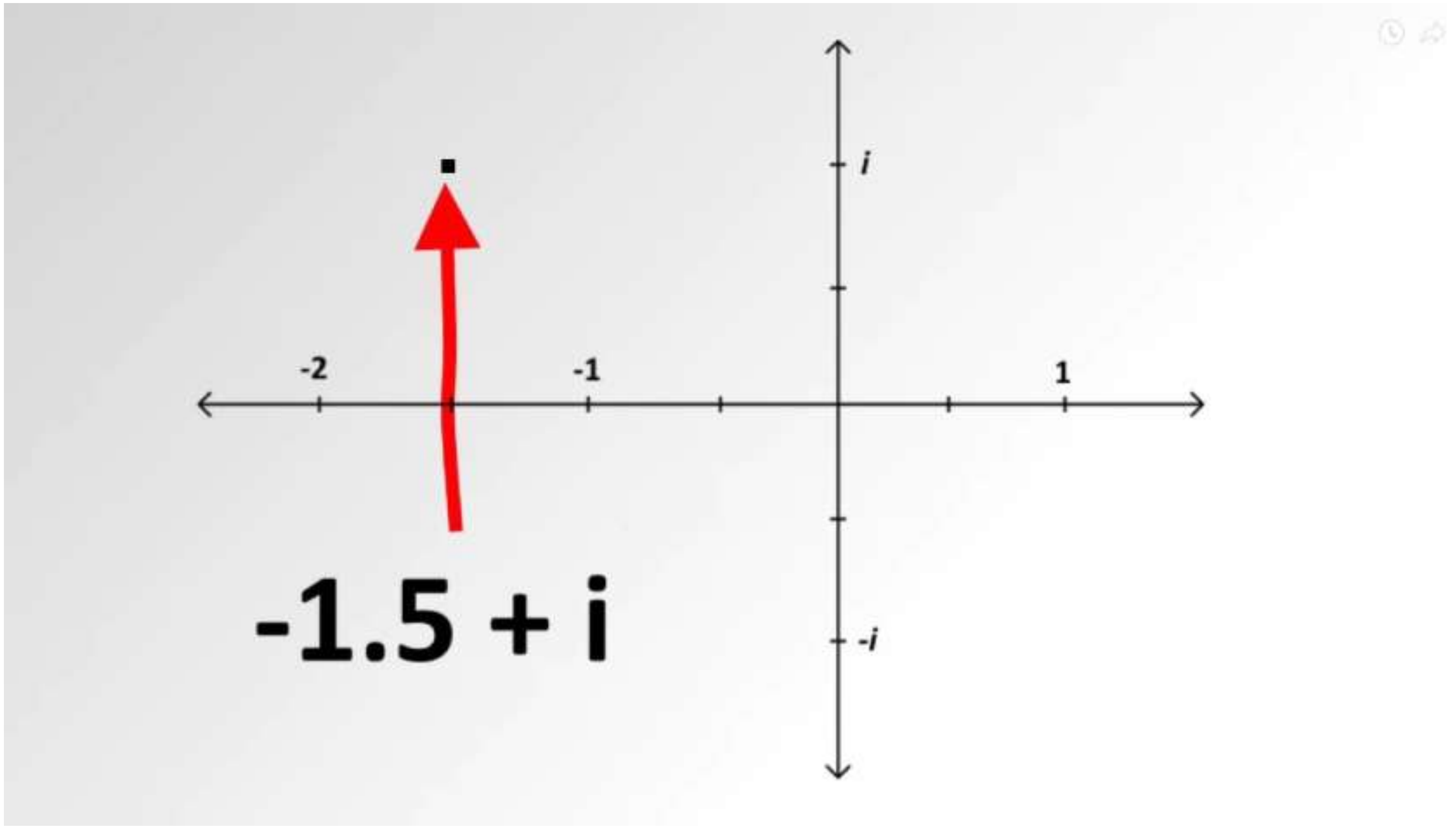
$Z_7 = (9.74e{+}5 - 2.94e{+}5\,i)^2 + -1.5 + i = 8.6e{+}11 + 5.7e{+}11\,i$

$Z_8 = (8.6e{+}11 + 5.7e{+}11\,i)^2 + -1.5 + i = 4.2e{+}23 + 9.9e{+}23\,i$



$-1.5 + i$

NOT IN THE SET


-2
-1
1
i
-i
-1.5 + i

$Z_n = Z_{n-1}^2 + C$ $C = -1$

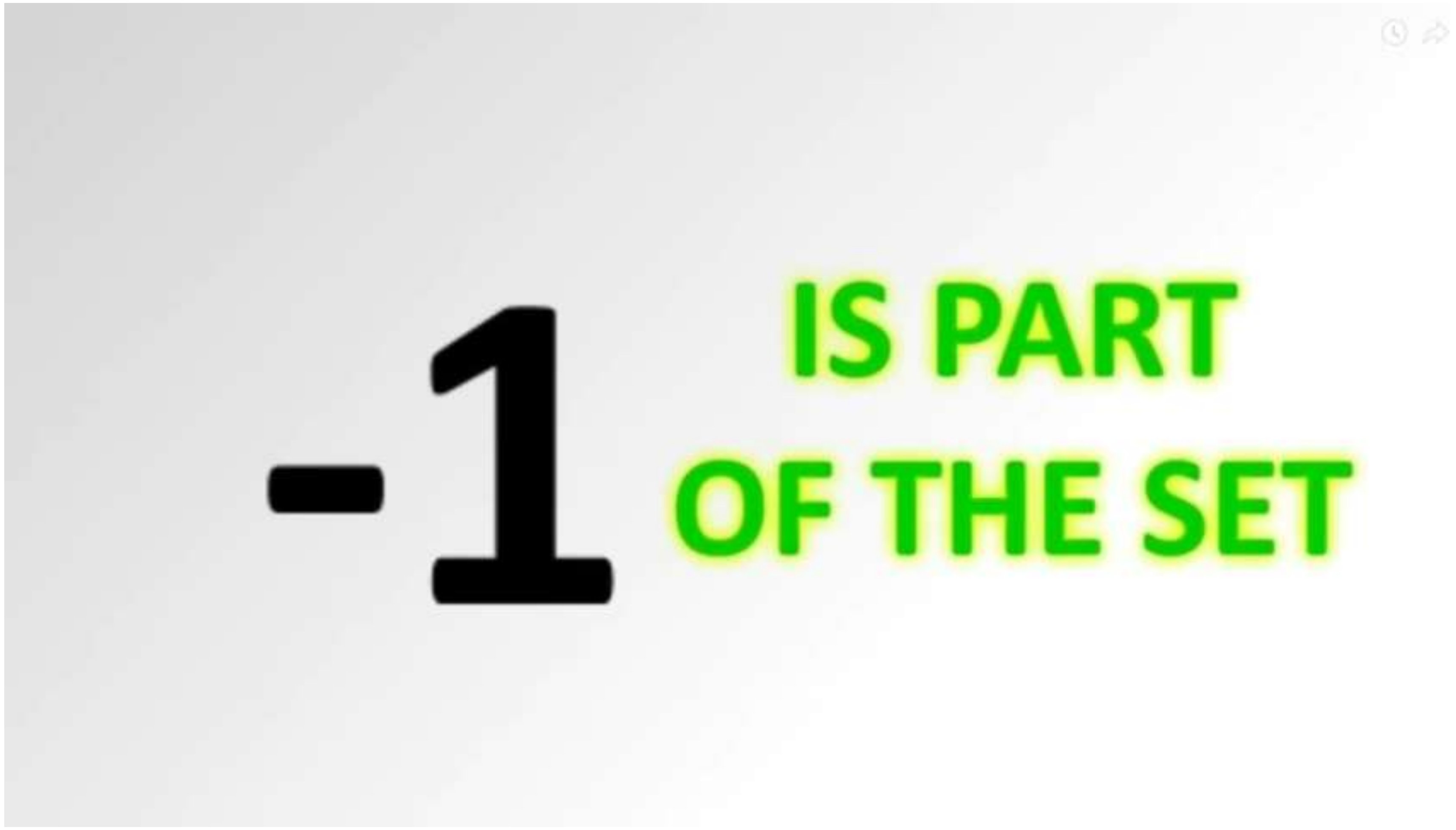
$Z_0 = 0$

$Z_1 = 0^2 + (-1) = -1$

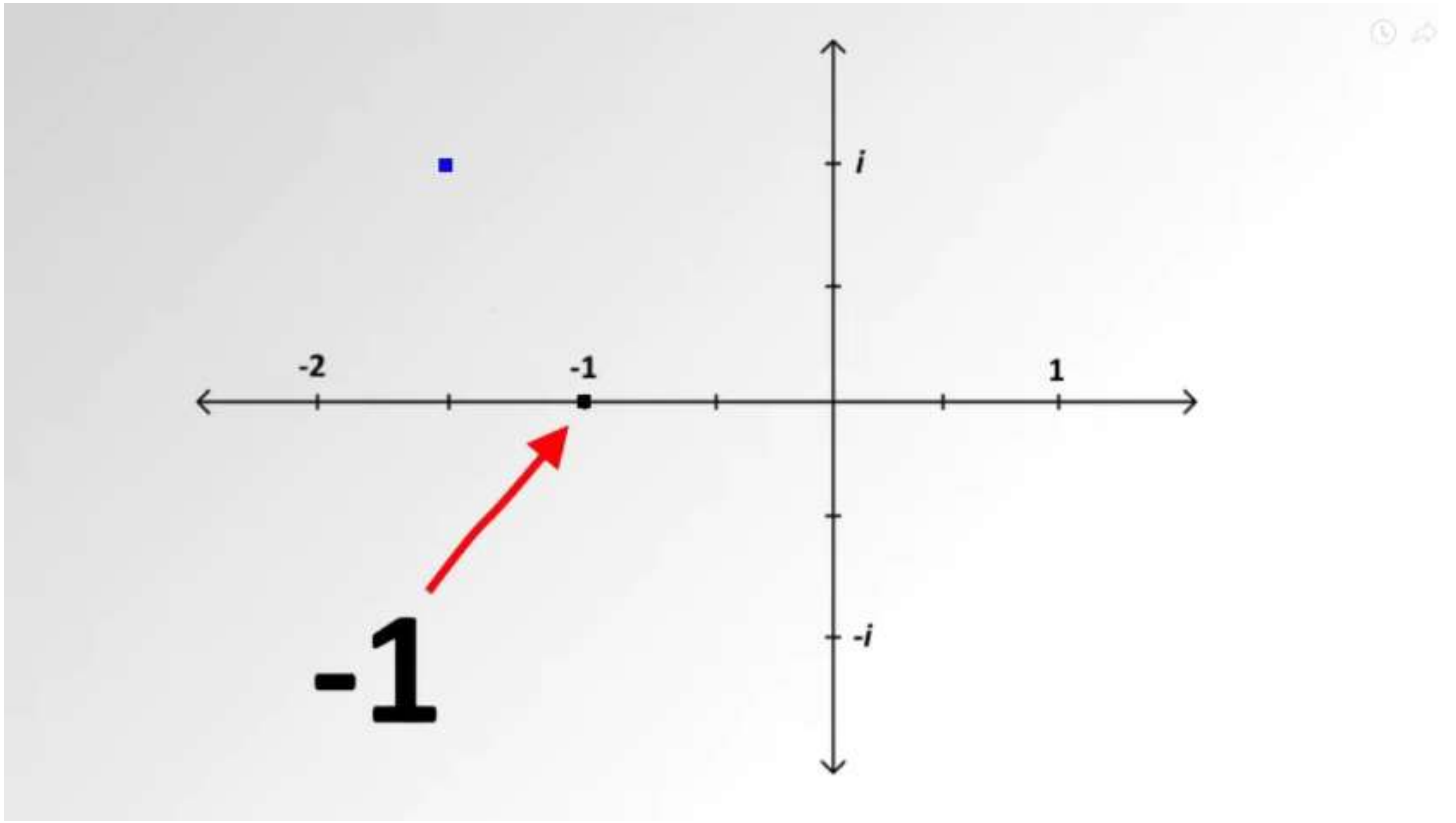
$Z_2 = -1^2 + (-1) = 0$

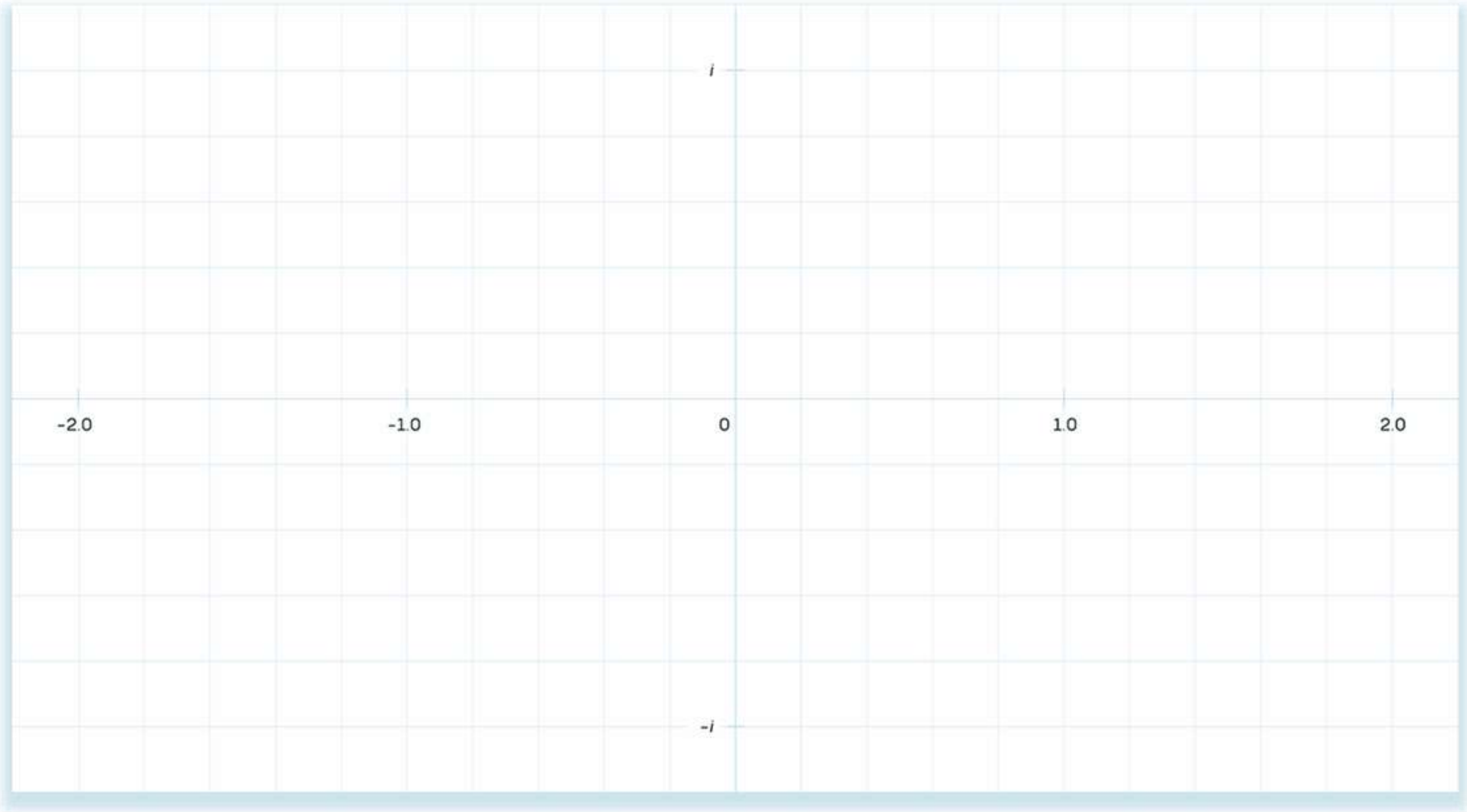
$Z_3 = 0^2 + (-1) = -1$

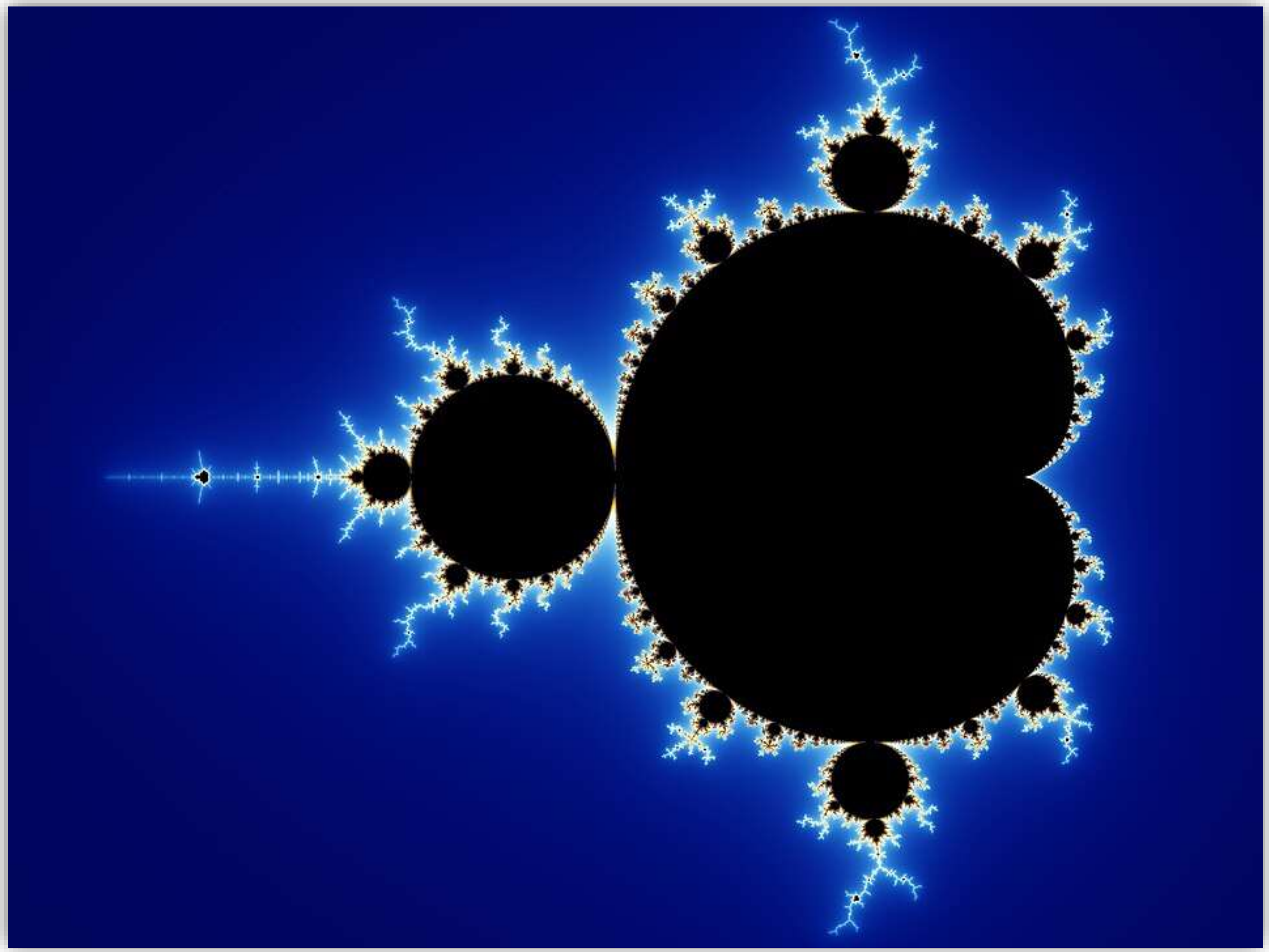
$Z_4 = -1^2 + (-1) = 0$



-1 IS PART OF THE SET


i
-2
-1
1
-i
-1

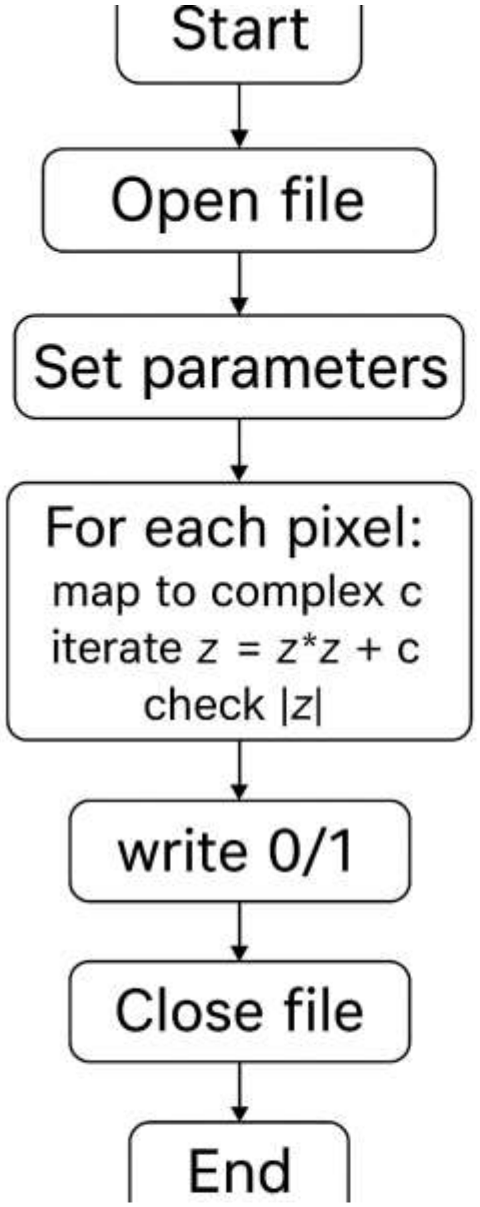

i
-i
-2.0
-1.0
0
1.0
2.0



Algorithm (Flow)

Steps:

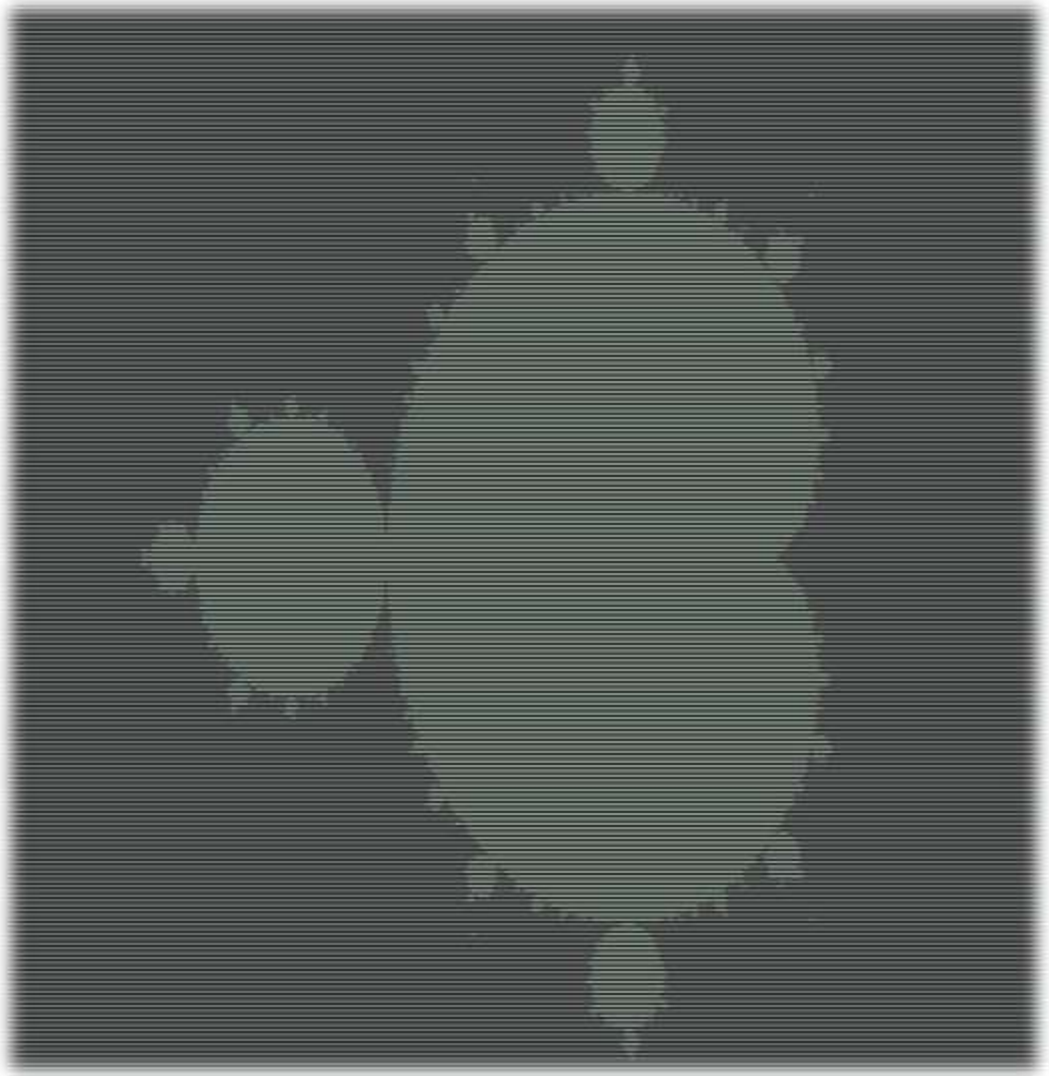
- Initialize parameters — width, height, max iterations.
- Loop over each pixel on the screen.
- Map pixel to complex coordinate (x + yi).
- Apply iteration: $z = z^2 + c$.
- Assign output 0 or 1 depending on escape status..
- Save pixel data to file.

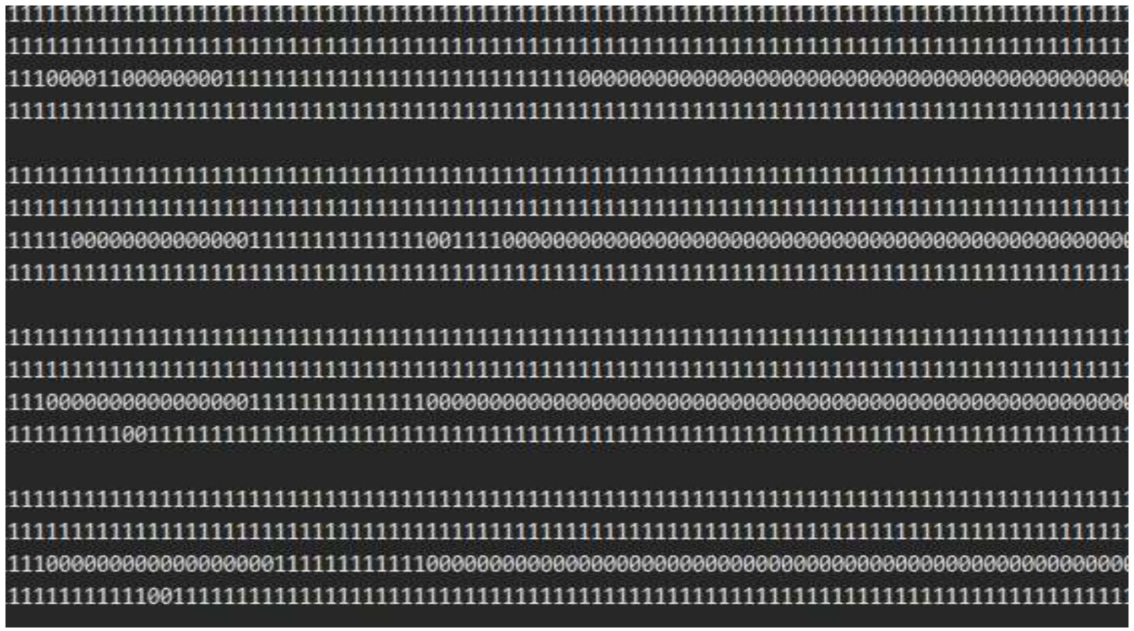


Code Snippet

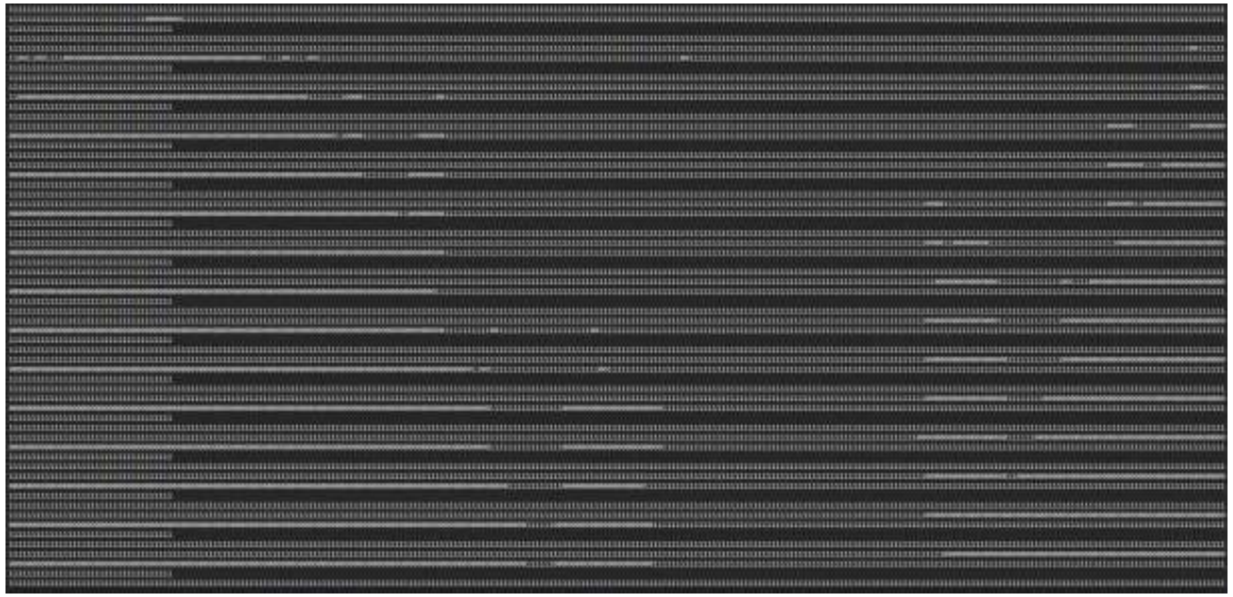
```
// Escape-time function
int mandelbrot(double complex c) {
    double complex z = 0;
    int n = 0;

    while (cabs(z) <= 2.0 && n < MAX_ITER) {
        z = z*z + c; // z = f(z,c)
        n++;
    }
    return (n == MAX_ITER);   // 1 = inside, 0 = outside
}



for (int py = 0; py < HEIGHT; py++) {
    for (int px = 0; px < WIDTH; px++) {

        double x = x_min + (x_max - x_min) * px / (WIDTH - 1);
        double y = y_min + (y_max - y_min) * py / (HEIGHT - 1);

        double complex c = x + y * I;

        int inside = mandelbrot(c);
        fprintf(fp, inside ? "0" : "1");
    }
    fprintf(fp, "\n");
}
```

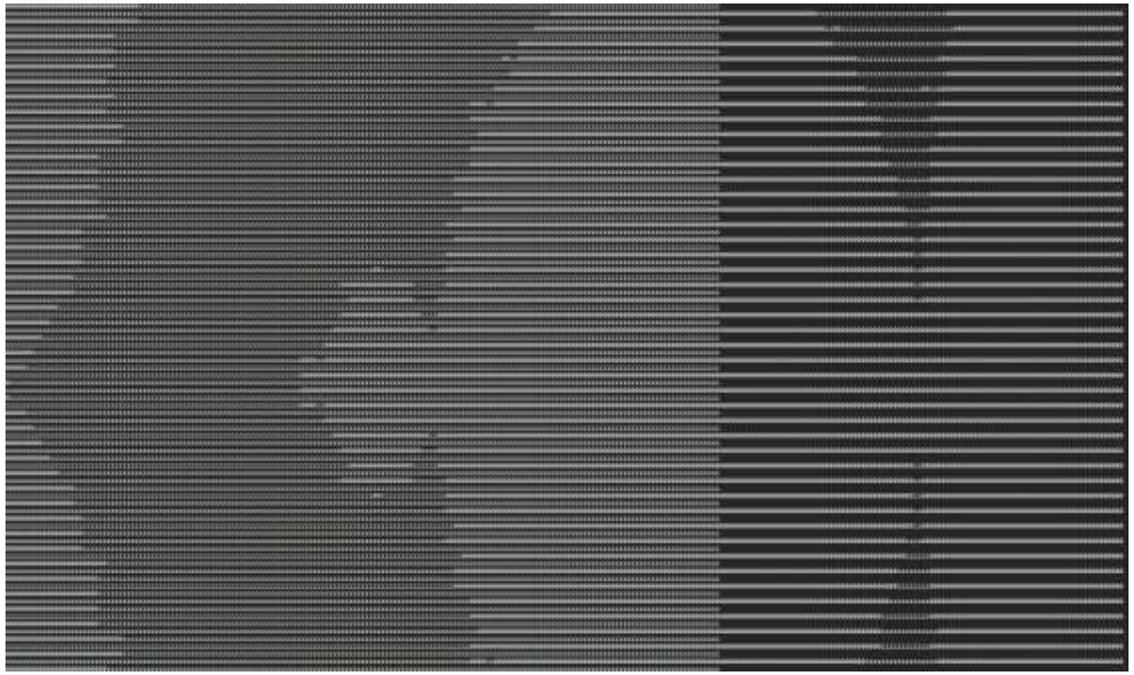
Code Output

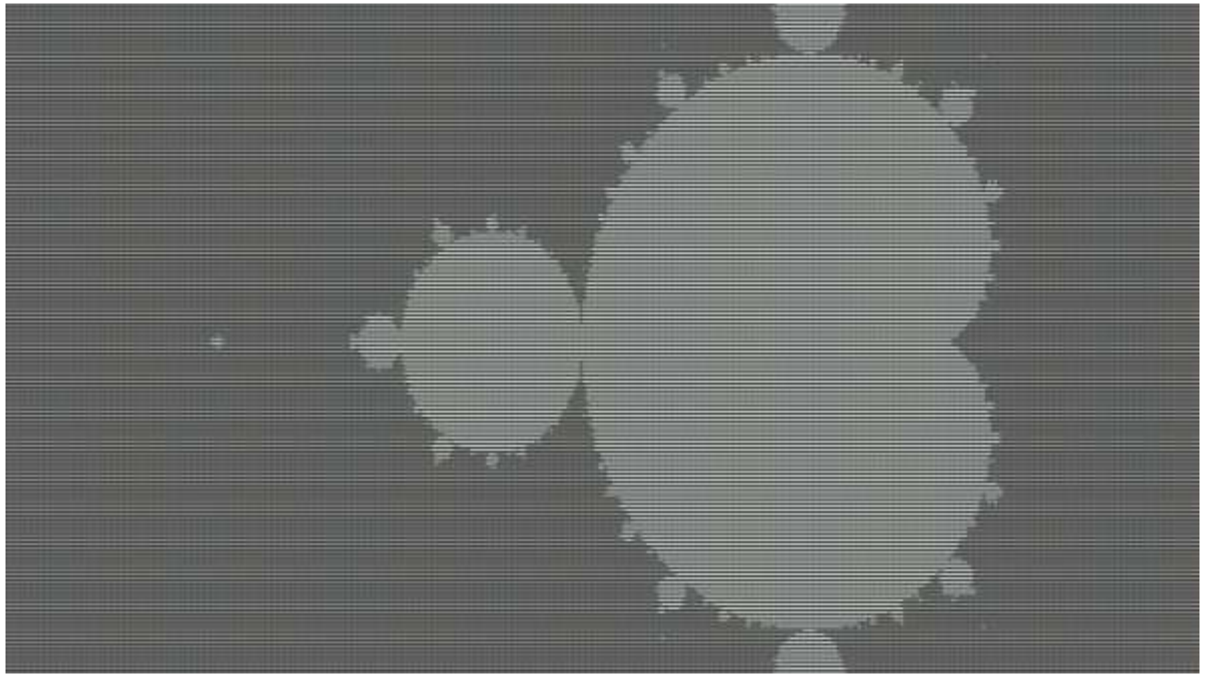




```
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111000011000000000111111111111111111111111111110000000000000000000000000000000000000000000000000
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111

111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111110000000000000001111111111111001110000000000000000000000000000000000000000000000000000000000
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111

111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111000000000000000011111111111110000000000000000000000000000000000000000000000000000000000000000
111111111001111111111111111111111111111111111111111111111111111111111111111111111111111111111111

111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111000000000000000000111111111110000000000000000000000000000000000000000000000000000000000000000
111111111110011111111111111111111111111111111111111111111111111111111111111111111111111111111111
```







Implementation in C

Technical Stack

- Language: C99 standard
- Key headers: `stdio.h`, `complex.h`
- Output format: TXT (text file)
- Build command:

```
gcc -o mandelbrot mandelbrot.c -lm
gcc -o julia julia.c -lm
gcc -o tricorn tricorn.c -lm
```

Output Generation

- Output file: mandelbrot.txt
- Image Resolution: 420 × 236 pixels.
- “0”, inside Mandelbrot set, “1”, outside.

Other Fractals (Extension)

- Experimented with:
  - **Julia Set** → zn+1=zn2+kz_{n+1} = z_n^2 + kzn+1=zn2+k
  - **Tricorn Set** → zn+1=zn¯2+cz_{n+1} = \bar{z_n}^2 + czn+1=zn¯2+c
- Observed distinct fractal patterns.
- Demonstrated flexibility of code for modification.

Julia and Tricorn: Fascinating Variants

Julia Sets

The Julia set uses the same quadratic iteration but with a crucial difference:

$$z_{n+1} = z_n^2 + c$$

Here, *c* is a fixed complex constant whilst *z_0* equals the pixel coordinate. Different values of *c* produce dramatically different fractal shapes—from connected dragons to scattered dust.

The Tricorn (Mandelbar)

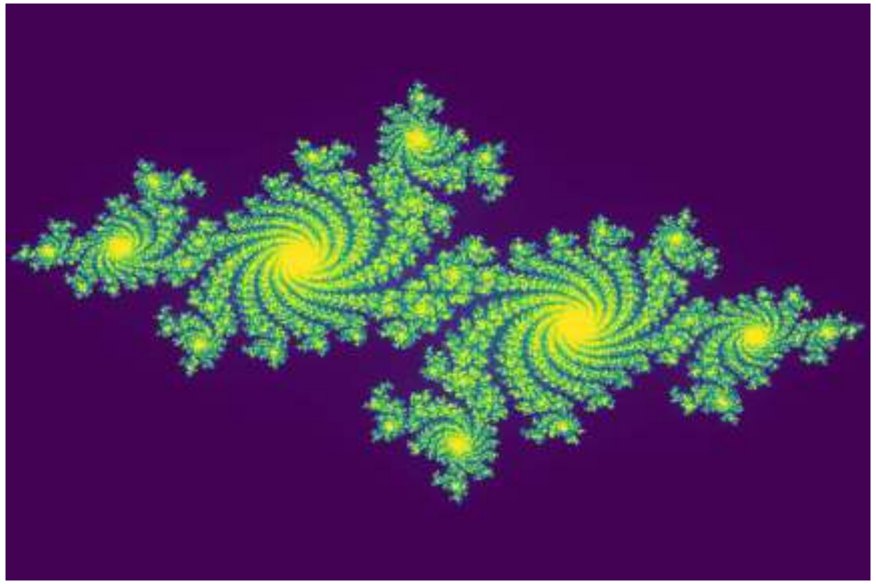
This variant introduces the complex conjugate, creating unique symmetry:

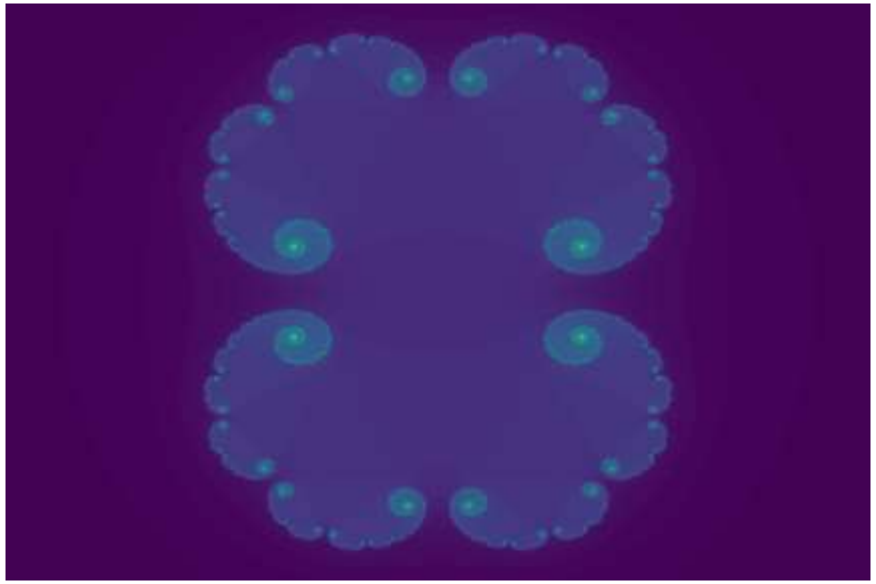
$$z_{n+1} = \overline{z_n}^2 + c$$

The conjugate operation reflects points across the real axis, generating distinctive heart-shaped features and altered self-similarity patterns.

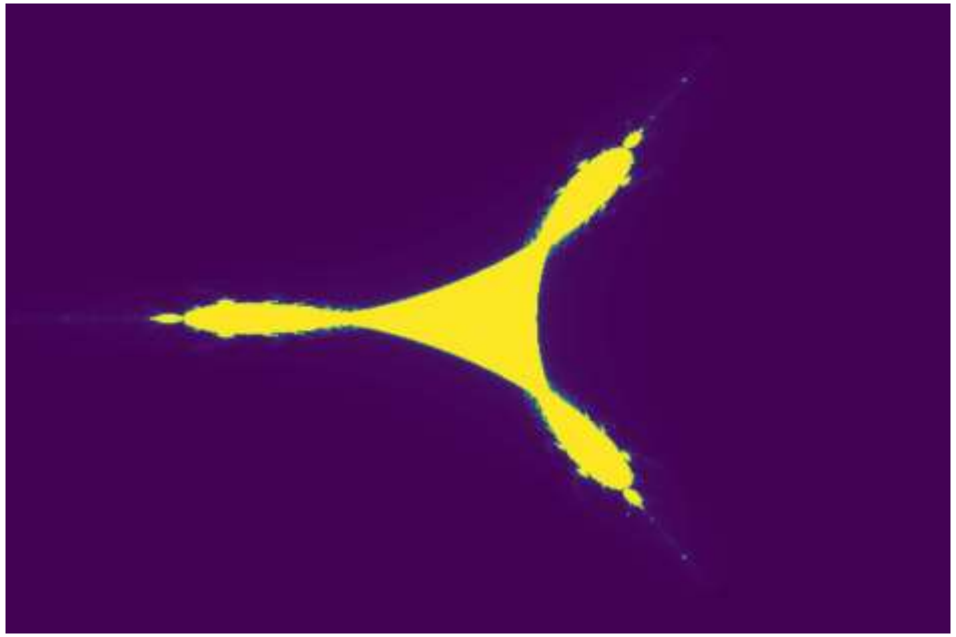
Escape condition: For all three sets, we test whether */z/ > 2*. If this occurs before MAX_ITER iterations, the point escapes. Otherwise, it belongs to the set's interior .

Results: Julia





Results: Tricorn



Testing & Debugging

- Verified for different resolutions (400×400, 800×800).
- Tested with varying MAX_ITER values.
- Fixed:
- Overflow of complex numbers.
- File writing errors.
- Result validation with expected fractal shape.

Conclusion & Future Scope

Conclusion:

- Project achieved accurate fractal generation.
- Showed mathematical and programming integration.

Future Scope:

- Zoom & interactive GUI.
- Parallel GPU rendering.
- Real-time fractal animation.

Results & Analysis

- Successfully visualized Mandelbrot fractal.
- Observed:
  - Infinite self-similarity.
  - Complex boundary structures.
  - Dependence on iteration depth.
- Demonstrated link between math and art.

References

- 1. Wikipedia: Plotting algorithms for the Mandelbrot set
- 2. LibreTexts: Julia and fractals
- 3. Project notes